

SESSION

UNIFIED MODELING LANGUAGE (UML), OBJECT ORIENTED METHODS, AND CASE STUDIES

Chair(s)

TBA

UML Model Based Design of the Claw Car Robot

Andre Layne¹, Adria Mason¹, Yujian Fu¹, Mezemir Wagaw²

¹Department of Electrical Engineering & Computer Science,

²Department of Food & Science,

Alabama A&M University, Normal, AL, United States

Abstract - Robots are intricate systems and applied in many aspects of today's society. It is highly desirable to design and develop robust robotics systems. This paper aims at developing an autonomous robotic system using object-oriented software development (OOSD) methodology – UML – to ensure the quality of the system. Unified Modeling Language (UML), a typical OOSD method, is a standard visualization language for object-oriented system development that has been widely used in the design of safety critical and mission critical systems (e.g. aeronautic systems, missile defense systems, etc). In this paper, the UML class diagram is used to represent the static structure and relations of objects. The dynamic behavior is modeled in the state machine diagram. A case study is performed in the LEGO NXT tool kit, a low cost highly integrated educational robot setting, in Java programming language. In this case study, a Claw Car Robot is designed and assembled with the function of continuously moving forward and stop on the color definition. The robot can close the claw and clasp the object in its path once an object is detected. The LEGO NXT tool kit includes multiple sensors and supported by several platforms including Java and C++. This LEGO NXT tool kit is very convenient for the design and implementation of robotics systems, and has been widely adopted in institutions for educational and initial research purposes.

Keywords: UML; Autonomous; NXT; Object-Oriented Programming Language

1 Introduction

The robot population is doubling every few years. According to IEEE Spectrum [6], the world population of robots had reached over 4.5 million at the end of 2006 and 8.5 million at the end of 2008. The size and growth of these numbers show that robots contribute to a very important role in our society today. These robots use various types of integrated technologies to achieve specific goals in various types of environment. Therefore, reliability and quality of the robotics system is becoming more and more important. However, there is not enough research on the building of reliable robot systems. In this research work, we developed the UML model of the robotics system, and then implemented in Java, based on the assembled LEGO NXT tool kit.

A typical feature of robotic systems is multiple interfaces and multiple objects. To provide flexible functionality, a robot

usually integrates with several different types of sensors to get the data from its environment as well as multiple devices for different purposes such as arms, claws and some other tools.

All these integrated sensors and devices apply their own way to read and handle information, which is defined by various APIs. Therefore, designing and the programming of the multiple interfaces, and also integrating them to the system smoothly, are challenges for the robotics design. Any miscommunication between controller and sensors or devices may cause unexpected results and huge losses. It is key to build and develop reliable and quality software for the robotics systems.

Object-oriented software development (OOSD) methodology has been widely used in the design of safety critical and mission critical systems (e.g. aeronautic systems, missile defense systems, etc). Unified Modeling Language (UML) [9, 10, 11], a typical OOSD method, is a standard visualized language for object-oriented system development. In this paper, we proposed a UML 2.0 model that includes the class diagram and state machine diagram on the robotics systems to specify the system design requirements, and use Object Constraint Language (OCL) to define the desired properties. Therefore, the object-oriented design model of a robotics system includes three components in general – static structure, dynamic behavior, and property specification. This paper presented the UML based model of the robotics system that is represented by the above three components.

This work is implemented in an assembled claw car using LEGO NXT tool kit and implemented with the Java programming language. The Mindstorm NXT brick uses a 32-bit ARM processor as its main processor, with 256 kilobytes of flash memory available for program storage and 64 kilobytes of RAM for data storage during program execution. To acquire data from the input sensors, another processor is included that has 4 kilobytes of flash memory and 512 bytes of RAM. Two motors can be synchronized as a drive unit. To give the robot the ability to “see,” the ultrasonic sensor, which is accurate to 3 centimeters and can measure up to 255 centimeters, and the light sensor, which can distinguish between light and dark, can be attached to the brick. Finally, the two touch sensors give the ability for a robot to determine if it has been pressed, released, or bumped, and react accordingly [12].

This paper is organized as follows. Section 2 introduces the background knowledge used in this work. Section 3 presents the hardware assembly and functions for the LEGO robot. Section 4 shows the UML model of the robotics

systems. Section 5 presented the Java implemented with the LeJOS package. Section 6 discusses the results and the conclusion.

2 Background

In the Object-Oriented Software Development (OOSD) approach, the system is viewed as a collection of multiple objects and with various interfaces definitions. The functionality of the system is achieved by the interaction and communication among these objects through messages. The Unified Modeling Language – UML – is developed on the above principles and widely used in the complex embedded system and large scale software intensive system development. In this paper, we presented a UML based model driven architecture for the robotics design that includes three components – static structure (class diagram), dynamic structure (state machine diagram) and property specification (OCL). Therefore, we simply introduce each component in the UML syntax.

UML defines twelve types of diagrams which fall into three categories [9, 10]: (i) *Structural Diagrams* which include the Class Diagram, Object Diagram, Component Diagram, and Deployment Diagram, focus on the static organization of instance of the system; (ii) *Behavior Diagrams* which include the Use Case Diagram, Activity Diagram, Communication Diagram, and State Machine Diagram, focus on the functions and collaborations among instances; and (iii) *Model Management Diagrams* which include Packages, Components, and Subsystems, focus on the packaging and setting of diagrams. UML has been used across a wide variety of domains, from computational to physical, making it suitable for specifying systems independently of whether the implementation is accomplished via software or hardware. Since UML was initially introduced in the software domain, most commercial tools based on UML descriptions have the ability of generating software code, such as Java and C++.

However, no such tools are commercially available that can design and synthesize UML models into a model for robotics system model directly, thus imposing a limitation for the usage of UML in robotics system design. Additionally, it is also observed that assuring correct functional behavior is the dominating factor of a successful hardware design. It shows that up to 80% of the overall circuit design costs are due to verification tasks. Assurance of quality of robotics is a key issue now.

To complement the UML diagrammatic notation, the Object Constraint Language (OCL) [14, 15] can be used to express constraints and specify the effect of operations in a declarative way. In each predicate of OCL, the logical statements must be satisfied by all valid instances of the system that are represented by constraints.

The OCL [15, 14, 13] is a textual, declarative language based on first-order logic and set theory. In addition to expressing constraints on class diagrams, OCL can also be used to specify the effect of the execution of an operation, using pre and post conditions. A pre condition is an OCL statement that has to evaluate to true before the execution of

an operation, while a post condition is a statement that has to evaluate to true when the operation terminates.

3 Lego Robot & Functionalities

The LEGO Mindstorm tool kit is composed of five external sensors and three motors except for many other pieces that give the physical design and construction of the robot flexibility. In this section, we introduce the assembly and functionalities of the LEGO robot claw car design as well as the challenge issues during the development.

3.1 Functionalities

The Claw Car Robot needs to move forward on the color - white. However, once the color - black is detected, the robot will close the claw and clasp the object in its path. The robot will also slow down once black is detected and the claw is closed. As the Claw Car Robot passes over white again, the robot continues in this same state until the color - black is detected again. Once black is detected again, the robot will open the claw and release the object. In addition, almost simultaneously, the Claw Car Robot will end its program, as designed. In this work, each sensor has its own API. The NXT needs to be able to work with the sensors properly to realize a stable and reliable system. All these factors point to a strong need to maximize software and system development productivity through the use of embedded system platforms, reuse, and synthesis methods driven from system-level models.

3.2 Hardware Development

In the robotics community, most robots manipulate objects using what is called an End of Arm Tool (EOAT) [5]. The most common type of EOAT is the robotic gripper. These grippers come in various shapes and sizes. There are two different categories of robotic grippers which are friction and encompassing robotic grippers. Friction robotic grippers are used to hold objects by using force only. Encompassing robotic grippers surround objects to grasp them and do not use much force at all. To make a decision on which type of gripper we would use, we had to take into consideration the material we would build the object with as well as the type and size of the objects this robot would manipulate. Since our Robot has been built with light-weight materials, it will be used to pick up light-weight and flexible objects. An encompassing style gripper shown in figure 1 would be the best type to use.



Figure 1. A robotic gripper from NXTPrograms.com

3.3 Challenge Issues

It is not hard to build a mobile robot with the above expected functionalities using the LEGO NXT Mindstorm toolkit. The challenge issue is how to build a reliable robot to satisfy the expected properties and maintain the stable behaviors as required. For instance, a light sensor is used to detect light values given off from the surface beneath the robot. One problem is how to make sure the claw car can detect light values efficiently, and manipulate objects in its environment according to the light values detected. We needed an approach that can efficiently respond to light values of the terrain beneath the robot, and perform object manipulation based on these values. In sum, the key issue is how to develop a robotics system with respect to the user requirements, minimize risk, maintain correct behavior, and improve quality.

The unreliability issue comes from two aspects in the LEGO robot design from our study. One is from the imprecision of the sensors. The LEGO kit is used for the educational purpose and many sensors do not have high precision to reflect the required value. For instance, the responded value for white color and light yellow color can vary between 49 to 52, depending on the lighting condition. Another major issue is from the control software design, which is the one this research focuses on. For instance, even though we give enough space for the white value to be changed, the system may still do not maintain stable status due to internal or external stimuli. For instance, the light sensor may stop working. If there is no other way to fix this sensor, the robot will fail the duty. To solve that problem, we included an alternative light sensor and require the system to continue working with the same behavior, if one of these two sensors fails.

3.4 Redundancy Design

We needed a redundant sensor that will be able to perform the work needed if the default sensor was down. To mitigate this concern we added a secondary, or back up sensor that will become active if the default or primary sensor isn't available. This back up sensor is working during the robot running time. Which means the back up sensor is detecting color, responding to the NXT brick, and maintains the same duty as the default sensor. Once the default light sensor is unplugged, the back up sensor can automatically resume duty without any interruption of the system. Figure 2 displays the image of the robot with redundant sensors. In Figure 2(a), the robot is driving with both sensors working. In Figure 2(b), the default sensor is unplugged and the back up sensor is taking control after the primary sensor isn't functioning.

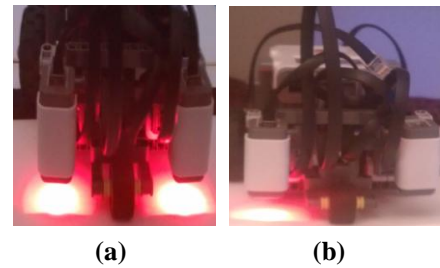


Figure 2. View of default & secondary light System Design

Initially proposed as a unifying notation for object-oriented design, UML has added a semantic underpinning that makes it possible to build platform independent descriptions that can be used by designers and architects to make informed decisions about hardware/software tradeoffs. We present UML based architecture for the robotics system design to ensure the quality of the robot. The architecture is composed of three components (Figure 3.) – static structure (represented by class diagram), dynamic structure (represented by communication diagram or state machine diagram), and system properties (specified by OCL).

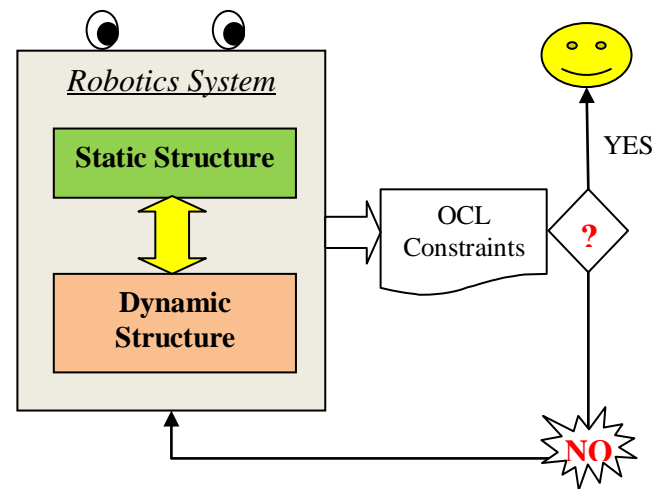


Figure 3. UML based robotics system architecture

The OCL specification can describe all desired constraints on the static structure and dynamic structure. From Figure 3, we can see that is the OCL properties are not satisfied, we can go back to the model architecture and find out what is the problem. After the properties and constraints are ensured, then the system can be implemented based on the model. The verification of OCL properties can be done by software testing tools or model checking techniques.

In the following, we use the LEGO claw car robot to illustrate the UML architecture model.

4.1 Class Diagram

According to Michael Blaha and James Rumbaugh [1], a “class diagram provides a graphic notation for modeling

classes and their relationships, thereby describing possible objects. Class diagrams are useful both for abstract modeling and for designing actual programs.” The class diagram of this LEGO claw car robot is shown in Figure 4.

The Claw Car Robot is a robot that drives forward, and is composed of aggregate parts; motors, a light sensor, and a backup light sensor. To represent the above function, the Class diagram shows this relationship as a model with each one of the real world objects represented appropriately. The FinalClawCar Class is the container class for the remaining classes. Therefore, Figure 3 shows the FinalClawCar Class with an aggregation relationship to the other classes.

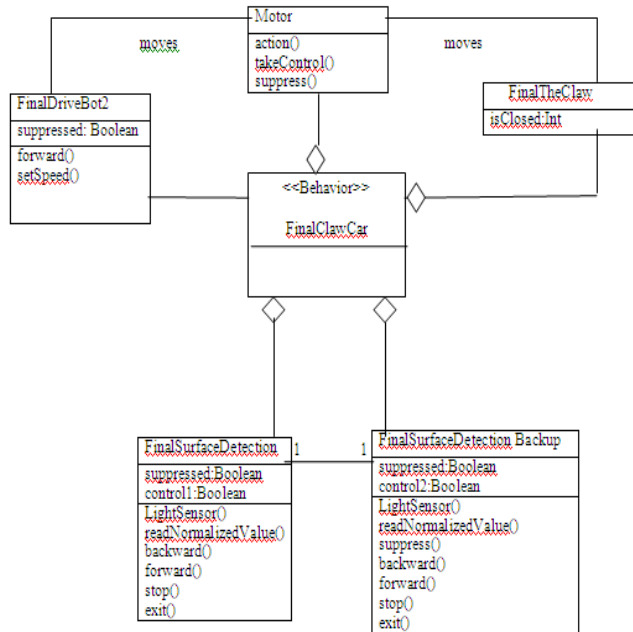


Figure 4. Class Diagram of the Claw Car Robot

4.2 State Machine Diagram

The dynamic behaviors are represented by the State Diagram, which includes state, transition and events. A state is an abstraction of the values and links of an object. An event is an occurrence at a point in time. Events represent points in time. States represent intervals of time. [1]. Before implementing our actual code, it was necessary to first provide a full description of what each object would do in response to different events.

The possible states, transitions, and events of the Claw Car Robot are shown in Figure 5. The black dot represents the entry into the diagram. In this case, the object is originally in the idle state until the Enter button is pressed on the NXT Brick, and the robot begins to drive forward. While the robot is in the ‘Moving Forward’ state, it responds to two possible events. In this diagram, either the robot detects black for the first time, which in this case, it moves into a claw close state. If the color black is detected a second time the claw opens, releasing any obtained object. The challenge issue here is the claw cannot close if there is no object in hand. However, how

to detect the claw holding the object is a challenge. Therefore, we reduce the question to a simple case – a) first, detect the object; b) once the object is detected, the car moves and pick up the object; and the claw closes. From the state machine diagram in Figure 5, you can see the car releasing the object without detecting if there is object holding. But, the robot will check the line color before releasing the object (Figure 5).

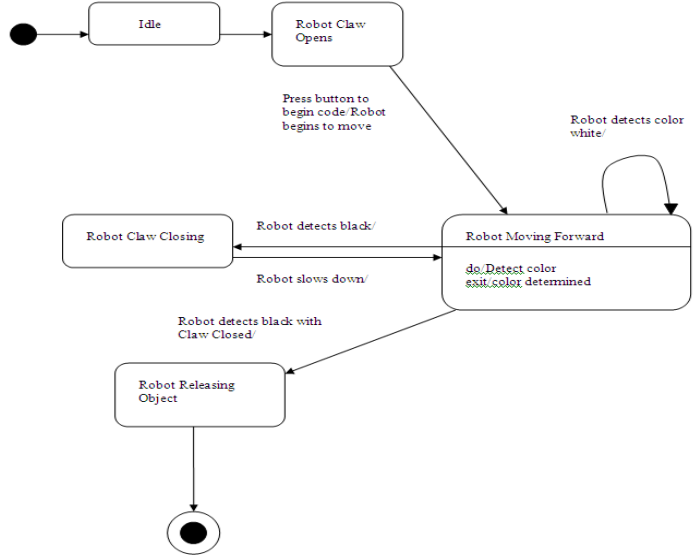


Figure 5. State machine diagram of the Claw Car Robot

4.3 Quality Of Robot – Object Constraint Language

UML provides a formal language to specify and express the constraints within a system, named the Object Constraint Language (OCL) [8]. A constraint restricts the values that elements can assume. We utilized the OCL 2.0 version to specify the constraints within this system. Several constraints are defined for the system on classes. For example, for the class Motor, we define an invariant as if the speed is larger than zero, then the robot is moving (shown in formula (b)). As discussed before, the claw open and close checking is critical to grab the object. Therefore, we have defined the invariant for the claw close checking (shown in formula (f)).

Context: Motor

INV: Motor.forward *implies* Motor.isMoving (a)

INV: Motor.setSpeed() >0 *implies* Motor.isMoving (b)

Pre: Motor.stop() (c)

Post: Motor.isMoving = false (d)

Context: LightSensor

INV: LightSensor.getLightValue() >330 *implies* LightSensor.isWhite (e)

Context: FinalTheClaw

INV: FinalTheClaw.isClosed= = 0 *implies*
FinalTheClaw.isOpen (f)

5 Robot Implementation In Java

This robot is implemented in Java, a typical object-oriented programming language (OOPL). There are several reasons to choose Java as the choice coding language.

First, Java is considered a pure OOPL with typical OO features, including encapsulation, inheritance, polymorphism, besides objects and classes. In Java, object communication is defined by response to classes on messages. When a certain method is executed in response to a message to another object, this method can generate new information that can be a message. Therefore, Java is a perfect implementation language of UML model. Secondly, currently, there is not enough LEGO NXT projects that are implemented in Java. Most of LEGO projects are used for the motivation of high school students and the demonstration of robotics. Therefore, the implementation is mainly reduced to the simple diagram based language such as LabView. However, there is no space for the students to explore the design issues and quality assurance of the software control systems. The last reason is LEGO Mindstorm Kit provides free platform for Java applications named LeJOS [16].

Most robotics implementation uses Behavior Programming (BP), which is supported by the LeJOS package. The important aspect of BP is sequential ordering of the concurrent behaviors issued from multiple objects of a robot. In other words, BP uses sequence to implement concurrence. The Behavior interface is located in a package called Robotics.Subsumption. The Subsumption package also provides a class that handles the Behavior objects called an arbitrator. The Behavior interface provide three methods that allow the code to work in a more logical manner than writing a large amount of if-else statements, which LejosSourceForge.com calls “spaghetti code”. Instead of our code being tangled with all if-else statements, we were able to take a behavioral approach and write the application so that each method and function performed would work in a logical sequence.

The Behavior interface uses three methods to provide a seamless, behavioral interaction between multiple objects. The methods are; takeControl(), suppress(), and action(). These three methods are described in further detail below:

suppress() - The suppress method returns true whenever a specific object doesn't want to take control, or when the object's takeControl() method is false.

takeControl() - An object's takeControl() method returns a Boolean value whenever it reaches a condition where it can return a true value. When this occurs, the object's action() function will perform some action according to its priority level. The suppress method is turned false, and all other Behavior objects should remain suppressed.

action() - when an object's suppress() method returns a false value and the takeControl condition returns true, the code that is within the action() method is performed.

The implementation of the robot is based on the UML model and maintains the constraints defined in the OCL. In this section, we illustrate the Java implementation on the LEGO NXT Mindstorm tool kit on each class. All the classes are defined in Robotic.Subsumption package and Behavior Interface and discussed in the following.

The FinalDriveBot2 Class (Figure 6) is responsible for controlling how and when the claw car robot drives. Since the robot should always want to drive, we initially set the suppress and takeControl functions to false. Although the FinalDriveBot2's suppress() method is set to false, if any higher level priority object's takeControl() method returns true, the FinalDriveBot2's suppress() method will set to the true value. Once all other object's takeControl() method returns false, FinalDriveBot2 will resume.

The FinalSurfaceDetection Class (Figure 7) utilizes the LightSensor Class. Through a method called readNormalizedValue, the FinalSurfaceDetection Class will detect light values reflected from the surface beneath the Robot. This class is set to take control if it detects the color black or a light value above 330. Once this class takes control, it will check the value of a static variable named FinalTheClaw.isClosed. If isClosed is set to 0, the robots claw will close and change the value of FinalTheClaw.isClosed to 1. If black is detected and the value of FinalTheClaw.isClosed is 1, then the claw will open. The logic behind this is to provide the robot with a flag to indicate whether the claw is in an opened or closed state.

The FnalSurfaceDetectectionBackup Class (Figure 8) provides a 2nd LightSensor object for the default sensor. The code is very similar to the FinalSurfaceDetection Class since it will be checking for most of the same conditions. This will allow redundancy between the two sensors. The design is set up so that if one sensor stops working, the other sensor will resume the work without any downtime. The takeControl() method for this class is set so that it will only takeControl if black is detected and sensor1 is returning 0 (indicating sensor 1 is not functioning). Since there is a static variable set up in the FinalTheClaw Class, to represent the state of the claw, the second sensor will know the current state of the claw, and therefore will be able to make a logical decision on the next state of the claw.

The FinalDriveBot2 class consists of only one static class variable. This variable is isOpen(). The isOpen variable is used by the FinalSurfaceDetection and the FinalSurfaceDetectionBackup class to determine whether the claw is in an open, or closed state.

```
lejos.nxt.Motor;
import lejos.robotics.subsumption.Behavior;
public class FinalDriveBot2 implements Behavior{
private boolean suppressed = false;
public boolean takeControl() {return true;}
public void suppress() {suppressed = true;}
public void action() {
suppressed = false;
Motor.A.forward();
Motor.C.forward();
while(!suppressed) Thread.yield();
Motor.A.setSpeed(100);
Motor.C.setSpeed(100); } }
```

```

public class FinalSurfaceDetection implements Behavior {
    LightSensor light = new LightSensor
        (SensorPort.S1); //default light Sensor object
    boolean suppressed = false;
    public boolean takeControl(){
        // this function will take control
        //if a light value is returned less than 330
        //but not equal to 0
        boolean control1=false;
        if(light.readNormalizedValue() < 330
            && light.readNormalizedValue() > 200){
            control1=true;}
        return control1; }
    public void action() {
        suppressed = false;
        if( FinalTheClaw.isClosed == 0 ){
            Motor.B.backward();// close claw
            FinalTheClaw.isClosed = 1; //flag
            //used to indicate if and object has been
            //grabbed
            while(Motor.B.isMoving() )
                Thread.yield(); }
        else { Motor.B.forward();//open claw
            Motor.A.stop();
            Motor.C.stop();
            System.exit(0);// exit program
            while(Motor.B.isMoving() )
                Thread.yield();
            }
        }
    }
}

```

Figure 7. The code of class FinalSurfaceDetection

```

public class FinalSurfaceDetectionBackup implements
Behavior{
    LightSensor light1 = new
        LightSensor(SensorPort.S1);
    LightSensor light2 = new
    LightSensor(SensorPort.S3);
    boolean suppressed = false;
    public boolean takeControl(){
        boolean control2=false;
        if(light2.readNormalizedValue()< 330 &&
            light1.readNormalizedValue() ==0)
            { control2 = true;}
        return control2; }
    public void action() {
        suppressed = false;
        if( FinalTheClaw.isClosed == 0 ){
            .....
        } else {
            .....
            while(Motor.B.isMoving())
                Thread.yield();
            } } }
}

```

Figure 8. The code of class FinalSurfaceDetectionBackup

5.1 Integrated Technologies

This robot was integrated with different technologies to make it successful in its proposed goal. Light Sensor technology was used to sense and determine light values. The Robotic Motor, or mechanical motor technology, was used to mobilize our robot as well as give the gripper functionality. Lego's NXT brick was used as a processor to perform logical decisions and calculations commanded by the downloaded software application. Each one of these technologies worked together in this robot to perform each required action as efficient and seamless as possible.

5.2 Robot Behavior Validation

The system validation is done by checking if the robot meets all requirements and constraints specified. Initially, there were some problems observed. Some of the issues were caused by the design and some were introduced in the implementation. For instance, the robot could not resume the duty when one sensor was down during execution of the robotics system. This was caused by the off-the-shelf light sensor. The back up light sensor was originally built as an off-the-shelf light sensor. Therefore, it wasn't able to resume functionality after the default sensor was down. After examining the design, we found that it can be solved by fixing a variable declaration. To realize the runtime back up sensor, a static variable is introduced and is updated during the sensor switch. To validate the OCL properties, we use the LCD to display the information that is consistent with the robot moving. During implementation, the display is consistent with the robot movement and we found that all constraints are maintained.

6 Conclusion

This paper presented UML based robotics system design architecture on the three components – static structure, behavior structure, and OCL constraints. The approach to building correct and reliable robotics is validated in a LEGO NXT Mindstorm tool kit on a well developed claw car robot. During the study, we have carefully developed the UML robotics architecture model, design, assembly, and implemented the software code in the Java platform. Afterwards, a system validation was conducted to validate the OCL constraints of the robotics system.

From the study on the LEGO NXT tool kit, we can conclude the following: First, the fundament diagrams of UML model with the OCL constraints are suitable for the design and development of reliable robotics systems. Secondly, the UML based robotics architecture can be used for the general robotics system design. Finally, the LEGO NXT tool kit can be used for the fundamental design and implementation research study.

For the future work, we expect two aspects – one is the real time embedded system specification. The other aspect is developing model checking on the OCL constraints. The UML based robotics system architecture can be used to describe the real time system architecture. OCL is suitable for the specification of timing constraints if the class diagram and behavior diagrams include the timing concerns. Secondly, the constraints can be validated in two other ways – a) model checking and b) assertion implementation.

UML robotics architecture model allows the system analyst and developer to focus on front end conceptual issues before implementation. Using this architecture, we are able to develop reliable robotic controller code that performed all specifications and requirements. It is worth to note that LEGO NXT tool kit is a good set for the low cost robot design. However, the imprecision of the accessories causes some problems during actual implementation.

This study shows that combining the use of UML and OCL, the flexibility of the Lego NXT kit, and the robustness of Java LeJOS, we were able to build a robot that met requirements. Although there were specific hardware parts necessary for these abilities to be realized, the use of UML gave us the ability to use a clear, concise method in the software development process to reach each one of the robot's projected goals.

7 Acknowledgements

We would like to extend our thanks for valuable comments from reviewers.

8 References

- [1] Blaha, M. and Rumbaugh, J., *Object-Oriented Modeling and Design with UML Second Edition*. Pearson Prentice Hall, 2005
- [2] The Association for Computing Machinery, Inc. A list of Implicit Subject Descriptors in ACM CCS (N.D.) Available from: <http://dl.acm.org/lookup/CcsNoun.cfm>
- [3] The Association for Computing Machinery, Inc. Copyright 2011 Retrieved on November 26, 2011 available from: www.acm.org
- [4] Dave Parker, LEGO MINDSTORMS NXT! Gives fun projects and building instructions using LEGO MINDSTORMS NXT robotic kit. Copyright 2007-2011, available from www.nxtprograms.com
- [5] Robotic Equipment Spotlight, January, 2011. Available from: <http://www.robots.com/blog.php?tag=496>
- [6] Guizzo, Erico IEEE Spectrum Inside Technology, World Robot Population Reaches 8.6 Million Wed, April 14, 2010 Available from <http://www.robots.com/blog.php?tag=496>
- [7] A. Brown, Robot Population Expansion, ASME, February 2009, <http://www.asme.org/kb/news---articles/articles/robotics/robot-population-explosion>
- [8] Document Associated With Object Constraint Language, version 2. <http://www.omg.org/spec/OCL/2.0/>. May 2006.
- [9] Jacobson, I., G. Booch and J. Rumbaugh, 1999. *The Unified Software Development Process*, Addison-Wesley.
- [10] Gomma, H., 2000. *Designing Concurrent, Distributed and Real-Time Applications with UML*. Addison-Wesley.
- [11] Maciaszecz, L., 2001. *Requirements analysis and system design*, Addison-Wesley.
- [12] Bagnall, B., 2007. *Maximum Lego NXT: Building Robots with Java Brains*. Variant Press.
- [13] JosWarmer and Anneke Kleppe. *The Object Constraint Language: Getting Your Models Ready for MDA*. The Addison-Wesley object technology series. Addison Wesley, Reading, Mass., 2 edition, 2003.
- [14] JosWarmer and Anneke Kleppe. *The Object Constraint Language : Precise Modeling with UML*. Addison-Wesley, 1998.
- [15] OMG. OCL Version 2.0, 2006. Document id: formal/06-05-01. [cited April 2008]. Available from: URL: <http://www.omg.org>.
- [16] LEGO Team. NXJ API. Available from: <http://lejos.sourceforge.net/nxt/nxj/api/index.html>.

UML Based Design of LEGO Robots

Lorenzo Jones, Janise Fowler, Samuel James, and Yujian Fu

Department of Computer Science, Alabama A&M University, Normal, AL, USA

Abstract—Robots are often used to increase human productivity and decrease human error. It is a key to build a reliable and correct robot that does not contain bugs and errors and demonstrate fault behaviors. In this paper we design and develop an intelligent, multifunctional robot using object oriented software development (OOSD) approach - a UML (Unified Modeling Language) based robotics architecture that uses the UML diagrams and Object Constraint Language (OCL). UML is a standard graphical-based design language that has been widely used in the software-intensive system design. However, there is not enough research work that has been done in the reliability modeling and analysis of robotics. In this paper, the robotics architecture is described by three components - the static structure (class diagram), the dynamic behavior (state machine diagrams) and the property constraints (OCL) on the static and dynamic components. To validate the approach, a robot was built from LEGO Mindstorm NXT tool kit and implemented in the Java platform. The built robot is tested against the OCL properties to validate the required properties. LEGO Mindstorm NXT tool kit is a low cost, multiple platforms, and high integrated setting that mainly used for educational and research robot. Behavior Programming is used in the Java LeJOS platform to sequence the concurrent behaviors in an interleaving way by prioritizing each behavior defined in a class. In conclusion, through our case study robot on the LEGO Mindstorm NXT tool kit, we found that the UML based robotics architecture can be used to successfully design and develop correct and reliable robotics systems.

Keywords: UML model, class diagram, state machine diagram, object oriented design, Java

1. Introduction

In the past several years the interest in robots and autonomous systems has risen. Although robots date back to as early as the 1400's the actual term "robotics" was not used until 1940 by Issac Asimov. While introducing this term Asimov developed the Three Robot Laws: (1) A robot may not injure a human being, or, through inaction, allow a humanity to come to harm (2) A robot may not injure humanity, or, through inaction, allow humanity to come to harm, unless this would violate a higher order law, (3) A robot must protect its own existence as long as such protection does not conflict with the First or Second Law [11].

After decades past, these Three Robot Laws are not enough to build a robotics system. More and more rules are added upon these Three Robot Laws. On the other hand, to maintain and obey these three laws, a lot of challenges of robotic development are issued, more techniques are desired to meet more requirements for a dynamic mobile robot to develop a multifunctional, robust, and reliable robot. The goal of this work is to study a new framework on the UML model for the robotics system to investigate the suitability of using the object oriented software development process (OOSD) in the robot development. Unified Modeling Language (UML) is used on the robotics system design and implementation. UML is a standard graphical based language that has been widely used in the software-intensive design and specification. In this paper, we proposed a robotics system modeling framework based on the UML diagrams by structuralizing the system into three components - static structure (represented by class diagram), dynamic behaviors (represented by state machine diagram) and property description (OCL constraints). State machine diagram is associated with the classes defined in the class diagram or the system, and represents the observable state transitions during event stimuli. OCL constraints are a set of formulae based on the set theory and first order logic and used to ensure the correctness and reliable behavior of the robot.

Autonomous control requires the ability of flexible interaction of system elements autonomously. Therefore, it is very important to handle the dynamics and complexity due to the greater flexibility and autonomy of decision making in the robotic systems. With this study, we show that this process modeling framework for the autonomous control is suitable for the quality assurance of mobile systems.

LEGO Mindstorm NXT tool kit is used to implement the above modeling framework and validate the design requirement specified in OCL. The LEGO Mindstrom NXT robot system is codified in JAVA using Eclipse, which is plugged into the java API's for LEGO Mindstorm NXT package.

The paper is outlined as follows. Section 2 will give a brief discussion of object oriented software development (OOSD) process, a short introduction of UML model and OCL. Section 3 will present the related works of robotics and complex embedded system design using OOSD and UML. Section 4 presents the UML model of the robot and system constraints that the robot is expected to satisfy after an introduction of primary function of the robot. Two case studies on the

LEGO Mindstorm NXT toolkits are presented in Section 5 with the illustration of UML framework application. Section 6 discusses the implementation of the above robotics system in JAVA with Behavior Programming (BP) and some of the limitations and malfunctions of the system. Section 7 concludes the work and discusses the future work.

2. Preliminaries

This section will introduce the preliminary concepts used in this project - Object Oriented Software Development (OOSD), Unified Modeling Language (UML), and Object Constraint Language (OCL).

2.1 Object Oriented Software Development (OOSD)

The object oriented software development (OOSD) process has five major steps which are requirement gathering and analysis, system design, implementation, integration, operations, and maintenance. OOSD views the system as a collection of objects resulting in a more complex system than other models. The functionalities of the system are defined by the interactions and messages between objects.

Unlike other development processes, like the waterfall and spiral processes, OOSD requires the developer to place more thought and emphasis into the analysis and design phase of the model. This project uses OOSD methodology (UML) to model the robot and realize the system in JAVA, an object oriented programming language (OOPL). Some of the key concepts of object oriented development are the usages of classes, polymorphism, and, inheritance, which are all reflected and implemented in JAVA.

During the design phase the developer is able to plan and analyze different design models and decide which models to implement to satisfy the requirements of their system. When the design models are selected the developer must take into account the complexity and functionality of the system. This process is very selective but it is imperative in order to have a correct and reliable robot. For our project we will be using UML to ensure that our robot is both correct and reliable.

2.2 Unified Modeling Language (UML)

UML is a standard graphical-based design language that has been widely used in the software-intensive design [12]. Although there is few work of modeling robotics systems in UML, many works has been done in the modeling and specification of embedded systems using UML. Several key attributes of UML are important for modeling embedded systems. Several key features are included in the UML such as profile and real time for embedded system design. Supporting for OOSD and appealing to the software community is another key feature. Besides, UML support for state-machine semantics which can be used for modeling and analysis. Finally, UML supports object-based structural decomposition and refinement.

To describe and specify the functionality of the robot and the interactions between the object, we use class diagram to model the static structure of the robot, and state machine diagram to represent the dynamic behavior of the system. Class diagrams represent relationships between classes that are represented by boxes with three sections, the top section indicates the name of the class, the middle section lists the attributes of the class, and the third sections of the diagram lists the methods [3]. State machine diagrams describe the interactions and states of different objects within and outside the system, as well as, with each other. States are graphically represented by a rounded rectangle that represents the elapsed time an object is in a state or waiting. There are three sections to the state machine diagram, from top to bottom, the sections are: name of the state, variables, and triggered operations.

2.3 Benefits of Object Oriented Development and Design

Object oriented development is vastly growing. One of the reasons of this growth are the benefits object oriented development provides. One reason why the OOSD approach is becoming popular is reusability. OOSD allows you to reuse objects and place them in other codes. With this usability it also allows the developer to add on and make changes to a particular object without making any changes to the entire system. Besides reusability OOSD has a number of other benefits, these are simplicity, flexibility, extensibility, and maintainability, and cost.

OOSD models real world objects reducing the complexity of the system but also presenting the program structure in a very simple way that is easily to understand. The real world objects are organized into classes of objects and are able to associate with other objects through behaviors. In order to describe these associations and behaviors OOSD uses class diagram which are a part of UML.

Unlike other developments, OOSD gives you a great and wide range of flexibility. By using objects one can build a new behavior from an existing object. This object can be called and/or created at any time within the system. This characteristic is important especially within our robot because we use several different behaviors in order to perform different functionalities of the robot. OOSD also provides extensibility. If requirements, customer needs, or any other issues occur were a feature must be added OOSD gives you that extensibility to easily add the new feature. During our project there was several times that we had to add new features. However, by adding this new object to satisfy the additional feature the current objects were not affected. Another major advantage of OOSD is maintainability. In OOSD maintaining an object can be done both separately and in different locations, and remotely, making maintainability a lot easier.

2.4 Object Constraint Language (OCL)

To complement the UML diagrammatic notation, the Object Constraint Language (OCL) [10], [6] can be used to express constraints and specify the effect of operations in a declarative way. In each predicate of OCL, the logical statements must be satisfied by all valid instances of the system that represented by constraints.

The OCL [10], [1] is a textual, declarative language based on first-order logic and set theory. In addition to expressing constraints on class diagrams, OCL can also be used to specify the effect of the execution of an operation, using pre and post conditions. A pre condition is an OCL statement that has to evaluate to true before the execution of an operation, while a post condition is a statement that has to evaluate to true when the operation terminates.

3. Related Work

Currently there is few work in the UML design of robotics systems. However, recently there is a trend to implement LEGO Mindstrom NXT using JAVA. JAVA allows the developer to write a wide range of applications, as well as, access to different devices like Bluetooth, cameras, mp3 players. API's provide JAVA with its wide range of communication resulting in a simple framework. Many works have been done in the modeling of embedded systems using UML model. UML is very popular because it makes for an easy transition from design to implementation.

There are many works in the embedded system design using UML models. Authors in [8] presented an approach for modeling embedded systems using extended UML as well as generating SystemC code from UML class and object diagrams. Damasevicius and Stuikeys [4] examined system level design processes that are aimed at designing a hardware system by integrating soft IPs at a high level of abstraction. They combine this concept with object-oriented hardware design using UML and meta-programming paradigm for describing generation of domain code.

However, none of the above approaches use OCL with UML diagram to describe the system constraints for the purpose of verifying the functional correctness of the synthesized system. Our work aims at modeling and validating the system properties to ensure the quality of robotics systems. OCL is used in the framework to specify the constraints of the static structure and dynamic behaviors of robotics UML model.

4. UML Based Object Oriented Robotics Framework

UML consists of a combination of data modeling, business modeling, object modeling, and component modeling [7], [5], [2]. UML diagrams can be categorized as three groups - structure diagrams, behavior diagrams, management package. Structure diagrams represent the organization of

the classes in the system in a static view and emphasize relations among classes. Behavior diagrams describe the state changes, communications among objects and instances, which capture the varieties of interaction and instantaneous states in the systems. Management Diagrams include Packages, Components, and Subsystems, focuses on the packaging and setting of diagrams. Behavior diagrams, focuses on the functionality of the system. They describe how the system is going to operate. Structural diagrams are widely used to describe the architecture and organization of a software system. While behavior diagrams are used for the dynamic operations during runtime.

In this paper, we presented a UML based architecture for the robotics system design to ensure the quality of the robot. The architecture is composed of three components (Fig. 1) - static structure (represented by class diagram), dynamic structure (represented by communication diagram or state machine diagram) and system properties (specified by OCL).

The OCL specification can describe all desired constraints on the static structure and dynamic structure. From Fig. 1, we can see that is the OCL properties are not satisfied, we can go back to the model architecture and find out what is the problem. After the properties and constraints are ensured, then the system can be implemented based on the model. The verification of OCL properties can be done by software testing tools or model checking techniques. In the following, we present two case studies using the LEGO NXT toolkit to realize the framework – object detection robot and claw strike robot.

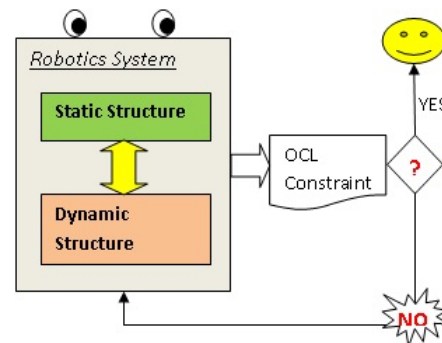


Fig. 1: The UML Framework for Robotics Systems

5. Case Studies – LEGO Robots

There are two case studies shown in this paper using the UML Robotic Framework (1). First, we introduce a on-path object detection robot. The sequence of functions of this robot are:

- 1) Identify the start point and ending point by color.
- 2) Following path by defined color.

- 3) Two ways of starting are: started by the button (in NXT brick), and start by clap (voice driven). The default is clap starting.
- 4) Picking up object on the path.
- 5) Dropping the object in the destination (after the ending point).

The second robot is a claw strike robot. The robot is able to realize following functions:

- 1) Identify the object during moving.
- 2) Stop and back for a short distance if object is found.
- 3) Rotate the motor to throw the strike and hit the object.

In the following, we discuss the application of UML framework in this two LEGO robot design. Three components are presented in the following – class diagram, state machine diagram and OCL constraints.

5.1 Class Diagram

We first present the class diagram of the path following object detection robot, after that, we will show the class diagram of claw strike robot.

On-path Object Detection Robot The class diagram (Fig. 2) for the object detection robot is a graphical description of class relation. It also describes how the light, sound, and ultrasonic sensor will react and communicate with each other and the NXT brick.

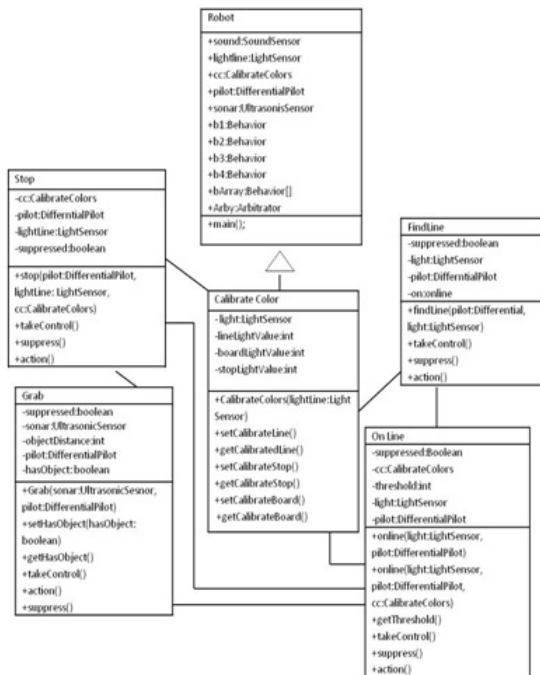


Fig. 2: The class diagram of on-path object detection robot

In the development of the software for the object Detection Robot a total of six classes were used. (Figure 2) Robot is the main class consisting of various objects

of the systems, which are objects of classes Stop, Grab, CalibrateColor, FindLine and OnLine. A crucial behavior array was used to represent a series of actual behaviors of the robot. Four behaviors - findLine, online, Grab, and stop - are defined in a corresponding class. The order that the behaviors appear in the array are decided by the priority which can be assigned as an integer number. This priority is used to control the instance of behaviors to be activated at the runtime. Therefore, each time there is one behavior is taken and executed.

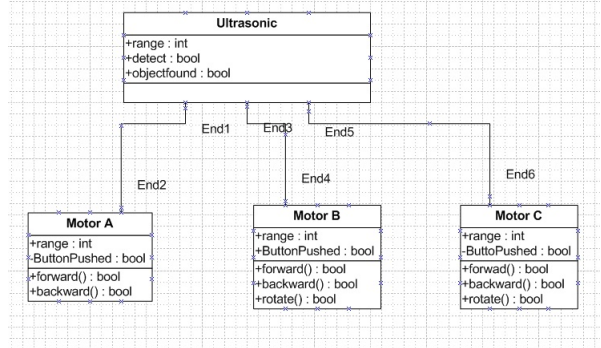


Fig. 3: The class diagram of claw strike robot

In this project, the robot's first behavior is finding line (in class findLine). Class FindLine contains three methods takeControl(), suppress(), and action().the method takeControl() returns a Boolean value to indicate is this behavior should become active. The method action() starts when the behavior becomes active. This method should exist when the action stops. Besides, the action() method also exists promptly is suppress() is invoked. When the action() method exits, we need to make sure to leave the robot in a safe state for the next behavior. The action that it will take will be to look for the path light value. Once the lightvalue for the path is found the motors will rotate and the robot will start on the path/line. Online class makes sure the robot stays on the path with two light values, a light value for the path, and a light value for the board. The action of this class is to set the speed of the motor. The action of the Grab class is to collect and release the object on the obstacle course. While on the path, the robot will use the ultrasonic sensor to detect an object. The motor on the claw will rotate to open then rotate to close the claw in order to grab the object. The Stop class is to stop the rotation of the motors to stop the robot from moving and rotate motor on the claw to release the object. In order for the Stop class to take control the lightvalue must be equal to stopLightvalue. StopLight, boardLight, and lineLight values are collected and calibrated in the CalibrateColor class right before the sound sensor collects input. This class is not in the behavior array but its values are used throughout the other classes in the program.

Claw Strike Robot The claw striker is a robot that detects an object within its perimeters then strikes the object once

detected. The robot not only need to know when the target is in range. Also, it should be able to detect when it is either too close or too far from the target itself. UML the class diagram (3) were initialized to show a visual display of the design for programming the robot. In Fig. 3, a partial design of class diagram is shown to ensure the above behavior. Four classes were implemented the Ultrasonic class, Motor A, Motor B, and Motor C classes respectively. The Ultrasonic class was a public class used to code the Ultrasonic sensor. In this part of the code, the range was an integer data type, two operations detect(), and objectfound() were implemented in this class. Once the object is found, the object needs to be in range; otherwise, the robot will move forward, backward and/or turn around to rotate the motors and move the robot to be in range.

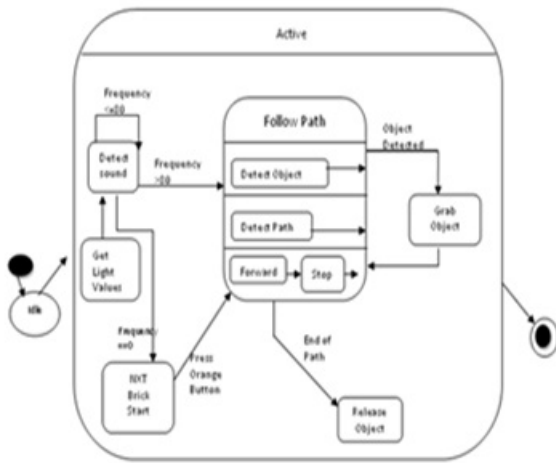


Fig. 4: The state diagram of the on-path object detection

5.2 State Machine Diagram

On-path Object Detection Robot In the on-path object detection robot, three typical states are identified - idle, active, and end states. The robot will be running on the line before reach the object. This period of time, the robot is in the idle state. After detecting the object, the robot will pick up the object and keep following the line before stops. The robot is in the end state when it stops in the specified position. Any actions taken during active state will be represented as the internal states (Figure 4.).

Six states are considered in the active state - GetLight-Values, DetectSounds, FollowPath, GrabObjects, ReleaseObject, and NXTBrickStarts - to illustrate the complexity of the Object Detection Robot. The robot is designed to have two starting ways - sound driven and NXT brick starts. Sound driven is the default one. Before starts, the robot will read the light value of the background (floor), the path value and the end point value into the variables. After reads into those

values, the robot can start to follow the path line based on the values.

The state FollowPath is defined as super state with three concurrent states - detectObject, detectPath, follow, which indicates that this super state needs to take care these three states simultaneously. The robot will enter into GrabObject state if the object with the defined color is identified by ultrasonic sensor. The key is when the robot entering into GrabObject state, the robot needs to return back to the FollowPath state.

Claw Strike Robot The state diagram of claw strike is shown in 5. There are four active states are identified except for the initializing state – moving, detecting, adjusting, and throwing. A synchronization bar is used to indicate that the robot will detect during moving status. The condition of *object found* must be validated before the robot throw the strike to assure that the ultrasonic sensor has the data return positively. Otherwise, the robot will keep moving and detecting.

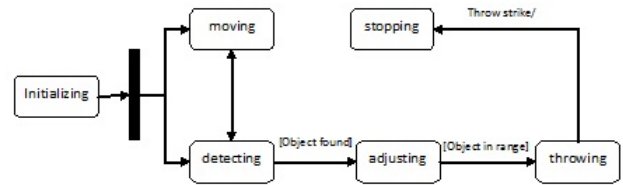


Fig. 5: The state diagram of the claw strike robot

5.3 System Constraints

To ensure the quality of the object detection LEGO robot, we use OCL to define system constraints. The system is expected to hold all the constraints as well as maintain the functionalities.

On-path Object Detection Robot A system invariant can be expressed as when an object is detected on the path during robot is moving, the robot must pick up the object (Formula i)). Another invariant is once an object is picked up, the robot must take the object to the destination, which is indicated by the end line of yellow color (Formula ii)).

```
Context:
Grab
Inv: If Grab.Grab() then
setHasObject() hasObject i)

Context:
Grab
Inv: If Grab.hasObject then getCalibrateLine()=yellow
takeControl()=clawopen ii)
```

In addition, each class has a set of associated OCL constraints that specifies the requirements of the class. the requirements are specified by pre-condition, post-condition and invariant of the class. Each state machine diagram has a

set of associated OCL constraints with pre-condition, post-condition and invariant specified. Due to the space limitation, we only show some of the OCL constraints in the Formulae iii) to Formula viii).

```
Context:
FindLine
Pre: if FindLine.findLine=true then takeControl()    iii)
Post: suppressed=true  action() = following line    iv)
```

```
Context:
Stop
Pre: CalibrateColor.stopLightValue = yellow        v)
Post: Stop.takeControl()  Stop.action()            vi)
```

```
Context:
OnLine
Pre: on = online  stopLightValue = 0                vii)
Post: on = online  OnLine.action()                  viii)
```

Claw Strike Robot The key class in the claw strike robot is Ultrasonic class. There are two conditions need to be met to ensure the correct behavior: a) the object is found, and b) the object is within the range. This can be specified in the following

```
Context:
Ultrasonic
Pre: if objectfound=true then checkrange            ix)
     if objectfound=true and range< then throwStrike x)
Post: motorc.rotate    xi)
```

6. Java Implementation of UML Model in LEGO Toolkit

In this section, we present how to implement LEGO robots based on the UML model shown in above and discuss the issues that were found during experimental.

The **on-path object detection robot** is a multifunctional and reliable object detection robot. LEGO NXT Mindstorm Tool Kit sensors are used to implement the robots functionality and make the robot more complex and intelligent. LEGO Mindstorm NXT tool kit has been widely used in the classroom for educational and research purposes. The robot will have several main functions: staying on the predefined path with the light sensor, being able to detect objects with the ultra sonic sensor, determine the color of the object with the light sensor, and collect input from user with the sound sensor. The final build of the robot is shown in Fig. 6.

The important feature we want to identify is the fault-tolerance using redundancy. Considering if the sound sensor is broken due to some unknown reason, the system is not able to starts normally. To make sure the system can start with the defined the functionalities, the NXT starts is designed and the brick will be automatically resumed if sound sensor stops working. This feature is reflected in the UML diagrams (Fig. 4).



Fig. 6: The on-path object detection robot

An implementation feature of this LEGO robot is using Behavior Programming (BP) [9]. Behavior programming (BP) is imperative in the use of LEGO NXT API's and introduces the theory of priority. The key concept of BP is serializing all behavior by assigned priority: only one behavior can be active and in control of the robot at any time, each behavior has a given value named "priority", the controller can determine if a behavior should take control based on the behavior queue by priority, the active behavior has higher priority than any other behavior that should take control [9]. Three methods – takeControl(), void action(), and void suppress() – need to be overridden in the BP. The method take control() is used to indicate if a particular behavior should be active. For our project it would indicate that that the robot will perform an action when an object is detected. The method action() performs the actions when takeControl() is true. This action might be to move the motor forward or backwards. The method suppress() is used to terminate the implementation of the action code. Finally, an Arbitrator is an array that regulates when each behavior should become active. The arbitrator regulates the behaviors by using priority with the zero element of the array representing the lowest priority. The Robot class also has the test for the sound sensor by using if statements. If the sound sensor receives a frequency reading frequency>80 then the robot is able to proceed to findLine. If the robot receives a frequency ≤ 80 the sound must loop through again until sound frequency is greater in 80. If this statement is true, the sound sensor is down, user must press orange button on NXT Brick.

In the **claw strike robot**, the robot not only need to know when the target is in range. Also, it should be able to detect when it is either too close or too far from the target itself. As shown in the class diagram (Fig. 3), The Ultrasonic class was is associated with three motor classes. In this part of the code, the range was an integer data type, two functions detect(), and objectfound() were initialized in this class. Once the button was pushed on the brick the ultrasonic sensor will use the detect() to start looking for any particular object within a range of 50cm. If the range was below 50 the classes of Motor A,B ,and C will be initialized. Motor A controls the striker which will only strike if the target is at

a range of equal to 40 cm. If the target is less than or equal to 39cm, motors B, and C move backward. if the range is between 49 to 41 cm the Motors B and C will move forward. If the ultrasonic sensor detects an object close to it but, not in front of it, motor B, and C will rotate. The problem that occurred was Motor A went forward but, did not proceed backwards once the target was struck by the claw. Also, the claw did not react quicker in striking the target as expected. The solution to these problems was adding a time delay. The time delay was set at 50ms meaning once the robot detects the robot and strikes the claw will autonomously return to its previous position. The final build of the claw strike robot is shown in Fig. 7.



Fig. 7: The claw strike robot

6.1 System Malfunction and Limitations

While we admit that LEGO NXT Mindstorm tool kit is a highly integrated lost cost robot set, some sensors do not have enough precision to maintain some required performance. This caused a lot of time of code revision and validation. If the robot is unable to receive a frequency input from the sound sensor the on-path object detection robot will still be able to operate. Instead the robot will be able to communicate with the user through the LCD screen on the NXT Brick. Although the on-path object detection robot consists of many functions it does have some system limitations. The robot cannot distinguish between different objects it can only determine the color of the objects. Because we are using a light sensor and not a color sensor, colors that have close together in the spectrum light white and yellow are hard for the light the sensor to distinguish between. The sound sensor can only recognize frequencies greater than 80. The robot cannot detect objects less than 3cm away. This can be improved in the version 2 of LEGO NXT Mindstorm tool kit.

7. Conclusions and Future Works

This paper presented a UML based robotics system modeling framework that is composed of three components - static structure, static structure (represented by class diagram), dynamic behaviors (represented by state machine diagram) and property description (OCL constraints). State

machine diagram is associated with the classes defined in the class diagram or the system, and represents the observable state transitions during event stimuli. OCL constraints are a set of formulae based on the set theory and first order logic and used to ensure the correctness and reliable behavior of the robot. The purpose of this work is to develop a reliable and correct, intelligent, multifunctional robot using OOSD approach. After the testing and validation of the robot, we can confidently ensure that the robot maintains the required function with the correct behaviors.

Future work for this project falls in two aspects. First, the current framework will be extended with verification component that include model checking technique. In that case, the static structure (class diagram) and dynamic behavior (state machine diagram) will be converted to the programming language that can be read by model checker. We will run this model against the OCL constraints to check if the model meets the constraints specified in OCL. The model checking technique is able to detect the problems in the model with meaningful feedback. Secondly, as a typical real time embedded system, timing concern is a key for the robotics systems. We will extend the current model to describe the timing issue of the robot.

Acknowledgment

The authors would like to thank all reviewers for the kindly comments and suggestions on this work.

References

- [1] Uml 2.0 ocl specification. Available from <http://www.omg.org/docs/ptc/03-10-14.pdf>.
- [2] Unified modeling language (uml), version 2.0. Available from <http://www.omg.org/technology/documents/formal/uml.htm>.
- [3] O. Andriyevska, N. Dragan, B. Simoes, and J. I. Maletic. Evaluating uml class diagram layout based on architectural importance. In *Proceedings of the 3rd IEEE International Workshop on Visualizing Software for Understanding and Analysis, VISSOFT '05*, pages 9–, Washington, DC, USA, 2005. IEEE Computer Society.
- [4] R. Damasevicius and V. Stuikeys. Application of uml for hardware design based on design process model. In *Proceedings of the 2004 Asia and South Pacific Design Automation Conference, ASP-DAC '04*, pages 244–249, Piscataway, NJ, USA, 2004. IEEE Press.
- [5] H. Gomma. *Designing Concurrent, Distributed, and Real-Time Applications with UML*. Addison-Wesley Professional, 2000.
- [6] M. Richters. *A Precise Approach to Validating UML Models and OCL Constraints*. PhD thesis, University at Bremen, Bremen, Germany, 2001.
- [7] J. Rumbaugh, I. Jacobson, and G. Booch. *The Unified Modeling Language Reference Manual*. Addison-Wesley Professional, July 2004.
- [8] V. Sinha, F. Doucet, C. Siska, R. Gupta, S. Liao, and A. Ghosh. Yaml: A tool for hardware design visualization and capture. In *Proc. Int. Symp. on System Synthesis*, pages 9–16, 2000.
- [9] L. Team. Nxj technology. <http://lejos.sourceforge.net/nxj.php>.
- [10] J. Warmer and A. Kleppe. *The Object Constraint Language: Precise Modeling with UML*. Addison-Wesley, 1998.
- [11] Wikipedia. Three Laws of Robotics. http://en.wikipedia.org/wiki/Three_Laws_of_Robotics.
- [12] O. Wongwirat, T. Hanidhikul, and N. Vuthikulvanich. A formal approach in robot development process using a uml model. In *Control, Automation, Robotics and Vision, 2008. ICARCV 2008. 10th International Conference on*, pages 1888 –1893, dec. 2008.

Model-based Generation of Workunits, Computation Sequences, Series and Service Interfaces for BOINC based Projects

Christian Benjamin Ries
Computational Materials Science
and Engineering (CMSE)
University of Applied Sciences
Bielefeld, Germany
www.visualgrid.org

Christian Schröder
Computational Materials Science
and Engineering (CMSE)
University of Applied Sciences
Bielefeld, Germany
Christian.Schroeder@fh-bielefeld.de

Vic Grout
Creative and Applied Research for
the Digital Society (CARDS)
Glyndŵr University, United Kingdom
v.grout@glyndwr.ac.uk

Abstract—Berkeley Open Infrastructure for Network Computing (BOINC) is a popular Grid Computing (GC) framework which allows the creation of high performance computing installations by means of Public Resource Computing (PRC). With BOINC's help one can solve large scale and complex computational problems. A fundamental element of BOINC is its so-called workunits (WUs), each computer works on its own WUs independently from each other and sends back its result to BOINC's project server. Handling of WUs is a challenging process: (1) the order of used input files is important, (2) even more contributory components has to know how these input files are structured and on which data format are they based for an accurate WU processing. Small modifications can have a high impact to a BOINC project. Indeed scientific applications, BOINC's components, and third-party applications all have to be adjusted to have a correctly running project with desired the functionality. This can be a highly error-prone and time-consuming task. In this paper we present a Unified Modeling Language (UML) model to give a high abstraction for BOINC's WU handling. Only a model description and a corresponding code-generator are necessary to construct a WU handling infrastructure with less development and implementation effort: (a) one model to fit most WU cases and (b) essential interfaces for WU access.

Keywords—BOINC, Code Generation, Modelling, UML, Work

I. INTRODUCTION

Set-up of a Berkeley Open Infrastructure for Network Computing (BOINC) project can be a challenging and sophisticated task. Despite the fact that it is necessary to implement a scientific application (SAPP) [10] and to establish a fully operable server infrastructure [7], moreover it is necessary to describe how SAPP and all BOINC components handle computational jobs. Here, participating clients retrieve a project specific SAPP from a BOINC project (BP) server along with so-called workunits (WUs), i.e. a number of parameter usually provided in data files of ASCII or binary format that are optionally needed by the application to perform specific tasks. The idea in this paper is to have a Unified Modeling

Language (UML) model and code-generation (CG) facilities, which have to support developers with the ability to generate all required WU configurations, interfaces for opening and accessing WUs, and creating one or more computational series and sequences, i.e. different computational jobs with varied runtime configurations.

A. Unified Modeling Language & Object Constraint Language

One of the primary goals of UML is to advance the state of the industry by enabling object visual modeling tool interoperability [15]. Since version 2.2, UML has 14 different diagram types subdivided in three categories: (1) structure diagrams, (2) behavior diagrams, and (3) interaction diagrams. In this paper we use the *Class* and *State Machine* diagrams. Class diagrams are used to specify system related elements, e.g. a class can describe a SAPP. An instantiated class element is seen as an object and mostly it is an executable instance. UML state machines help to model discrete behavior through finite state-transitions systems. It can be used to visualize the current state of one system, and orthogonal regions allow to model client-server state-machines where each side is working independently. The Object Constraint Language (OCL) is used to express constraints and properties of UML model elements [16].

B. BOINC's Workunit System

BOINC uses a fine-grained file based system to set-up WUs for a BOINC project (BP). WUs are packages with descriptions of input and output data needed by the SAPP to perform specific tasks [1]. Before a WU can be added to a BP, it is necessary to create several input files with planned to use datasets for one computation. Two additional template files are required: (1) an input template to describe which files are used as input, how they are ordered and which flags for them are set, and (2) a result template to describe how output files must be named by the SAPP, or how big in bytes they can be [3].

This project is funded by the German Federal Ministry of Education and Research.

C. Research Topics

To make the handling of WUs easier some questions arise and we will work on them within this paper.

- Which UML elements are necessary to create a model for WU creation?
- How can we model a sequential queue for WU progressing? The answer to this question should make it possible to have WUs with the need of pre-processed results by one or more other WUs.
- How can BOINC's validator and assimilator access result's data on a higher abstraction level? In addition, is it possible to have only one interface or description which makes it possible to allow access by all BOINC components? Here, BOINC's validator is responsible for validating a WU and developers of a BP can implement their own validator routines. The default behavior of BOINC's assimilator is storing of results within a file system.
- How can we track the lifetime of WUs when they are used in different scenarios, e.g. one WU is used within a sequential performed queue?

This paper can be seen as the conjunction of previous work [6], in which BOINC's services are described with UML to be deployable on server farms. This is why «Application» is added in Fig. 1 where previous work is followed in this paper.

The remainder of this paper is organised as follows. Section II describes the problematic of BOINC's architecture to handle different defined WUs for varied kinds of computations. Next, Section III proposes our idea as to how we can fill the gap of BOINC's problematic to utilize it with an easier and less vulnerable interface. In Section IV we use our UML model and apply it to a small case-study. Finally, Section V concludes this paper and Section VI suggests future work.

II. PROBLEMATIC OF BOINC'S ARCHITECTURE

WUs are packages with descriptions of input and output data [1]. These WUs are fundamental pieces for BOINC and contain information on how these data are defined and formatted, i.e. binary data or plain text and functionalities to describe how several data items can be used. The flexibility to define arbitrary structured WUs and input files can be a complex issue. It has been shown, that WUs within a BP are crucial elements and are essential for the BP success [9], [11]. At the time a BP is being established it must be defined how all BOINC components have to handle WUs, otherwise WUs will stop immediately wrongly configured and, as a result, without proper working components. A BOINC administrator needs answers to several questions before a BP can be set-up as a fully operable system. Certainly we think about our computational concern and how we can solve this problem firstly. In this paper we will not discuss this difficulty, previous work has focused on this field of activity [5], [10]. In this paper we will discuss a solution for the following questions:

- Is all information about WU's structure available at the beginning of it's use or are they gathered continuously

during BOINC's runtime? Here, it is also important to define how continuously created WUs differ from each other. It is necessary to know if their content differs and if they must be restructured or not, e.g. if different sigma values have to be set for statistical computations.

- How should WUs be opened and how should all potentially contained sub-elements be handled by a SAPP? Are the nested data defined as plain-text, or as encrypted text, or maybe a binary format?
- It is not only the WU input files that are important. The result files are also essential for the success of a computation. In the later BOINC process they must be validated and subsequently stored by an assimilator to make results usable for particular later cases.
- The assimilation process is used to store results, but what if one WU does not have enough results? E.g. one WU is distributed to three hosts, a minimum of two results must be returned but in one scenario two hosts are too late — deadline is reached — and only one result is available. In this case, BOINC's transitioner will flag the missing results as *overdue*, then directly flagged as *ready for assimilation* by BOINC's validator [2] and after this the assimilation process could create a duplicated WU for a retry. This can be done periodically until the WU is completely returned and successfully validated, or after some failed tries the available results can be stored in a database or on the file system which can be defined for failed results.
- Under some circumstances a computation relies on different sets of runtime parameters or they must adhere to a sequence of different runs, i.e. a result of a WU must be used as input for another WUs. In this case, the results must be converted to the right format of a new WU and it can be necessary to modify mentioned attributes for the different purposes of a WU, e.g. an unit conversion can be required before a WU result is usable for subsequent computations.

BOINC's architecture relies heavily on a fragile methodology; if one or more software components are misconfigured or disabled the WU handling chain will be stopped on the failed element, i.e. if the validator is not working properly no validation of returned WU results is executed and as a consequence the WU will never complete.

BOINC's WU consists of two template files, additional input files and, during computation, created output files. Template files are based on an XML [12] format and therefore they are not really human readable and XML-tags can be misspelled very easily. More important is the fact that all input files must be described within this template file and must have a specific order. In the header of the input template the numbering of input files is defined. After this part each file has optional attributes, e.g. a file is sticky and will not be deleted after one computation on one host. A similar approach is used for the description of result files. These files and the part of BOINC's framework for WU creation are elementary and every BOINC

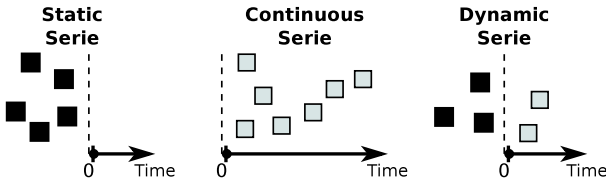


Fig. 2. Our UML model allows us to define «Series» in three different ways: (1) all WUs must be available before a BP is started, (2) during runtime WUs are created and added continuously, and (3) a mix of the first and second; some WUs are available at the beginning and during runtime additional WUs are added to one BP.

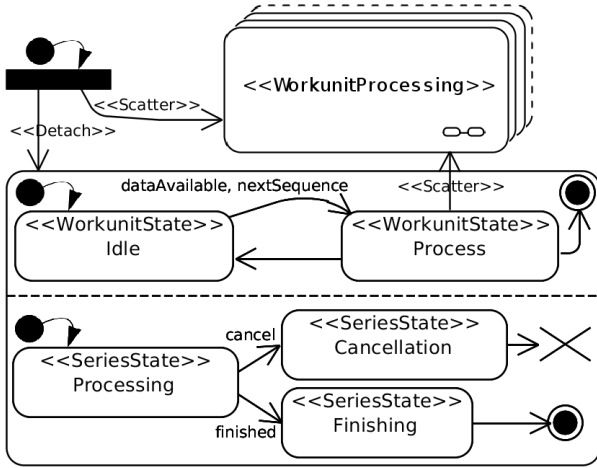


Fig. 3. First part of our UML statechart diagram to monitor instances of «Workunit» and «Series». State's top region is responsible for WU monitoring and creates new WUs when data is available or next WU in a sequence has to be performed. The bottom region monitors a «Series» and handles canceling events for a «Series» instance or if its finished and results can be merged.

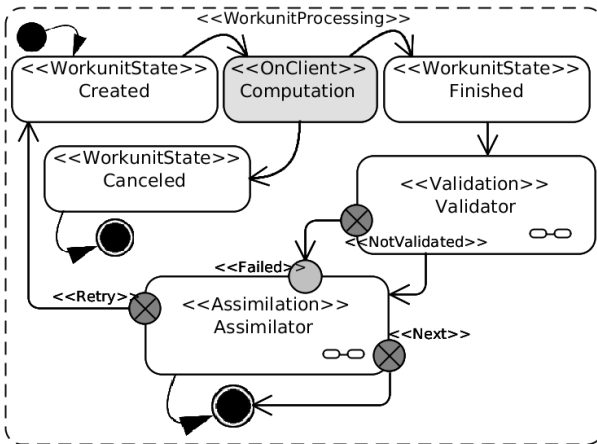


Fig. 4. Second part of our UML statechart diagram for WUs. During computation of one WU, clients can decide to cancel current WU, and therefore it has a changed WU state. When it is finished it will be validated, if this validation failed the exit pseudostate is used. The followed assimilation state can decide to retry this WU and a new WU is created with same «Input» values. If this WU is in a sequence, «Next» is used otherwise the statechart is finished.

type "C.B.Ries" and "Research, Sport" are of the enumeration type *FileType::String*, and
data mentioned string values are the real embedded infor-

mation.

```
<person name="C.B. Ries ">
  <interests topics="Research , Sport"/>
</person>
```

Listing 1. Example of «Datafield» usage to define a XML structure.

B. Rule-based Creation of Datafield Values

During WU creation data fields of input files or the input files themselves can be specified by «Range». Therefore values can be generated by «Range» specializations: (a) «RangeRule» and (b) «RangeSimple». With «RangeRule» a rule-set for value creation can be defined. For this purpose the tag-value rule can be filled with a user-defined rule, e.g. each WU within a specific «Series» can have a corresponding mode for algorithms. «Range» defines an operation *getValue()* which is used to query the related «Datafield» value. As shown in Fig. 1 each rule can only modify one «Datafield». «RangeSimple» is used to have a range-loop for one specific «Datafield», for this reason three tag-values are defined: (1) *start*, (2) *stop*, and (3) *step*. In combination with different additional rules each call of *getValue()* can increment the lower-bound value *start* by *step* to the upper-bound *stop*. One «Range» can be owned by several «Series», as a consequence it must be possible to retrieve which «Series» is calling *getValue()*. For this reason tag-value *activated* is specified. When this tag-value is valued by *true*, the association between «Range» and «Series» can be used to query the currently used «Series». This allows «Range» to access all information of a «Series» with associated «Workunits».

C. Continuous Creation of Workunits for Series

As seen in Fig. 2 a BP can have different scenarios for WU creation:

- **Static** All WUs are created before a «Series» will be created. Only these known WUs are handled by a BP.
- **Continuous** This configuration has no WUs at the beginning of a «Series». WUs are created on demand, e.g. when new data packages are available or when a time slot is reached.
- **Dynamic** In this configuration the previous two possibilities are merged.

These three approaches are supported by our model. «InterfaceDataSource» (IDS) is an interface which is implemented by a «Service» component for WU creation [6]. This component could have several connections to data sources. When these data sources signals new available data packages, IDS provides with the help of *getPath()* a file path which is usable for «Input» and a corresponding «Datafield» has *#File* as *type*.

Fig. 3 shows the first statechart diagram for our modelling approach. Depending on your BOINC scenario you can define how WUs are created. For all mentioned scenarios the statechart will always start at the initial point in the top-left corner. Immediately the process is subdivided into two parts with two transitions stereotyped by «Scatter» and «Detach». The BOINC's WUs are independently processed on

the client side from other processes. In addition to this WUs are structure elements and that's why they are not conceived to have a behavior. Other components have to deal with them and as a consequence these components can have behaviour definitions. While all WUs are public within BOINC's domain any component has access and can modify them.

«Detach» creates two orthogonal regions for the lifetime monitoring of one «Series» and all associated WUs. The top region is responsible for WU monitoring and the bottom region monitors the current «Series». The transition between "Idle" and "Process" is triggered by *dataAvailable* and *nextSequence*. In this transition *dataAvailable* is called by the IDS, and thereupon the file path is used to define a new WU. Fig. 4 shows the statechart of a single WU. In that statechart "Assimilation" has a "Next" named exit pseudostate and *nextSequence* is triggered when this exit is entered. As a result a new WU is created. It is defined that this exit pseudostate can only be used when a WU is part of a sequence as described in the next section. As at the initial point of this statechart, all available and new WUs are scattered and within this statechart are monitored. The top region is left when no more WUs are in process. The bottom region monitors the lifetime of a «Series» and "Processing" is only left under two circumstances: (1) the «Series» has to be canceled and (2) processing is complete and all results can be merged, which can be done in "Finishing", e.g. an average over all results of a monte-carlo simulation can be calculated.

D. Sequences of Workunits

In [4] a system for remote creation of chained WUs is shown, where one result can be used as input for other WUs. In our model we can handle a similar task. «WorkunitAssociation» enables one to define a «Series» with sequential computations. «InterfaceAssimilate» delegates these computations and can have an association to «WorkunitAssociation». As mentioned in the previous section the "Next" exit pseudostate in Fig. 4 is used when one WU is assimilated and has additional WUs to be performed. The following pseudocode demonstrates how the assimilation process can decide if one WU is in a sequence and if a WU follows:

```

Let ws As workunitAssociation . workunit . workunitState
If ws.seqid < ws.maximum_sequence Then
  For ro In Output
    Set workunitAssociation . input = ro
    Where
      workunitAssociation . input . name = ro . useAs . name
    EndFor
  EndIf
ws.seqid = ws.seqid + 1

```

A new WU is filled with «Datafield» values of one «Output» when they are associated by *useAs*. As a consequence *useAs* must only be set when «Output» is used for one «Input» configuration. The fact is, when no *useAs* is available then it makes no sense to check for a sequence.

In the case when all results of a WU are required, BOINC's assimilator can create additional WUs with a duplicated configuration, e.g. a WU is missed to complete a sequence and is

too often canceled by BOINC clients or WU's *delay_bound*¹ is reached. For this occurrence the WU can be copied and added to a «Series».

When a WU is part of a sequence, the WU's name has a special format to distinguish WUs. A similar approach for rBOINC is used [4]. rBOINC defines a specialized WU name and we modify this format to "NNN-SEQ-XX-YY":

- NNN is the name of the WU, and
- SEQ is a start pattern for a sequence description.

Here the embedded string "-XX-YY-" is defined as follows: XX is the current sequence id and YY is used for the maximum number of sequences. This WU name format is used to select sequenced WUs in section III-F.

E. State of Workunit Computation

«WorkunitState» is associated by «Workunit» (WU) and from the beginning of its existence the state of a WU can be queried at any time. The tag-value *state* holds the current state and can be valued with the following variables:

- **CREATED** WU is created.
- **FAILED** WU has failed and can not be finished.
- **COMPUTATION** WU is in progress and one or more clients work on it.
- **DONE** Enough clients have worked on one WU and it can be moved to the validation and assimilation process.
- **VALIDATION** WU has to be validated.
- **ASSIMILATION** WU has to be assimilated.
- **CANCELED** WU is canceled by an administrator or by other processes, e.g. when sequenced WUs have failed or are canceled.
- **FINISHED** WU is finished and ready for later use, e.g. to create a new «Series» or to use their computational results.

For the UML model it is important what the state of a WU is, as a matter of fact the state value is responsible for deciding which actions are performed during the WU processing, i.e. when a WU fails the assimilation process has to decide if it should be performed again. The accessory method *check()* is used to query the current state of a WU and returns a descriptive text value, i.e. the string contains the current state with additional more precise information such as the *timestamp* of the last check or how long a WU is currently processing. The other two tag-values *maximum_sequence* and *seqid* are used for the «WorkunitAssociation» in the next section.

F. Cancellation of Series and Workunits

A WU can be canceled at any time. This is done by *InterfaceWorkunit::cancel()* and it is necessary to cancel WUs which are related to a cancelled WU, i.e. when the current WU is cancelled and has associated WUs, these must be also cancelled because they can never be processed with missing «Input» values. «WorkunitAssociation» has an additional OCL operation to query which WUs are in the current sequence:

¹Deadline of one workunit.

```

WorkunitAssociation :: querySequencedWorkunits (
  seqid : Integer, name : String) : Set(Input);
querySequencedWorkunits =
  self.series ->select( s | s.workunit ->select(
    w | w.workunitState.seqid > seqid
    AND
    — NNN-SEQ-XX-YY
    w.name.substring(1,
      w.name.strpos("-SEQ")) = name
  )
)

```

With this OCL statement, WUs can be selected which are later defined within a sequence and as a consequence they can be cancelled. Following cancelled WUs can have other associated WUs and they must be cancelled. The call of *Workunit::cancel()* is used to cancel one WU. «Series» can be cancelled with this concept, it is enough to call *cancel()* during the iteration of all associated WUs.

G. Service Interfaces

BOINC has several components which need to access «Input», «Output» and their embedded «Datafield» fields. Fig. 1 shows three interfaces to access them: (1) «InterfaceDataset», (2) «InterfaceValidate» and (3) «InterfaceAssimilate». With the help of this structure all functionalities can be generated and this makes the access more comfortable. Changes in the model are automatically resolved and interfaces are always valid for use.

IV. CASE-STUDY

Our case-study modifies a movie, i.e. a movie is fragmented in single image sequences and basic image processing algorithm are applied to these sequences, some results could be seen on the project website [8]. Fig. 5 shows our use-case where one video is added by *C.B.Ries*, with the help of inotify [13] one BP is triggered by *dataAvailable* and WUs are created on demand. During WU adding it has to be clear which kind of data format is used, i.e. in our use-case we add a complete movie and subsequent implementations has to prepare this movie for WU creation. In this scenario we create ZIP-archives automatically and fill them with a number of image sequences.

Five «Series» are defined, in this case only the first fourth can be processed immediately. As shown in Equation 1 the fifth «Series» needs the result of the first four «Series».

$$\left[\begin{array}{l} \text{Series 1 (normalize)} \\ \text{Series 2 (painting)} \\ \text{Series 3 (negate)} \\ \text{Series 4 (edge)} \end{array} \right] \Rightarrow \text{Series 5 (merge)} \quad (1)$$

All «Series» instances have a different runtime configuration, e.g. in «Series» number four the image is manipulated by an edge algorithm. The fifth «Series» merges all previous results where for each image sequence they are added to a 2×2 raster image. On the right-hand side of Fig. 5 these different configurations are shown where the bottom configuration shows the mode “edge” for image manipulation and in the top configuration “merge” is assigned.

The fifth «Series» is not started until the other «Series» has finished. It is important to notice that the SAPP is always the same, only the input files are changed. In the first four computations only two files are necessary: (1) configuration for the mode of configuration and (2) the mentioned ZIP-archive with movie sequences. The last computation is altered and needs five files: (1) configuration as before with different values and (2) all four input files which are created by the other four computations.

This use-case describes a *dynamic series* where some WUs available on BP’s start and additional WUs created during runtime. In the case that one of the first four series is canceled, the fifth series can never be started because of missing input data. This is solved by our statechart construction in Fig. 3 where the “Cancellation” state is responsible for cancelling all related WUs and «Series» instances.

V. CONCLUSION

In this paper we describe a UML model for WU creation and how the lifetime of WU series and individual WUs can be monitored. We have shown that only one model description is necessary to allow BOINC related components to access WU’s input and output files, i.e. BOINC’s validator and assimilator must not be changed to access the WU, all necessary code elements can be generated with one model description. With our model it is possible to set-up different computational scenarios where WUs are generated statically, continuously or mixed by these two approaches. WUs can be added to a process before a BP is started or they can be added on-demand during the runtime of a BP. With the help of UML statechart diagrams we can set-up a BP configuration where we can define how *overdue* or *absent* WUs are handled. It is possible to recreate them or if it is wished, the related computational series and related WUs are aborted.

The proposed UML modeling approach can help to reduce errors during administration of BPs. As a matter of fact, in traditional BPs it is necessary to reconfigure and reimplement several parts when only one configuration is changed, i.e. if the format of computational results is changed then all related components such as BOINC’s validator and assimilator have to be similarly changed. Furthermore, adding or removing input files for one computation has an impact on several BOINC parts: (1) non well readable XML input files must be changed, (2) the call of BOINC’s WU creation tools has to be altered, (3) altered files has to be copied to BOINC’s download hierarchy, and (4) (maybe) input files have to be generated or prepared. Our modeling approach solves all these problems with the help of UML and OCL.

VI. FUTURE WORK

WU’s performance can have restrictions, e.g. the use of floating-point operations or allocation of hard disk space can be limited. Currently it is not clear if UML can help to detect perfectly fitted values for this purpose. During our use-case tests we noted a large number of failed WUs because of

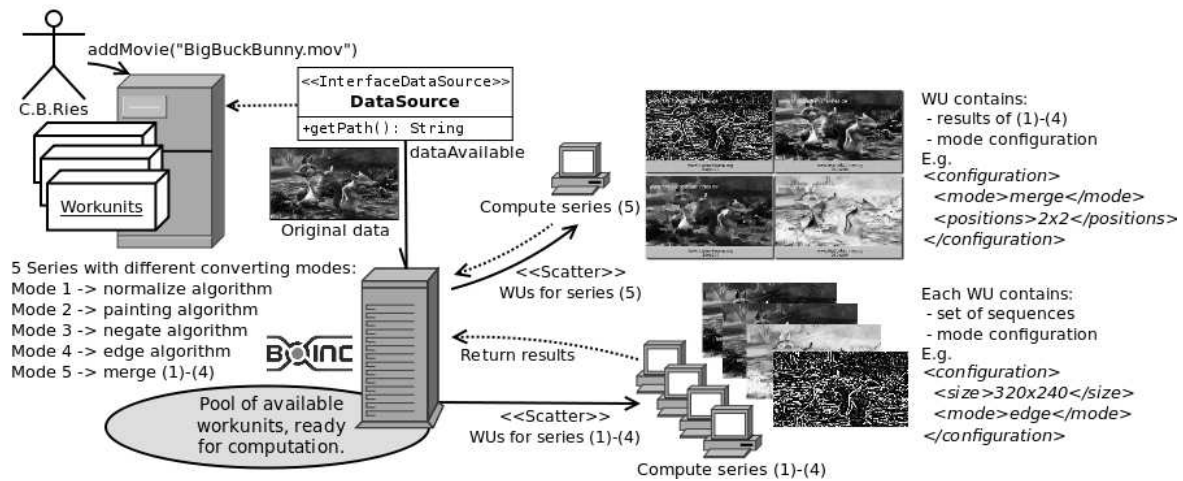


Fig. 5. Use-case to modify a movie: *C.B. Ries* adds a new movie to one server, running applications prepare this movie and fragment it into image sequences, thereupon these images are zipped into ZIP-archives. *DataSource* notifies a BP which automatically adds all announced WUs to this BP. Adding of one ZIP-archive implies four added WUs to this BP but with different configurations, each «Series» instance has a different mode for computation: *normalize*, *painting*, *negate*, and *edge*. These WUs are processed and the computational results are used as input for a fifth «Series», again with a changed mode for computation: *merge*.

wrongly adjusted boundary values for the restrictions mentioned. The set-up of this values has to be more precise and at best automatic. In future work we will work on this objective.

During the writing of this paper implementation with the support of this model are hard-coded and can not be changed — they are not flexible during runtime. One idea is to add a Domain-specific language (DSL) to describe WUs, series and sequences of computation. It could be possible to interpret this DSL during the runtime of a BP, to change the behavior of this BP and to generate code for all necessary components for WU handling on-demand.

Additional thought should also be spent on how our presented model can be used for conventional supercomputers, where other technologies like Message Parsing Interface (MPI) [14] or OpenMP [17] are used. It should be clear that MPI has to process workunits like BOINC, although admittedly with less input files; instead it uses more numerical values which are communicated between all involved computation nodes. The fact is that BOINC can be perfectly used to solve embarrassing parallel computational problems with less communication between all involved nodes. MPI enables one to use a distributed computing environment with several autonomous interacting nodes to achieve a common goal.

REFERENCES

- [1] D. P. Anderson, C. Christensen, and B. Allen. "Designing a Runtime System for Volunteer Computing." in *Proc. ACM/IEEE SC*, 2006, Article No. 126
- [2] BOINC. "Backend program logic," Internet: <http://boinc.berkeley.edu/trac/wiki/BackendLogic> [Version 2]
- [3] BOINC. "Submitting jobs," Internet: <http://boinc.berkeley.edu/trac/wiki/JobSubmission> [Version 19]
- [4] T. Giorgino, M. J. Harvey and G. De Fabritiis. "Distributed computing as a virtual supercomputer: Tools to run and manage large-scale BOINC simulations". *Computer Physics Communications*, vol. 181, February, 2010
- [5] C. B. Ries. "BOINC - Hochleistungsrechnen mit Berkeley Open Infrastructure for Network Computing." Berlin Heidelberg: Springer-Verlag, 2012
- [6] C. B. Ries, C. Schröder, and V. Grout. "Approach of a UML Profile for Berkeley Open Infrastructure for Network Computing (BOINC)," in *Proc. ICCAIE*, 2011, pp. 483-488
- [7] C. B. Ries, C. Schröder, and V. Grout. "Generation of an Integrated Development Environment (IDE) for Berkeley Open Infrastructure for Network Computing (BOINC)," in *Proc. SEIN*, 2011, pp. 67-76
- [8] C. B. Ries and C. Schröder. "Public Resource Computing mit Boinc." *Linux-Magazin*, vol. 3, pp. 106-110, March 2011. Internet: boinc.sourceforge.net
- [9] C. B. Ries and C. Schröder. "ComsolGrid - A Framework For Performing Large-Scale Parameter Studies Using Comsol Multiphysics and Berkeley Open Infrastructure for Network Computing (BOINC)," in *Proc. COMSOL Conf.*, Paris, 2010
- [10] C. B. Ries, T. Hilbig, and C. Schröder. "A Modeling Language Approach for the Abstraction of the Berkeley Open Infrastructure for Network Computing (BOINC) Framework," in *Proc. IEEE-IMCSIT*, 2010, pp. 663-670
- [11] C. B. Ries. "ComsolGrid - Konzeption, Entwicklung und Implementierung eines Frameworks zur Kopplung von COMSOL Multiphysics und BOINC um hoch-skalierbare Parameterstudien zu erstellen." M.Sc. thesis, University of Applied Sciences Bielefeld, Germany, 2010.
- [12] W3C. "Extensible Markup Language (XML) 1.0 (Fifth Edition)," Internet: <http://www.w3.org/TR/REC-xml/>
- [13] J. McCutchan, R. Love, and A. Griffiths. "inotify - monitoring file system events," *Linux man pages*(7)
- [14] Message Passing Interface. "The Message Passing Interface (MPI) standard," Internet: <http://www.mcs.anl.gov/research/projects/mmpi/> [18th February 2012]
- [15] Object Management Group. "OMG Unified Modeling Language (OMG UML) Superstructure." formal/2010-05-05, May, 2010.
- [16] Object Management Group. "Object Constraint Language." Version 2.2, Feb., 2010
- [17] OpenMP. "The OpenMP API Specification for Parallel Programming," Internet: <http://www.openmp.org> [18th February 2012]
- [18] PKWARE. "APPNOTE.TXT - .ZIP File Format Specification." Version 6.3.2, Sept., 2007

Automatic Driving System Using LEGO

Chuanxi Zhou¹, Yujian Fu², and Mezemir Wagaw²

¹Department of Computer Science, Alabama A&M University, Normal, AL, USA

²Department of Food & Science, Alabama A&M University, Normal, AL, USA

Abstract - LEGO is not only a toy, but also a tool for education. By analyzing and solving the all sorts of problems with LEGO, students have better preparation for a career in science and engineer. Our LEGO-robot is built to pick up a ball, find its way to follow a color curved line, put down the ball at the destination, and show the sound values on the NXT screen. A touch sensor, light sensor, sound sensor, Ultrasonic sensor and three motors are used for the robot to complete this challenge. Our LEGO-robot automatic driving system is designed with UML. UML as a graphical modeling language is a standard way for visualizing, specifying, constructing, and documenting an Object-oriented software system. A class diagram in the UML is used to describe the information and relations about seven classes, including the CorrectingDirection class, the LegoMotor class, the SoundSensor class, the UltrasonicSensor class, the LightSeonsor class, and the TouchSensor class. A use case diagram is also used to describe our robot system's behavior as it responds to a request from the outside of the system. A state diagram is used to describe the internal behavior in a class with natural states. This project is implemented using Java with leJOS. The system works steady with expected design functionalities, which indicates that we have successfully designed the LEGO-robot automatic driving system using UML.

Keywords: UML, class diagram, use case diagram, state machine diagram, LEGO robot, software design

1 Introduction

LEGO is not only a toy, but also a tool for education. With LEGO, students can build models, utilize motors and sensors, learn how to design, program and control models. By analyzing and solving the all sorts of problems with LEGO, students move forward for a better preparation for science and engineer career.

Every year, the First LEGO League (FLL) sponsors a challenge tournament around the world. It is an international robotics team competition for children aged from 9 to 16. Today, FLL tournaments take place in more than 40 countries worldwide with participation of over 10,000 teams [4]. The tournament challenges vary in each year with the

basic skills and concepts hold the same. For example, the challenge of following color line appeared in both "The Smart Move Robot Game" (2009) and "The Body Forward Challenge" (2010). Therefore, in this project, we will focus on this challenge. Our LEGO-Robot moves forward following a color line, picks up a ball when the touch sensor is pressed, adjusts the direction if it needed, determines the terminus by pre-programmed distance, puts the ball down when it arrives the terminus, and detects the sound value and shows the value on the NXT screen. Our robot and the mat are shown in Fig 1.

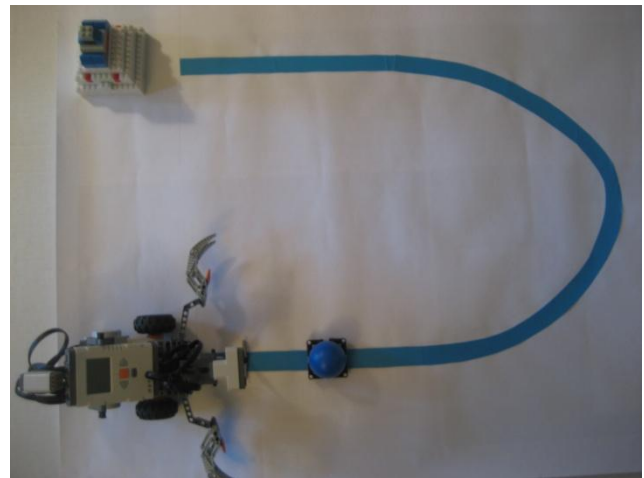


Figure 1, LEGO-Robot and LEGO-pad

This LEGO-Robot Automatic Driving System is designed with the Unified Modeling Language (UML). UML is a graphical modeling language which has been widely recognized as a standard way for visualizing, specifying, constructing, and documenting the artifacts of an Object-oriented software system [6]. It can also be applied for defining Software Architecture, modeling of business processes and their design, managing complexity, etc.

The remainder of this paper is organized as follows. We discuss the related works in Section 2. After that, the system design of automatic driving system (ADS) is presented in Section 3. The implementation in Java using LEGO NXT toolkit is shown in Section 4. Conclusion and future investigations are discussed in the Section 5.

2 Related Works

LEGO NXT tool kit is widely used for the educational and research purposes in academic now [2]. The Mindstorms NXT brick uses a 32-bit ARM processor as its main processor, with 256 kilobytes of flash memory available for program storage and 64 kilobytes of RAM for data storage during program execution. To acquire data from the input sensors, another processor is included that has 4 kilobytes of flash memory and 512 bytes of RAM. Two motors can be synchronized as a drive unit. To give the robot the ability to “see, the ultrasonic sensor, which is accurate to 3 centimeters and can measure up to 255 centimeters, and the light sensor, which can distinguish between light and dark, can be attached to the brick. A sound sensor that can be adjusted to the sensitivity of the human ear can be used to give the robot the ability to hear and react, if programmed, to noises. Finally, the two touch sensors give the ability for a robot to determine if it has been pressed, released, or bumped, and react accordingly [2].

Although LEGO NXT is a highly integrated and low cost educational settings, currently, there is not much work has been done for the designing of a reliable LEGO robot to realize the expected functionalities in literature. There are several approaches available for the software intensive design, UML is one of the most popular methods that is currently widely used in both industry and academic. In this section, we discuss some existing works on design of embedded systems using UML.

Grady Booch et al. [6], [1] introduced the UML 2.0 concepts and notations. Its advantages of allowing users to model everything from enterprise information systems and distributed Web-based applications to real-time embedded systems.

Saxena et al. [7] presented a UML model of multithreaded programs on a Dual Core processor. Performances of the programs in JAVA and C# on the basis of UML design were compared and evaluated.

João M. Fernandes [5] demonstrated the utilizing of UML to

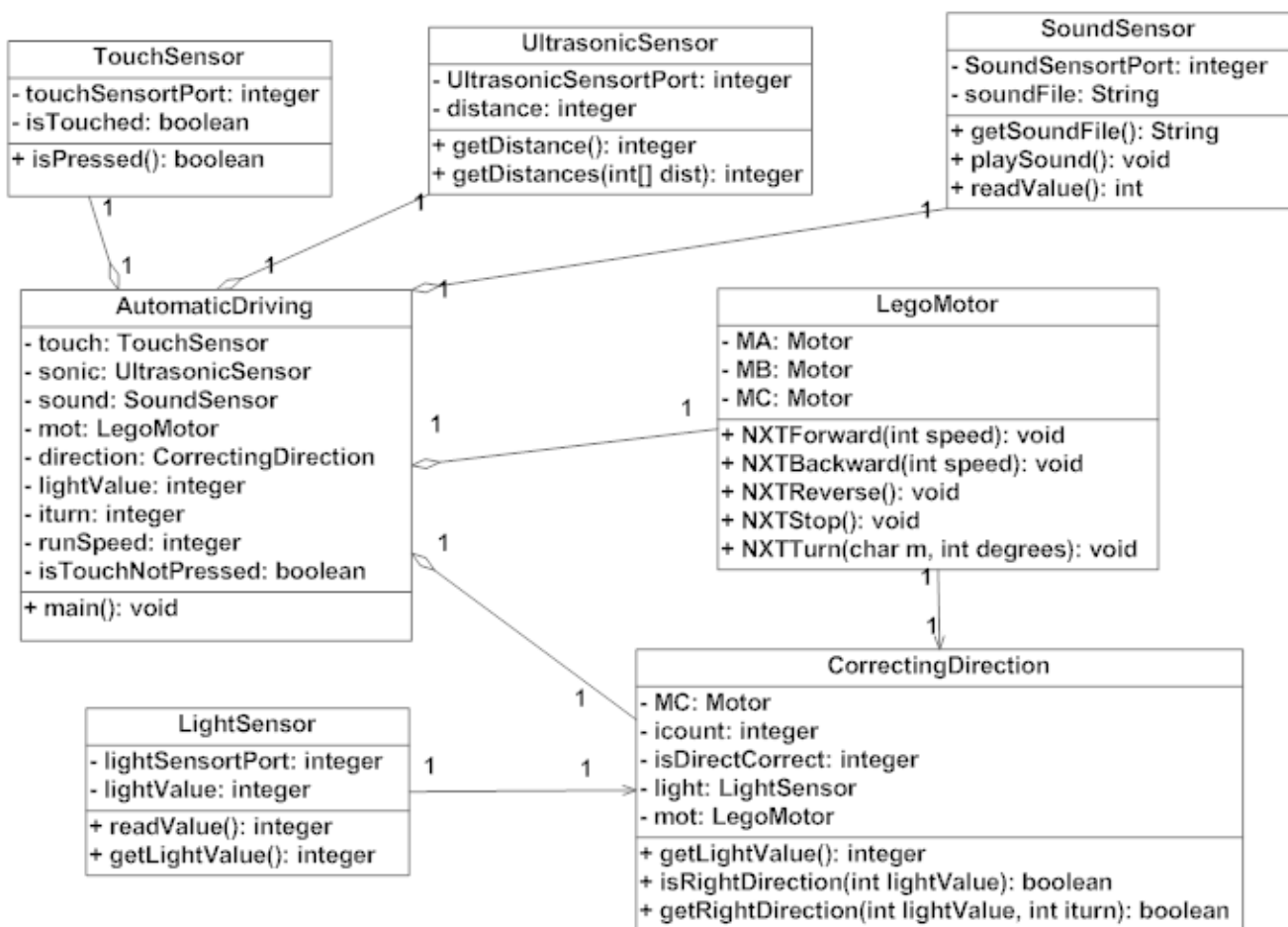


Figure 2 – Class Diagram for the LEGO-Robot Automatic Driving System

model industrial embedded systems. Using a car radios production line as an example, this study illustrated the modeling process following the analysis phase of complex control applications. The authors used some guidelines to transform the use case diagrams into a single object diagram to guarantee the continuity mapping of the models.

Our work is different from above works in following aspects: a) This work aims at developing a reliable robotics system using object oriented methodology – UML. Robotics systems are complex, concurrent embedded systems that include multiple objects and multiple interfaces. Our work has demonstrated that it is important to produce reliable robots for using UML design method. b) Our work applied UML design on the LEGO NXT settings, which provides a new successful application of UML on the embedded system design.

3 System Design – UML Model

In this section, we present our approach to design LEGO-Robot Automatic Driving System with UML.

3.1 Class Diagram

A class diagram in the UML is used to describe static information about classes, with operations and data (attributes), and to describe relations (including inheritance, aggregation, association, etc.) between different classes. As mentioned by Michael Blaha and James Rumbaugh [3], a “class diagram provides a graphic notation for modeling classes and their relationships, thereby describing possible objects. Class diagrams are useful both for abstract modeling and for designing actual programs.” Class diagrams are the mainstay of object-oriented analysis and design.

Fig. 2 shows the class diagram of our LEGO-robot automatic driving system. The NXT is the brain of a LEGO-robot. It's an intelligent, computer-controlled LEGO brick that makes a LEGO-robot alive and perform the programmed activities. The NXT has three output ports to attach motors and four input ports to attach sensors. The three motors are grouped into the LegoMotor class, which including the LightSensor class, The TouchSensor class, the UltrasonicSensor class, and the SoundSensor class corresponding to the light sensor, the touch sensor, the Ultrasonic sensor, and the sound sensor, respectively. For detecting and correcting the moving direction of the robot, the light sensor and motors are grouped into the CorrectingDirection class. The AutomaticDriving class is consisted of the CorrectingDirection class, the LegoMotor class, the SoundSensor class, the UltrasonicSensor class, and the TouchSensor class.

3.2 Use Case Diagram

A use case is a description of a system's behavior as it responds to a request that originates from outside of that system. In other words, a use case describes "who" can do "what" with the system in question [1]. The use case diagram of our LEGO-Robot Automatic Driving System is shown in Fig. 3. This figure clearly shows that a person can turn on the NXT, find certain program, run certain program, and turn off the NXT.

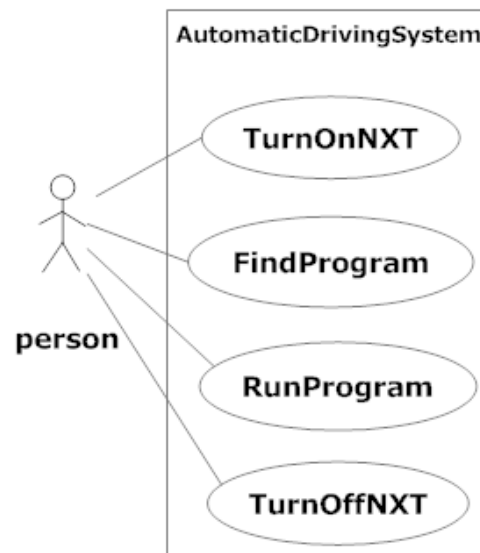


Figure 3 – Use Case Diagram for the LEGO-Robot Automatic Driving System.

3.3 State Machine Diagram

A state diagram has states, events and guards. They are used during the design phase to describe internal behavior in a class with natural states. State diagrams are used to give an abstract description of the behavior of a system. This behavior is analyzed and represented in series of events, that could occur in one or more possible states [1].

Fig. 4 shows the state diagram of our LEGO-robot Automatic Driving System. When the actor presses run on NXT, the LEGO-robot will firstly wait 5 seconds and let the actor to adjust the robot to the right location. After that, the robot is programmed to move forward. When the touch sensor is pressed along its way forward, the robot will firstly move backward to pick up and hold the ball. Then the robot will continue follow the color line which includes a half circle in the middle of the mat. The light sensor is used here for the robot to find the correct way to follow the curve. Here is how it does that, if the light sensor detects a different color value from the previous one, that means the robot is in the wrong direction, the robot will then turn around to look for the

previous color value. If the light sensor cannot find the same value after turning a certain degree (here we set it as 3400), the robot will then simply stop. The Ultrasonic sensor is used to find the destination of the robot. When the Ultrasonic sensor detects the pre-programmed distance, the robot will just put the ball down and stop. During the whole moving activity, the sound sensor is detecting the sound values in the environment and shows them on NXT screen.

not flat or something stuck underneath. But only the first time pressing will allow the ball to be picked up. 2) The light value for the light sensor should be set as a region not a single number. Because the color values on the line are not a constant number. There may have a place where is slightly darker or lighter than other places. In addition, under different illuminating conditions could influence the light sensor to define different values for the color line. 3) When

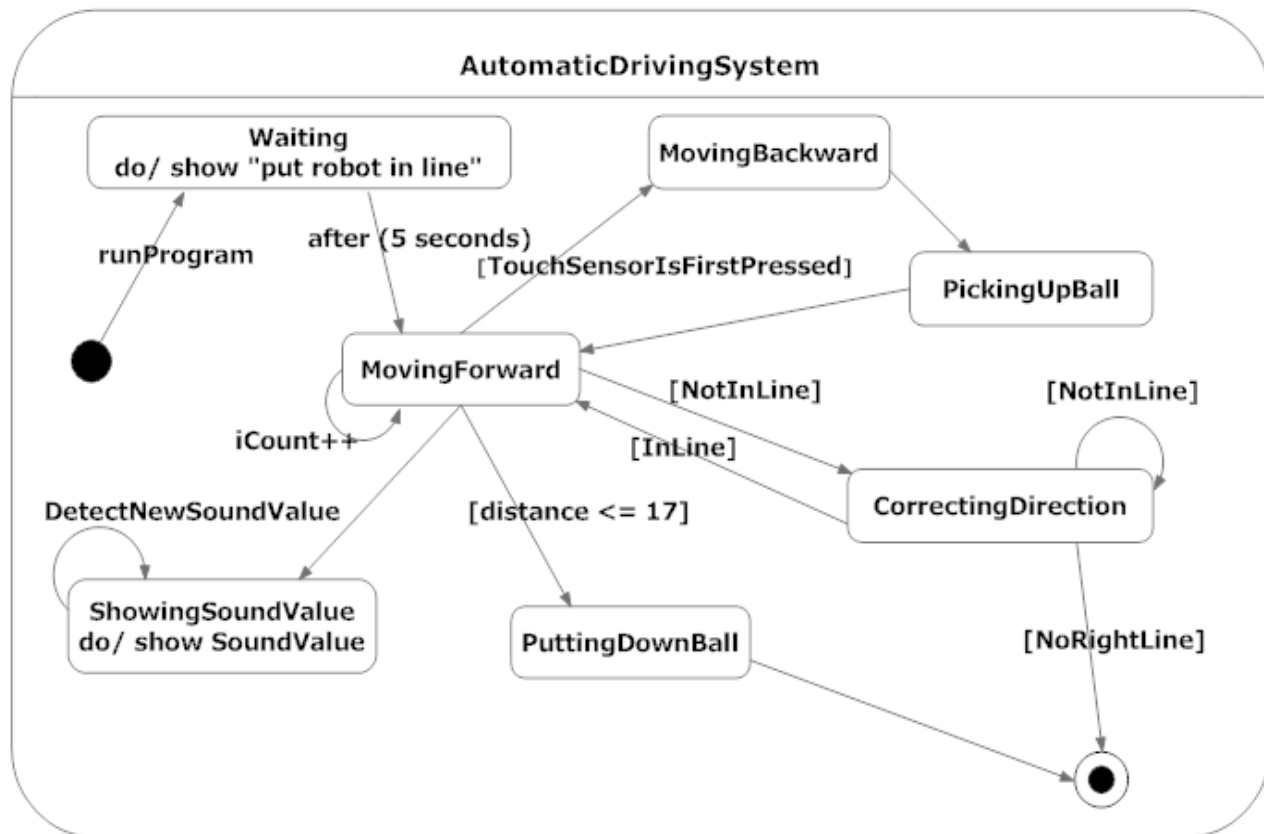


Figure 4 – State Diagram for the LEGO-Robot Automatic Driving System.

4 Implementation in Java

Java/leJOS is used in this project to implement our program. leJOS is a tiny Java Virtual Machine [8],[2]. LeJOS NXJ supports the NXT brick which allows us to code the LEGO-robot Automatic Driving System with the Java programming language. The Java coding gives the LEGO-robot a lot of abilities. Characteristics of Java are utilized in the coding, e. g., object-oriented, distributed, interpreted, robust, secure, architecture-neutral, portable, high-performance, multithreaded, and dynamic.

There are some issues that should be pay additional attention to when coding the program. 1) The robot will pick up the ball only when the touch sensor is pressed for the first time. The touch sensor could be pressed many times if the mat is

the robot moves to a region that has a different color value from the previous one, we programmed to let the robot turn left at a small angle, then right at a small angle, and so on, until the previous value is found. Or, the robot will stop if no correct way is found.

5 Conclusions

In this project, a LEGO-robot automatic driving system is designed with UML. This robot is built to pick up a ball, find its way to follow a color curved line, put down the ball at destination, and show the sound value on the NXT screen. A touch sensor, light sensor, sound sensor, Ultrasonic sensor and three motors are used for the robot to complete this challenge. The system is designed with UML. A class diagram in the UML is used to describe the information and relations about seven classes, including the

CorrectingDirection class, the LegoMotor class, the SoundSensor class, the UltrasonicSensor class, the LightSensor class, and the TouchSensor class. In this project, a use case diagram is also used to describe our robot system's behavior as it responds to a request from people. A state diagram is used to describe the internal behavior in a class with natural states. UML as a graphical modeling language is a standard way for visualizing, specifying, constructing, and documenting an Object-oriented software system. This project indicated with an example that UML can be used to design a system independently. This project is implemented using Java with leJOS. The system works steady, which indicates that we have successfully designed the LEGO-robot automatic driving system using UML.

In this project, the robot is designed to complete some basic activities. For more complex activities, the robot system can be upgraded with more equipment. For example, we also can use two light sensors to detect the correct direction for the robot. A camera can be used to detect a terminus. We can also add parts to control the activities of the LEGO-robot by sound commands.

Acknowledgements

We would like to thank Joseph Shi and Cheng Zhou for helping us to build the LEGO-robot. We would like to thank for all valuable comments of reviewers.

6 References

- [1] Unified modeling language (uml), version 2.0. Available from <http://www.omg.org/technology/documents/formal/uml.htm>.
- [2] B. Bagnall. *Maximum Lego NXT: Building Robots with Java Brains*. Variant Press, 2007.
- [3] M. R. Blaha and J. R. Rumbaugh. *Object-Oriented Modeling and Design with UML*. Prentice Hall, December 2004.
- [4] LEGO development team. LEGO education. Available from: <http://www.lego.com/education/competitions/default.asp>.
- [5] J. A. M. Fernandes, R. J. Machado, and H. D. Santos. Modeling industrial embedded systems with uml. In *Proceedings of the eighth international workshop on Hardware/software codesign*, CODES '00, pages 18–22, New York, NY, USA, 2000. ACM.
- [6] J. Rumbaugh, I. Jacobson, and G. Booch. *The Unified Modeling Language Reference Manual*. Addison-Wesley Professional, July 2004.
- [7] V. Saxena and M. Shrivastava. UML modeling and performance evaluation of multithreaded programs on dual core processor. *International Journal of Hybrid Information Technology*, 2, July 2009.
- [8] LEGO Team. LEGO NXJ technology. LEJOS APIs. Available from: <http://lejos.sourceforge.net/nxj.php>.

ClipBits – A Case Study in Model-Driven Software Engineering

Thomas D. Lovette, Devon M. Simmonds, Michelle A. Wilcox, Yuli Bonner
University of North Carolina Wilmington
601 South College Road, Wilmington, NC 28403
{ tdl6020, simmondsd, maw3067, myb7721 }@uncw.edu

Abstract

Model-driven software engineering is an approach to software development that centralizes the use of models and transforms the software lifecycle from a code-centric to a model-centric undertaking. This paper reports on use of a model-driven approach in the development of ClipBits – a lightweight clipboard management application, designed to support data reuse and increase user productivity by extending the functionality of the Operating System clipboard. ClipBits provides a dockable, lightweight, user-friendly interface that is intuitive and unobtrusive. The application allows a user to easily store text, images, and files in the form of ClipButtons that can be restored back to the system clipboard. In addition, these ClipButtons can be readily organized by storing them in user-defined Button Sets. Results and inferences are presented.

Keywords: software engineering, model driven engineering, software architecture, UML, clipboard.

1. Introduction

Software reuse [1] is an importance and pragmatic discipline in software engineering. Software reuse can lead to lower development costs and lessen time spent on deployment [2]. However, reuse of software is hard to achieve without some form of generalization [2]. The concept of generalization involves taking something that is a specific concept, and allowing it to be applied to wider and general principles. Software reuse can be approached from either data-centric or code-centric viewpoints.

From a data-centric viewpoint, the everyday use of computers and related technologies, provide many opportunities for generalization and reuse as mechanisms for increasing productivity. Indeed,

whether using a mobile device or a more traditional PC, application end-users often require the repeated storage and reuse of numerous types of data. Typically, users accomplish this task by copying data to the clipboard and pasting it where needed it as many times as necessary to meet the desired end. When the users have the ability to store more than one item on the clipboard, they are able to refer back to the generated clipboard history and reuse data more efficiently [3](Frakes).

In using the clipboard, a problem arises when there are multiple pieces of data that require replication in a nonconsecutive way. In such a case users are forced to refer back to some source to copy the data back onto the clipboard before they can continue the process of repeatedly pasting it. This process of referring back to a previous source every time a user wishes to access the data is tedious and time consuming. Furthermore, when a user quits working on a project for any duration, it is unlikely that any data to which repeated access is needed will still be on the clipboard.

Several clipboard management [4 – 11] software are currently available. Such software may be evaluated using two basic features: how clips are stored and flexibility – a measure of the number and types of data that the software stores and the range of programs with which the clipboard software can interact. Some available clipboard management applications have limited storage features and support only text while others allow for images and other types of data. Three examples of clipboard management software are Microsoft Office 2000 Clipboard [8], xNeat Clipboard Manager [4] and Ditto [5].

Microsoft Office 2000 Clipboard was an early attempt at clipboard management software. A clear limitation of Microsoft Office 2000 Clipboard is that it only works among the programs included in the MS Office suite. Another limitation of the software is that it can only store 12 past entries (later expanded to 24 in Office XP). xNeat Clipboard Manager [4] is an

example of a more recent clipboard management system. Concerning the storage of clip data, XNeat allows the user to define the number of clip entries that will be stored. The number of entries can vary from 1 to 99. Additionally, xNeat lets a user “sticky” entries which makes it so that those entries are always available for access. While these features are a welcome inclusion, beyond “stickying” xNeat does not offer any way to navigate through the list of entries. And with up to 99 entries trying to find the right entry can be a chore.

Finally, Ditto [5] is a third example of a clipboard manager. This software offers some improvements over xNeat but also suffers from similar shortcomings. Like xNeat, Ditto allows multiple data types and can be used in any environment and, also like xNeat, it can only be accessed through the system tray or with a hotkey command. While this may not be hindrance to some users, it may prove unintuitive for other and could be improved by allowing the option for a persistent GUI. One of the most significant improvements that Ditto has over xNeat is the ability to search through the clip history by entering a term in a search box. This might seem like an ideal way to navigate through the list of clip entries, especially since there is no limit to the number of entries in Ditto. However, the name that is generated for clip data is not always indicative of the content of that data and there is no way to rename the clip. Thus, the search term may not yield the desired results.

The paper reports on design and development of *ClipBits* – a lightweight clipboard management application that supports data reuse by offering features that complement the system clipboard in any environment in which it might be used. The application is designed with an intuitive, user-friendly graphical user interface through which users can access an unlimited set of past clipboard entries. A model-driven engineering (MDE) [12 – 14] approach was used for the project. MDE was used because while software may be inherently complex [15], MDE provides mechanisms and tools that are better able to address complexity by removing some accidental complexities associated with code-centric development. Our goal is both to encourage use of models as well as provide

case study data that may be used by others in the scientific community.

One of the main goals of the ClipBits application is to keep it as unobtrusive as possible. Its purpose is to enhance the user’s productivity while they simultaneously work on other applications. As such, ClipBits is developed to be as unobtrusive and non-distracting as possible, so that disrupting or inhibiting a user’s workflow is minimized.

2. Software Design

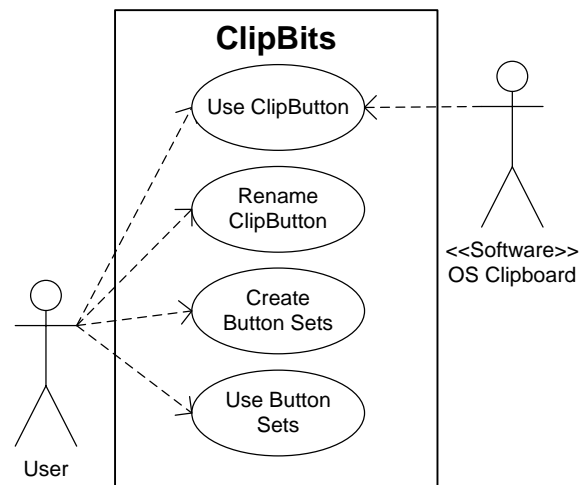


Figure 1. ClipBits Use Case Diagram

ClipBits was designed using a Model-Driven approach, which includes UML [16] models such as use case (See Figure 1), activity diagram (see Figure 2), architectural diagram (see Figure 3), and Class diagram (see Figure 4).

The Use Case Diagram (Figure 1) demonstrates typical actor interaction with the ClipBits software. Actors include the end User and the Operating System Clipboard, both of which interact with the application. However, the methods through which they interact are slightly different. The end User interacts with ClipBits through the use of Clip Buttons and UI components, whereas the OS Clipboard interacts with the application via Windows messaging.

The Activity Diagram above (see Figure 2) demonstrates a high-level overview of user interaction with the ClipBits software. The only available entry point into the application is through a typical program launch, and the only intended way to exit the application is to click “Exit” in the UI. All operations

Of the numerous architectural styles in use today, the relative merits of two styles were evaluated: a traditional Layered Architecture and the Model-View-Controller (MVC) pattern. Each style offers features that, while similar, differ enough to warrant a closer

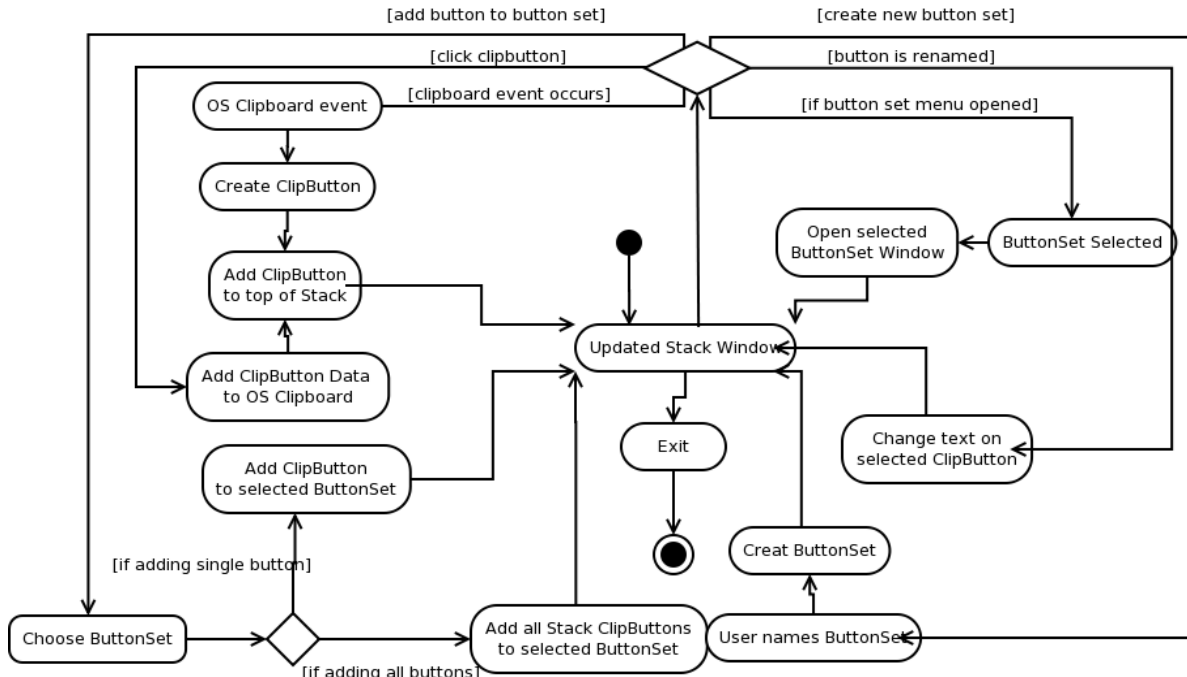


Figure 2. ClipBits Activity Diagram

proceed from the program in a loaded state, and all interactions ultimately terminate in an updated User Interface and Stack Window.

2.1. Architectural Design

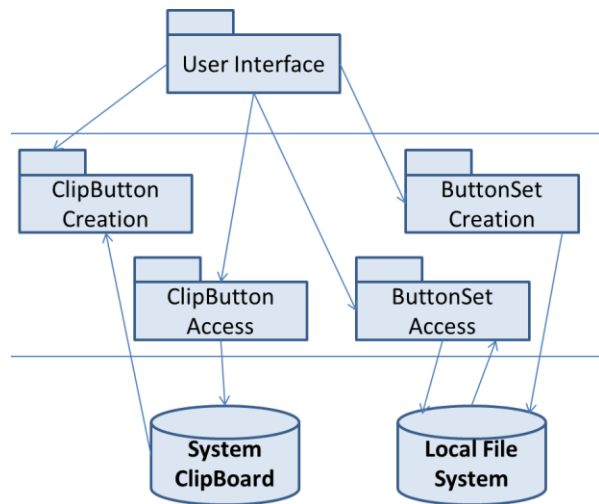


Figure 3. ClipBits Layered Architecture

look. Both styles were found to be compatible with the implementation of the ClipBits architecture, but a layered approach presented a best-fit for this type of application given its desktop-based nature.

As Figure 3 demonstrates, a layered architectural style allows for a clear separation of concerns. In this model the different areas into which the program is divided include the UI layer, the logic layer and the data layer. Each of these layers encapsulates a set of concerns with little to no overlap of functionality between these sets. Furthermore, the points of contact between these layers are clearly defined.

The enhanced modularity afforded by the layered architectural approach lends adaptability and expandability to ClipBits, making it easier to support. New modules can be added to a layer without directly affecting the functionality in the other layers. This means that any bugs introduced by an addition would be isolated from the other layers, preventing large scale instability due to expansion.

The clear separation of concerns provides an ideal strategy for distributing development tasks among developers. In the initial construction of the system, the layered architecture allows developers to work on

different layers concurrently without experiencing major conflict. This will increase the speed at which the system can be developed as well as assist in the minimization of bugs introduced during the integration process.

With the potential for increased development speed and enhanced bug mitigation, the layered architecture presents an excellent framework for any system that can be easily separated into areas of functionality.

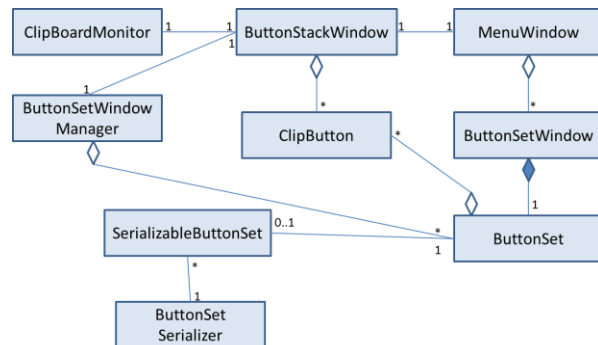


Figure 4. ClipBits Class Diagram

A layered architecture provides many benefits. One such benefit is the clear definition of boundaries between different functionalities (see Figure 15). These boundaries along with well-defined points of contact enhance the modularity of design as well as providing for separation of concerns.

2.2. Design Constraints

Using the Object Constraint Language (OCL) [16], we were able to define fairly standardized constraints which were applied to various structures within the application. The diagram below demonstrates a sample of such constraints.

The Object Constraint Language provided several benefits during the identification of design classes. For instance, it was useful to define the Button Stack Window (class from Figure 4) as an OCL context containing pre-and post-conditions on operations such as user interaction with UI elements. One such case involves the user's interaction with the "viewSetButton" control. When the proper delegate method (viewSetButton_click(...)) is executed, the subsequent operation should be designed in such a way as to conform to the constraints specified by the "post:" attribute. In this case, a UI window named "SetMenuWindow" is visually activated.

3. Results

The Model-Driven development of ClipBits produced excellent results. User Interface designs were

easily implemented, test cases were largely successful, and concerns regarding security, ethicality, and societal impact were addressed effectively. The results of development are expounded in the following sections.

3.1. User Interface

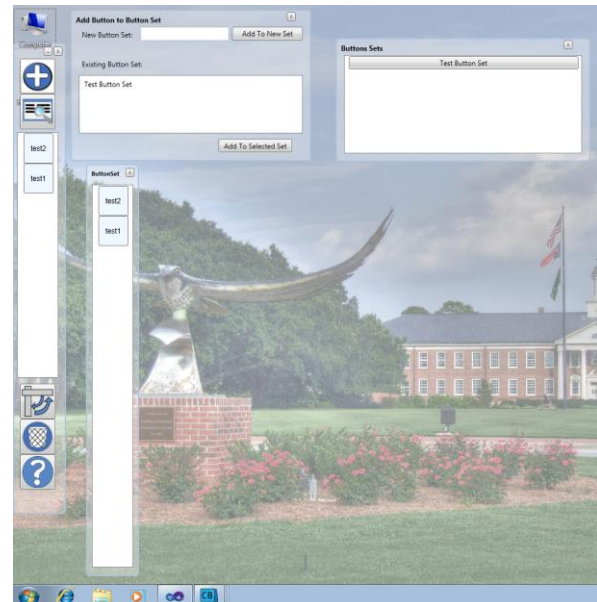


Figure 6. ClipBits User Interface

Figure 6 depicts all of the UI Windows with which a ClipBits user may interact: the Main Window, Add All Buttons to Button Set Window, Button Set Menu Window, and finally the individual Button Set window displaying the ClipButtons contained in the Button Set.

3.2. Description of Implementation and Testing Process

Once the test cases were decided upon, the integration of each team member's functions commenced. The clipboard monitor functions, Button Set serialization functions, and user interface design all needed to be put together and meet all of the software's functionality requirements. Once all of the class files were integrated into one project, the immediate task that arose was making all of the class functions compatible with other class functions through integration and systems testing. This was accomplished by executing the test cases and correcting errors as they arose. Multiple problems surfaced throughout this testing phase and with the team member who had written the code for which the bug was found leading, the modification process went fairly smoothly.

3.3. Addressing Ethical, Legal, Security, and Societal Implications

Several ethical issues [18-24] arise when dealing with the copying and transmission of data, including the legality of the entire process. Illegal copying of data such as music, images, and written works carries stiff fines throughout most of the world, and the threat of misuse by a user of an application that allows for the copying of data is always pertinent. The right of musicians to publish their own works is of special importance in this day and age. Cockburn reminds us that “the recent spate of lawsuits against individuals accused of sharing music files attest to the seriousness...[of] these rights [24].” With digital piracy on the rise, great care must be taken to ensure that any application dealing with the copying and/or retransmission of intellectual property be thoroughly secured. There is, however, a fine line between what is considered to be fair use and outright piracy [18].

Copyright issues are of prime concern to ClipBits, as the application allows users to generate Button Sets representing data that is of recurring utility. In and of itself, the creation of a personal, archival copy presents neither an infringement nor a violation of copyright law, as the Audio Home Recording Act of 1992 explains [20].

The important question for this project, is whether ClipBits is liable for any damages caused by unauthorized redistribution of copyrighted material via its software features? The answer lies in a little-considered but very important aspect of nearly every software package in existence today: *the End User License Agreement* (EULA). End User License Agreements are often used “...to impose restrictions on consumers regarding what they can legally do with their purchases” [21]. Therefore, the final deployment of ClipBits includes a legally-binding EULA specifying that users may not use this software to redistribute copyrighted materials without express permission from the copyright holder.

One of the many functions of technology is to increase productivity. Technology can help to accomplish this by making work easier and faster. Indeed as a result of information technology productivity has continued a trend of steady improvement for some time [25]. And as technology becomes more and more ubiquitous the impact it has on production may become incalculably large.

It is testament to the impact benefits of technological tools that many fields such as business or banking have integrated a technological framework intimately into the work environment. In many circumstances where a

software system once aided in a particular job, the job is now wholly encompassed by a software system. In order to increase productivity in these instances, we need secondary software tools that augment or enhance the primary software environment of the workplace. It is into this class of software designed to enhance other software into which ClipBits falls.

When used with other productivity software such as software development environments or desktop publishing suites, ClipBits can help to trim down the amount of time it takes to complete tasks. Over the course of many hours of work what may seem at first to be a marginal time savings can accumulate in to something substantial that directly translates into business savings.

As software enhancing products become more widely developed and more ways in which they can be applied become identified, they could very easily help to propel the trend of continued increase in productivity into the foreseeable future. And such an increase in productivity can help to strengthen the economy and thus, have a positive bearing on society as a whole.

It may seem a little odd to suggest that a niche product like ClipBits could have an impact on society, and taken as an isolated product such a suggestion probably would be absurd. However, taken with other similar products, the collective impact could be every bit as substantial as the introduction of primary productivity software such as word processors or spread sheets. In a world where human tasks are growing more and more specialized, it only seems appropriate to develop software for more and more specialized tasks, even when those tasks are to be performed using other software.

3.4. Lessons Learned

There were many challenges in the process of developing ClipBits. Along the way there were various mis-estimations and other unexpected hurdles. We did our best to adapt to unforeseen circumstances and persevere through the stress of tight deadlines. From a managerial standpoint, the implementation of ClipBits produced many opportunities for learning. Budgeting of time played an immense role in the process of development, as the team oftentimes had to work within the confines of a mere three hours together each week. This, however, encouraged greater communication within the team outside of scheduled meeting times, as problems were often discussed through email, and resolutions were proposed.

An early challenge was unexpected behavior from the IDE while developing the clipboard monitor.

Whenever the prototype was run from the IDE the messages that signaled that a clipboard event had occurred were being duplicated. The source of the anomaly was not obvious and ferreting out the cause was quite a challenge. Another issue that was of some concern was the compatibility between the different components that we were developing. Some of the core components, which we had originally assumed to be compatible, proved to be completely incompatible and required some conversion in order to work together properly. While the discovery of this incompatibility came as something of a shock, the rectification was not as difficult as it might have been.

The original cost and effort estimates for ClipBits were slightly misjudged. Original estimates called for 5,407 lines of code, but the final product contained far less: approximately 2,000. This is due largely to uncertainty about the extent to which Windows Presentation Foundation would be used in development. At the project's inception, WPF was intended for use in a strictly UI fashion, but as the project progressed, it assumed a more prominent role, as its declarative syntax showed it to be ideal for data-binding scenarios. Thus, operations that would have taken many lines of code to implement (such as UI updates) were relegated to the framework, drastically reducing the amount of necessary coding.

Implementation of ClipBits' Data Access Layer proved to be more difficult than originally expected. Though the final implementation differed only slightly from the original concept, the task of integrating prototypical work with the complete solution presented numerous problems, many of which were related to the use of WPF and data-binding. For instance, the maintenance of data-bound lists proved to be somewhat difficult, as WPF reserves a copy of whatever data is being bound to a list (such as a UI representation of a ClipButton) for a specific UI thread, which effectively locks it from other windows. Care was required to ensure that these memory references did not overlap. However, all problems were, in the end, effectively solved.

In general, the difficulties our team faced were not as bad as they might have been because our members were capable and hardworking. Indeed, the importance of the team itself cannot be overestimated, because ultimately any resulting success was a direct product of the people involved.

4. Conclusion and Future Work

The creation of the ClipBits software solution demonstrated a number of important lessons regarding

the use of a model-driven development methodology. First, the use of models during the requirements phase gave a clear direction to the team concerning the decided course of action, and provided an excellent visual resource from which to draw ideas during later iterations. The design phase greatly benefited from the model-driven approach; UML class diagrams combined with OCL constraints allowed the team to visualize patterns between program elements in ways that could not be readily achieved otherwise. Finally, the models presented a useful reference for the implementation and testing phases; code development was facilitated by the granularity of feature explanation in class and architectural models, and testing greatly benefited from such items as the OCL diagrams in terms of pre-and post-condition evaluation. Testing also was made easier by the inclusion of activity diagrams, which enabled the team to validate the agreed-upon and established workflow.

The future of ClipBits is very bright. While the development team is busy using the software in separate personal endeavors, important information is being gained with regard to usability, architectural optimizations, and overall utility. Future versions may include far more advanced features, but the guiding principles of simplicity and ease of use will still apply, as will the guidance of the models outlined herein.

References

- [1] Iam Sommerville. *Software Reuse in Software Engineering* 8th Ed., Chapter 18, pp 241 – 265, Addison-Wesley, 2007.
- [2] Damasevicius, Robertas. "Analysis of Components for Generalization using Multidimensional Scaling." *Fundamenta Informaticae* 91.3/4 (2009): 507-522. *Academic Search Premier*. Web.
- [3] Frakes, Dan. "Clipboard Managers." *Macworld* 27.7 (July 2010): 36-37. Web.
- [4] *xNeat.com*. xNeat Clipboard Manager. 2011. Web. 5 Nov. 2011.
- [5] <http://ditto-cp.sourceforge.net/>. Ditto Clipboard Manager. Web, Nov. 2011.
- [6] Stone, David "The Office Clipboard." *PC Magazine* 21, Dec. 2001: 83-86. Print.
- [7] Zardetto, Sharon. "Two Quick Copy and Paste Tricks." *Macworld* 27.4 (April 2010): 53. Web.

- [8] Microsoft.com. *How to use the office 200 clipboard*. Web, Nov. 2011, URL: <http://support.microsoft.com/kb/221190/en-us>.
- [9] Li, Shaobo, Lv, Shulin, Jia, Xiaohui, and Shao, Zhisheng. "Application of Clipboard Monitoring Technology in Graphic and Document Information Security Protection System." *Intelligent Information Technology and Security Informatics* (April 2010):423-6. *IEEE*. Web.
- [10] Pomeroy, Bryony and Wiseman, Simon. "Private Desktops and Shared Store." *Computer Security Applications Conference, Annual* (December 1998): 190-200. *IEEE*. Web.
- [11] Birss, Edward W.. "The Integrated Software and User Interface of Apple's Lisa." *National Computer Conference* (1984):319-28. *IEEE*. Web.
- [12] R. France and B. Rumpe. Model-driven development of complex software: A research roadmap. In *FOSE '07: 2007 Future of Software Engineering*, pages 37.54, Washington, DC, USA, 2007. IEEE Computer Society.
- [13] B. Selic. The pragmatics of model-driven development. *IEEE Software.*, 20(5):19.25, 2003.
- [14] Simmonds, D. M., Reddy, Y. R., Song, E. and Grant, E. "A Comparison of Aspect-Oriented Approaches to Model Driven Engineering", in Proceedings of the International Conference on Software Engineering Research and Practice, (SERP), 2009.
- [15] Fred Brooks, "No silver bullet: Essence and accidents of software engineering," *IEEE Computer*, 20(4):10-19, April 1987.
- [16] The Object Management Group (OMG). Unified Modeling Language: Superstructure. Version 2.2, Final Adopted Specification, OMG, <http://www.omg.org/uml>, February 2010.
- [17] The Object Management Group (OMG). Object Constraint Language 2.0, URL: <http://www.omg.org/spec/OCL/2.0/>, May 2006.
- [18] "Copyright And Fair Use." *ASHE Higher Education Report* 34.4 (2008): 31-52. *Academic Search Premier*. Web. 2 Nov. 2011.
- [19] Collins, Steve. "Digital Fair Prosumption And The Fair Use Defence." *Journal Of Consumer Culture* 10.1 (n.d.): 37-55. *ISI Web of Knowledge - Web of Science*. Web. 2 Nov. 2011.
- [20] Gaffney, BJ. "Copyright Statutes That Regulate Technology: A Comparative Analysis Of The Audio Home Recording Act And The Digital Millennium Copyright Act." *Washington Law Review* 75.2 (n.d.): 611-641. *ISI Web of Knowledge - Web of Science*. Web. 2 Nov. 2011.
- [21] Langenderfer, Jeff. "End-User License Agreements: A New Era Of Intellectual Property Control." *Journal Of Public Policy & Marketing* 28.2 (2009): 202-211. *Business Source Premier*. Web. 2 Nov. 2011.
- [22] Pomeroy, Bryony and Wiseman, Simon. "Private Desktops and Shared Store." *Computer Security Applications Conference, Annual* (December 1998): 190-200. *IEEE*. Web.
- [23] Rubenking, Neil J.. "Windows Security: This Time for Sure!" *PC Magazine* 24.15 (September 2005): 108-184. *Business Source Premier*. Web.
- [24] Cockburn, Brian. "Music And Copyright (Review)." *Notes* 62.1 (n.d.): 104-106. *Project MUSE*. Web. 2 Nov. 2011.
- [25] Champy, James. "Productivity Promise." *Financial Executive*. Oct. 2003: 35. Print.

Virtual Platform Generation Tool for Embedded Systems Design.

S. Villa, J. Villa, J. Yepes, J. Aedo

ARTICA, Microelectronic and Control Research Group
Electronics Department, Universidad de Antioquia,
Medellín, Colombia

Abstract – With the increasing complexity of embedded systems, to increase productivity in the design process has become a research area of great interest. For this reason, in recent years the literature has developed a series of strategies or methodologies to obtain dependable and flexible models in a short time. To support these design methodologies will be presented a rapprochement between the model of an embedded system and its corresponding executable model, based on model transformation and code generation from UML to SystemC code. It was found that transformation process will provide the exploration and system functional verification before being implemented.

Keywords - UML, MARTE, SystemC, Embedded System, Wireless Sensor Network, Code Generation, Simulation, Automatic Generation.

1. INTRODUCTION

The complexity of embedded systems has been increasing steadily in recent years. Currently, there is an increase in the complexity of the designs approximately 60% each year [1]. Moreover, has faced the growing complexity in the design by combining methodologies, skills of the designer and tools. Process led by the electronic design automation (EDA), which has achieved an increase in design efficiency by 21% over the same period of time [1]

Since the rate at which increases productivity is lower than the growth of complexity, it is increasingly difficult to achieve dependable designs, with development times and costs reasonable. This productivity gap in system design is due to design methodologies do not take full advantage of current technological advances offered by the integrated circuits industry.

In order to make optimal design, embedded systems modeling and simulation are becoming an important area of research. Also, for greater productivity in the design process, the literature has proposed the System Level Design (SLD) methodology, which suggests that the initial design stages are made at high level of abstraction, omitting the implementation details [2][3]. This involves

specifying the functionality of the system without having defined how it will implement.

This research supports the SLD methodology, proposing an approach for modeling an embedded system from the requirements. In this context, the Unified Modeling Language (UML) has been introduced to support the specification, design, and verification stages in the development process, describing both structure and functionality. A software for automatically generating code from UML models to SystemC executable models has been presented.

This paper is organized as follows: Section II presents the related work. Section III describes the transformation process. Section IV illustrates the process through a case study. Finally, Section V presents conclusions and future work.

2. RELATED WORK

In order to perform complex system designs, the software community has selected UML as standard modeling language. Moreover, system simulation is carried out by system description languages such as SystemC. Thus, a design trend is the integration of these two modeling languages, to obtain models based on HW/SW from high abstraction levels, appearing the automatic code generation between UML and SystemC.

Previous works in the transformation of code ([4] [5]) present a comparison between UML and SystemC. The purpose of the comparison is to find and motivate the mapping rules for automatic SystemC code from UML, which is one of the major steps in this investigation. The focus is on concepts that are equivalent in both languages, and the concepts and constraints that are present in each of the two languages. These works presents a series of formal rules between UML to SystemC for system platform (not functional model), but these do not present a modeling of an embedded system and the corresponding behavior transformation at high level of abstraction.

Seeking to improve the model transformation from UML to SystemC, works like [7] present a strategy to

exploit the capabilities of both languages by creating a profile UML/SystemC. This profile allows you to capture the structural characteristics such as system functional. However, this project only is possible to model the behavior of not very complex systems. Currently, this research seeks to describe the behavior of systems using various UML diagrams such as state diagrams and/or activity diagrams to allow for greater flexibility.

In [8] shows a way to convert the models developed in UML to SystemC simulation models. Initially, they developed a model with SysML sub-profile, bringing them closer through a series of restrictions. They performed a transformation 1 to 1 (diagram/code) between the models. Although this research presents an interesting approach in the process of transformation, which leads directly to the implementation models, it does not focus on embedded systems.

Trying to correct this, in [9][10][11][12], authors perform real-time systems models using SysML and MARTE profiles and employing Papyrus tool [13] (an open source tool based on Eclipse environment for modeling using UML2.0) and SC2 (based code generator models). The code transformation is divided into two parts, both based on the SystemC language. The first one models the behavior by *sc_methods* providing functionality. The second one the model structure by *sc_modules* connected by *sc_ports*. This research develops an innovation over others, allowing the insertion of code using Scripts. But the use of two profiles for the description of real-time systems, mean that the model is somewhat complex. Moreover, the embedded system behavior is modeled only by state machines, limiting the code generation.

Finally, [14] a design environment called Gaspard2 was presented. The process starts from a model made in MARTE, which represents a high-level model. This model is directly converted to a specific domain model. Then, this is transformed into a polyhedron model, in which the application is reconfigured and divided into different processing units. This derived model is transformed into a cycle model, where the application is represented according to a traditional pattern of nested loops. After these transformations, the simulation level is specified, such as SystemC TLM level. This environment focuses on MPSoC (Multiprocessor System-on-Chip), but not in WSN.

As illustrated by the literature, it is possible to adapt UML for modeling embedded systems. And although UML lacks real-time features, it has profiles that limit the domain of application according to the designers needs. Thus, in comparison with existing efforts, the approach this research approximates the modeling language (UML) to target language (SystemC) through the UML/SystemC profile; while real-time requirements are modeled through the MARTE profile. In this way,

models that approximate the real system and are easily adjustable to the SystemC code are obtained.

Therefore, this study develops a tool to automatically generate code from UML to SystemC models, aimed at the functional simulation of embedded systems.

3. METHODOLOGY

As mentioned above, there are studies that seek to transform UML models to SystemC code. Many of these studies perform an incomplete system model; some of them focus on the system structure without addressing the behavior, while others only seek to represent the behavior regardless the system physical architecture; and those which try to cover both fronts are not able to model complex systems.

In this research the solution to these problems by implementing the transformation process shown in Figure 1 was sought. It started with the system specification using the UML 2.0, and then there were the automatic generation of SystemC code through a series of analysis. Finally a simulation and validation model was obtained.

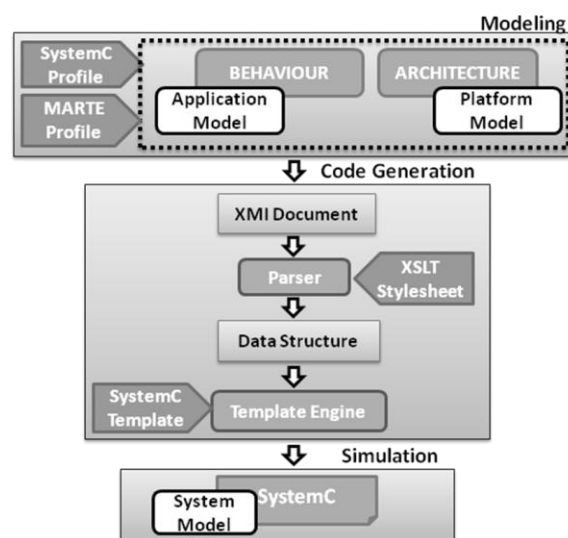


Figure 1. Transformation Process.

3.1 Modeling System

The transformation process begins with a system description at a high level of abstraction. Functional model (application model) and physical structure (platform model) are separately described. Thus, it is possible to make a platform independent application, obtaining a greater flexibility in designs.

The physical structure describes platform execution including processing units (such as CPU), storage units (such memories), and communication modules. In the physical structure the platform is empty, since the CPU does not execute any instructions and components are not connected with logic yet.

The functionality describes the application with real-time constraints. The functions are synchronized through

events, exchange of variables and message queues. The application model has two aspects: the structural aspect responsible for encapsulating the various functions which provides paths for exchanging data among them; the behavioral aspect provides flow used in data processing and sync functions.

Finally, a virtual system is the result of the mapping stage, which involves the functions allocation on the various modules of the platform; i.e. the application model is projected on the model platform to add physical constraints, such as scheduling, communication rate and memory size. The assignment can be deemed as a model itself.

3.1.1 UML diagrams selection

Due to UML is a really broad language, it is necessary to make a diagrams subset selection for the research domain. It was found that the diagrams that best fit the research objectives are: package diagram, class diagram, composite structure diagram, state diagram, activity diagrams.

3.1.1.1 Package Diagram

Package diagram is used to group elements, providing an order in the system modeling. This diagram reflects the separation between application models and platform. Thus, functional package contains the system behavior. And component package contains the elements of the platform model, which can be reused.

3.1.1.2 Class Diagrams

These diagrams are used to model the static system structure, which defines its elements, relationships, ports and interfaces. The elements describe two aspects:

- *Components for platform model* that will be part of physical system structure, these describe the basic SystemC entities (such as modules and channels).
- *Structures for application model* which encapsulate the behavior of the system, declaring attributes and methods of a function, they also represent the elements hierarchy.

3.1.1.3 Composite Structure Diagrams

These diagrams describe the internal structure of models. There diagrams show the modules, channels, ports and connectors, and how the communication takes place among them. This communication is carried out through ports, which provide/require a service. Also allowing improving the research designs and facilitating the generation of code.

3.1.1.4 State Diagrams

State diagrams are used in the functional architecture, modeling the system dynamic aspects. The transition among states is triggered by events (sensitive signals). These diagrams define the system behavior at a high abstraction level.

3.1.1.5 Activity Diagrams

Activity diagrams can describe the internal workings of the methods made by state diagrams; that is, they offer the possibility to specify more thoroughly the system operations. Additionally, these diagrams allow bringing the application model to SystemC methods.

3.1.2 UML profile

To extend the capabilities of research tool a set of extensions called profiles was selected. In particular two profiles –UML/SystemC and UML/MARTE– have been applied.

3.1.2.1 SystemC profile

This profile is disposed to map UML models developed in SystemC code. It defines a set of constraints and similarities between both languages, as shown in Table 1.

UML	SystemC	Stereotypes
STRUCTURAL		
Class	Module	<<sc_module>>
Class	Primitive Channel	<<sc_prim_channel>>
Class	Hierarchical channel	<<sc_channel>>
Class	Signal	<<sc_signal>>
Class	Fifo	<<sc_fifo>>
Port	Port	<<sc_port>>
Port	Input/Output	<<sc_in/out>>
Connector	Connector	<<sc_connector>>
Interface	Interface	<<sc_interface>>
BEHAVIOR		
State machines	Method	<<sc_method>>
State machines	Thread	<<sc_thread>>
Operation	Constructor	<<sc_constructor>>
Operation	Event	<<sc_event>>

Table 1. SystemC Profile

This table shows that the SystemC profile has been divided into two parts: the first one it related to model physical architecture (structural) and the second one to their behavior.

3.1.2.2 MARTE profile

MARTE profile will allow modeling some hardware devices and real-time constraints. For modeling hardware components, MARTE has the Hardware Resource Modeling (HRM) sub-profile, with which can represent certain resources, such as: a set of processing resources (<<HwComputing>>), devices involved in the communication process (<<HwCommunication>>), memory resources (<<HwStorage>>), assistive devices that are not so critical (<<HwDevice>>) [15].

MARTE also offers another subprofile to behavior model called Software Resource Modeling (SRM). This sub-profile can represent entities that compete for computing resources by <<SwConcurrency>>, synchronization mechanisms to control the flow of execution by <<SwInteraction>>, interfaces between peripherals and software implementation support <<SwBroking>> [15]. Furthermore, it allows adding time's constraints and events to the functional model

3.2 Code Generation

The next step in the transformation process is the code generation and definition rules between UML and SystemC models. This is accomplished through a series of analysis to different system representations. The first representation is described by UML diagrams with MagicDraw tool [16]; this makes it possible to generate a second code representation XMI (XML Metadata Interchange). XMI is a standard language which allows the easy exchange of data among modeling tools, specializing in models based on UML [17]. This XMI code is captured by our tool, which is integrated in the Eclipse environment, where it applies a XSLT Stylesheet (eXtensible Stylesheet Language Transformations) by Saxon-B open source processor which performs a filtering code, and changes it to another XMI document. The new document contains only XMI information, eliminating irrelevant data and providing continuity in the next step.

Then, within the same working environment, a data structures generator, called DOM was applied. It transforms the XMI document to a tree-like structure in Java. With this representation, the model hierarchy is captured, obtaining the structured elements (hardware, attributes and methods). This structure is organized into arrays with predefined classes.

Thus, the data structure fits into a SystemC template using the template engine Freemaker. This is an engine to generate HTML output, but can also be used to generate the standard text code [17]. The data structure defines the model, while the template indicates how this model should be organized in a format source. Thus, the data structure, the template and the java application produce the output source code in specific language, which in this case is SystemC. A template with Freemaker syntax based on the system data structure and target language, which allows expressing the system in SystemC code was created. For the sake of simplicity, in Figure 2 shows a small part of the template.

```
#include "${SubMod}.h"
class ${modelo.NAME}: public ${modelo.IDENT}{
public:
// module port declaration
${port.TYPE}<${port.INTERF}<${port.SIZE}>>${port.NAME};
//submodule declaration
${SubMod.CLASS} *${SubMod.NAME};
//process declaration
${(Funcnt.VISIBILITY)}:
${(Funcnt.RETURN)}${(Funcnt.NAME)}(${par.TYPE}${par.NAME});
private:
${(Funcnt.RETURN)}${(Funcnt.NAME)}(${par.TYPE}${par.NAME});
. . .
// constructor
SC_HAS_PROCESS(${modelo.CLASS_NAME});
${modelo.CLASS} (sc_module_name name:sc_module(name){
//module local channel implementation
${Channel.NAME}=new${Channel.TYPE}(${Channel.DATA});
//submodule port binding
${SubMod.NAME}.${SubMod.PORTS} (${SubMod.BINDING});
. . .
}
//member variables
private:
${var.type}<${var.size}> ${var.name}=${var.value};
. . .
};
```

Figure 2. Freemaker template.

3.3 Simulated model

Finally, system model in SystemC language was obtained. This model allows the simulation to validate platform architecture and application. This simulation is carried out under the same Eclipse environment, which should include the SystemC-2.2.0 library [18]. Additionally, more features to generated code to make measurements of any of the quality factors mentioned at the beginning can be manually added.

4. CASE STUDY: WIRELESS SENSOR NETWORK

This case study aims to model a wireless sensor network. It consists on a series of small smart devices, called nodes, which are spatially distributed on body; it communicates wirelessly with other nodes; it has the ability to sense vital signs and some computing power.

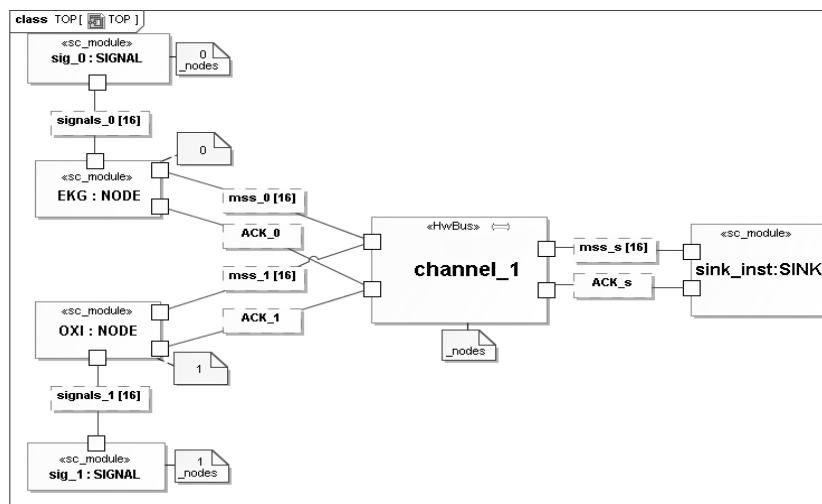


Figure 3. Wireless Sensor Network Model.

This model system consists in 3 nodes and a wireless channel (Figure 3). A node called Sink, which is in charge of gathering information. The other node will sense the oxygen saturation (OXI Node), and last one sense the cardiac activity (EKG Node). Also, the node behavior can be change by sense motion (Accelerometer Node) or blood pressure (Pressure Node).

The code generation process followed in this research begins by modeling the system components. Then, this section lists each of the diagrams used to specify the elements of the platform and the system behavior.

Each node is modeled by means of: Central Processing Unit (CPU), a radio unit, a serial communication channel, a memory, and an analog-digital converter a sensing module (ADC), among others.

Each module is represented by a class, which contains attributes, methods, and ports. Figure 4 shows the CPU class.

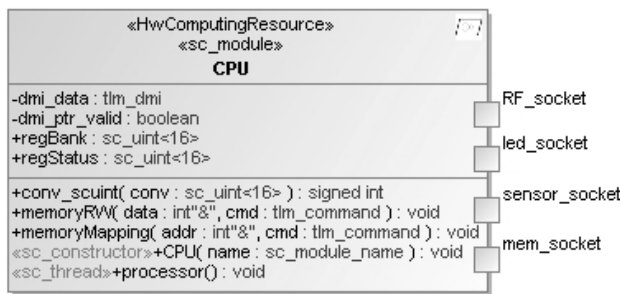


Figure 4. CPU Class.

This module was stereotyped as <<HwComputingResource>> using the MARTE profile and it was stereotyped as <<sc_module>> using the SystemC profile. Also, the module methods can be stereotyped with the SystemC profile, as simulation processes (sc_thread or sc_method). These processes are functions within the modules that are "registered" in the simulation kernel [19]. Furthermore, a method to be the class constructor (<<sc_constructor>>) can be stereotyped.

The connection among the internal modules of a node is performed by the composite structure diagram. Communication is done through ports, which either provide or require interfaces. Figure 5 shows the internal structure of the accelerometer node.

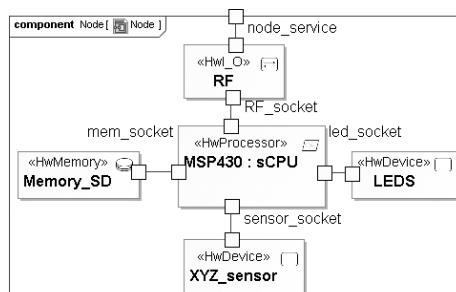


Figure 5. Node Composite Structure Diagram.

Therefore, Figure 6 shows an example of CPU model in SystemC language as a result of automatic code generation derived from UML diagrams. This figure describes the CPU module header file (CPU.h file). In the first lines, the modules that compose CPU module were indicated. Lines 10 to 12 declare module ports stereotyped with SystemC profile, which allow the data flow among modules. Lines 13 to 16 instances of the modules were declared. The functions were defined in lines 19 and 20, which can be public or private, and their functionality were modeled using state diagrams and activity diagrams as shown in Figure 8 and Figure 9. The constructor is declared between the lines 22 and 36 it carried out registration functions in the simulation kernel, and it also carried out the interconnections among modules. And on lines 38, 39, and 40 module private variables were declared.

```

1  #ifndef MSP430_H
2  #define MSP430_H
3  #include "systemc.h"
4  #include "ADC.h"
5  #include "Clock.h"
6  ...
7  class MSP430: public sc_module {
8  public:
9  //Module Port Declaration
10 tlm:tlm_b_initiator_socket< > RF_socket;
11 tlm:tlm_b_initiator_socket< > led_socket;
12 ...
13 //Submodule Declaration
14 ADC *adc1;
15 Clock *clock1;
16 ...
17 //Functions
18 public:
19 signed int conv_scuint(sc_uint<16> conv);
20 void memoryRW(int &data, tlm::tlm_command cmd);
21 ...
22 //Constructor
23 SC_HAS_PROCESS(CPU);
24 CPU(sc_module_name name):sc_module(name){
25   sc_constructor(CPU);
26   sc_thread(processor);
27   ...
28 //Submodule Implementation
29   adc1 = new ADC("adc1");
30   clock1 = new Clock("clock1");
31   ...
32 //Sub-Module Port Binding
33   adc1.ADC_init_socket(ADC_target_socket);
34   clock.time_init_socket(time_targer_socket);
35   ...
36 }
37 //Attributes
38 private:
39   tlm:tlm_dmi dmi_data;
40   sc_uint<16> regBank;
41   ...
42 };
43 #endif
  
```

Figure 6. CPU SystemC code generated.

Figure 7 shows the general nodes behavior. The behavioral model is developed using state diagrams. Model was applied to some restrictions on MARTE such as: <<TimeDomain>> which defines a time limit on states to implement its processes; <<GaEventTrace>>

which indicates signals are activated by external events; and <<ResourceUsage >> which specifies that state makes use of physical resources.

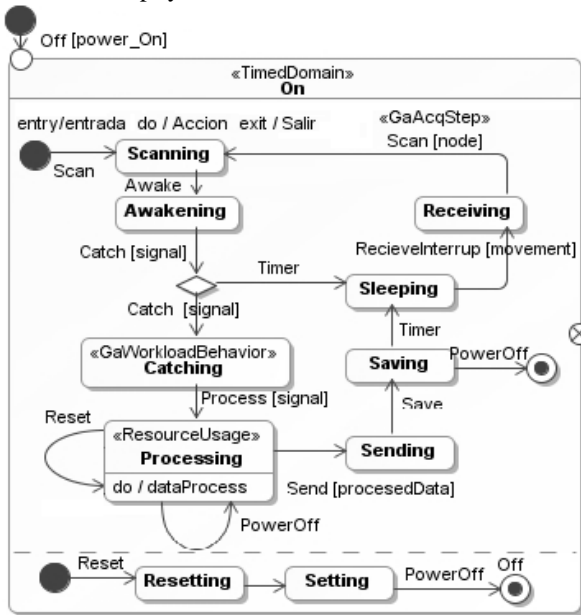


Figure 7. NodeBehavior - StateDiagram .

As mentioned earlier, the model of the behavior of an embedded system could be described by state machines and/or activity diagrams. This second diagram, allowing to easily manipulate data, is mainly used to specify in detail the functions within the states. Figure 8 shows an example of how it can be an activity diagram and Figure 9 shows the corresponding code translation.

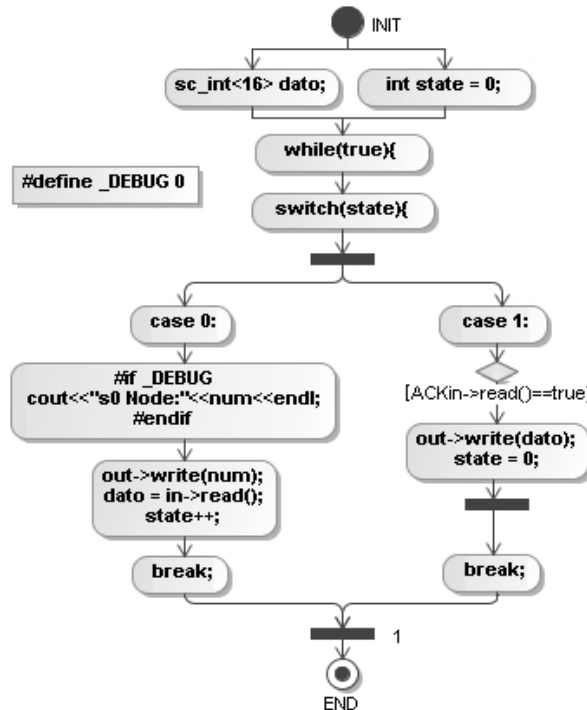


Figure 8. Internal State – Activity Diagram.

After completing the system models using UML, the transformation process suggests the generation of code, in order to get the virtual model of our sensor network. The generation begins by describing the code obtained from the hardware platform. This is achieved searching various structural diagrams (class and composite structure) and selecting the attributes, methods, connections between modules, among others.

Also, the code generation for system behavior is carried out by the state machines and activity diagrams. The figure shows the generation obtained from the activity diagram shown above (figure).

```

1 void CPU_NODE::internal (void) {
2   sc_int<16> dato;
3   int state = 0;
4   while(true){
5     switch(state){
6       case 0:
7         #if _DEBUG
8           cout<<"s0 Node:"<<num<<endl;
9         #endif
10        out->write(num);
11        dato = in->read();
12        state++;
13        break;
14       case 1:
15        if (ACKin->read() == true){
16          out->write(dato);
17          state = 0;
18        }
19        break;
20     }
21   }
22 }
    
```

Figure 9. SystemC behavior.

Finally, the final step of code generation tool is the implementation of the model in SystemC Figure. The model simulation is shown below. This can be visualized that node 1 captures the input signal and generates a request to the node sink. This, to be available, responds to the node can send the captured data. Node 1 sends the data and displays it sink. This process is repeated each time a signal is generated. To display these messages on screen, the model allows the designer to enable debug mode, facilitating the correction of the model.

```

1          SystemC 2.2.0
2 Node: 0 capturing.. value: 732mV (NODE)
3 Event node: 0 waiting... (SINK)
4 Node: 0 sending... (NODE)
5 Data: 732mV (SINK)
6 Node: 1 capturing.. value: 684mV (NODE)
7 Event node: 1 waiting... (SINK)
8 Node: 1 sending... (NODE)
9 Data: 684mV (SINK)
10 Node: 0 capturing.. value: 661mV (NODE)
11 Event node: 0 waiting... (SINK)
12 Node: 0 sending... (NODE)
13 Data: 661mV (SINK)
14 ...
    
```

Figure 10. CPU SystemC code generated.

5. CONCLUSIONS

In this paper, a design process for automatic code generation, to improve productivity and quality in system design was followed. It was necessary to merge the capabilities' UML to model systems with the power to simulate of SystemC language. In this process designs flexible designs due to its modularity, and the separation between platform model and application model was achieved. UML allows modeling through graphical notations and approach our models to a specific domain, due to the profiles that can handle this language, making language UML valuable for the design of embedded systems and real-time systems.

The code generation tool was developed so that he can accept hierarchical models of the system on both fronts: platform (Class diagram and composite structure) and behavior (activity diagram and state machines). This feature allows the designer obtain modular system, leading to the reuse of models and facilitating their maintenance.

Furthermore, due to the implementation of two diagrams for describing the behavior (state machine diagram and activity diagram), expanded the number and complexity of applications that can be modeled and generated.

As future work it has been pretended to study the restrictions that MARTE profile offers to perform modeling real time systems. Also, it is intend to continue improving the rules for both modeling and code generation, in order to ensure the right transformation to SystemC models.

6. ACKNOWLEDGMENT

Research supported by COLCIENCIAS, TIC Ministry at Colombia, and ARTICA (excellence research center) with the project "Design Methodology of Embedded Systems with High Reliability and Performance Focused on Developing of Critical Applications," Colombia.

7. REFERENCES

- [1] W. Wolf. "High-Performance Embedded Computing: Architectures, Applications and Methodologies". Morgan Kaufmann. 2007.
- [2] Alberto Sangiovanni-Vincentelli "Quo Vadis, SLD: Reasoning About the Trends and Challenges of System Level Design", Proceedings of the IEEE, Vol. 95 No. 3, March 2007.
- [3] A. Sangiovanni-Vincentelli. "Is a Unified Methodology for System-Level Design Possible?". IEEE Design and Test of Computers Special Issue on Design in the Late and Post-Silicon Era, 25(4):346-358, July 2008.
- [4] Per Andersson, Martin Höst. "UML and SystemC – A Comparison and Mapping Rules for Automatic Code Generation", Embedded Systems Specification and Design Languages, Chapter 14, Editorial Springer, 2007.
- [5] P. Andersson, M. Höst, M. Bergström. "UML to SystemC Transformation in the MARTES Project". Embedded Systems Specification and Design Languages – FDL'07, 2007.
- [6] E. Riccobene, P. Scandurra, A. Rosti, and S. Bocchio. "An HW-SW Co-design Environment based on UML and SystemC". 2008
- [7] Correa B.A., Eusse J.F., Múnera D., Vélez J F., Aedo J.E. "High Level System-on-Chip Design using UML and SystemC". Electronics, Robotics and Automotive Mechanics Conference (CERMA).
- [8] J. Vidal, F. de Lamotte, G. Gogniat, Philippe Soulard, Jean-Philippe Diguët: "A co-design approach for embedded system modeling and code generation with UML and MARTE". Design, Automation & Test in Europe Conference & Exhibition, 2009.
- [9] M. Mura, A. Panda, M. Prevostini. "Executable Models and Verification from MARTE and SysML: a Comparative Study of Code Generation Capabilities". Workshop, 2008.
- [10] L. Murillo, M. Mura, M. Prevostini. "Model-based Design Space Exploration for RTES with SysML and MARTE". Forum on Specification, Verification and Design Languages, 2008.
- [11] L. Murillo, M. Mura, M. Prevostini. "Bridging the Gap between Model Driven Engineering and HWSW Codesign". Master Thesis, 2009
- [12] M. Mura, M. Prevostini. "Code Generation from Statecharts: Simulation of Wireless Sensor Networks". Conference on Digital System Design Architectures, Methods and Tools, 2008.
- [13] Open source Tool for graphical UML2 Modelling www.papyrusuml.org
- [14] É. Piel, R. Ben Attitallah, P. Marquet, S. Meftali, S. Niar, A. Etien, J. Dekeyser, and P. Boulet. "Gaspard2: from MARTE to SystemC Simulation". DATE 08, Munich, Germany, March 2008.
- [15] "A UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded systems, Beta 2". OMG Adopted Specification. June 2008.
- [16] Visual UML modeling tool www.magicdraw.com
- [17] "MOF 2.0/XMI Mapping, Version 2.1.1". OMG Available Specification. December 2007
- [18] The Open SystemC Initiative (OSCI). www.systemc.org
- [19] D. Black, J. Donovan, B. Bunton, A. Keist. "SystemC: From the Ground Up". Springer Science+Business Media, LLC. 2010.

An Object-Oriented Social Networking to Link People with Similar Interests and Activities

Ching-Cheng Lee, Prachi Garg

Department of Computer Science, California State University at East Bay, Hayward, California, USA

Abstract - *Social Networking sites have become increasingly prevalent in the society as a medium to socialize and connect with friends. These websites have changed the way we communicate and share our thoughts and experiences with other people. However, most social networking websites lack in connecting people over common activities or interests. Users often befriend other users with the objective to swap profiles and increase their friend list. Overtime, these connections become meaningless since there is no common interest or activity to bind them. This research proposes an effective approach to find users' interests and suggest friends by implementing object-based features in a social networking application. Object centered sociality is suited for most social networking sites since it provides users a means to connect with other users who have similar level of interest.*

Keywords: MyBook, MVC, JSP, Apache Tomcat, J2EE

1 Introduction

Most social networking sites focus primarily on building friends list and are more useful for contacting and locating old friends or make new ones. They allow users to create profiles, swap messages, and share photos – all with the goal of expanding their circle of online friends. These websites are good at representing links between people, but it does not explain what connects those particular people and not others [1]. They provide little attraction for repeated visits and therefore after some point become boring. Connecting content and people would provide a meaning to social networking [2]. Associating social networking site members with others who have similar values and topical interests are more likely to be friends, and therefore lead to an effective communication. Developing an Object-Centered sociality can facilitate focused interactions among online communications.

1.1 Object-Oriented Sociality

Social networking sites provide features to find people with similar interests. However, they only use interests that are already specified by users in their profile and therefore are very static in nature. Often times users search for people on social networking sites by specifying their interests for example, cooking, in the search bar. The result shows a list of people who have mentioned cooking either in their interest

section or somewhere in their profile name. But there is no information about their (users) level of interest in cooking. Object-centered sociality on the web emphasizes on linking people dynamically through shared objects of interests like jobs, workplaces, sports and hobbies. This technique involves analyzing content posted by a user on his/her homepage. Content analysis will provide the system with information about user's interest level in various fields. When users post blogs, upload files or pictures they can annotate them via tags. Similar annotations are then searched through the system that determines the interests of other users. The result is further analyzed to determine the interest level of those users in that particular subject and create a compatibility meter. A friend list is then prepared and suggested to the user.

2 Background

The possibility to publish and gather personal information has been a major factor in the success of the Web from the beginning. Remarkably it was only in the year 2003 that the Web has become an active space of socialization for the majority of users [3]. That year has seen the rapid emergence of a new breed of Web sites, collectively referred to as social networking services (SNS). The first-mover Friendster attracted over 5 million registered users in the span of a few months [4] which was followed by Google and Microsoft starting or announcing similar services. Many popular Web applications are now exploiting user-driven information networks built by means of social annotations[5][6]. Social annotations are freely established associations between Web resources and metadata (keywords, categories, ratings) performed by a community of Web users with little or no central coordination. A mechanism of this kind, which has swiftly become well established is that of collaborative tagging[7][8], whereby Web users associate free-form keywords – called “tags” – with on-line content such as Web pages, digital photographs, bibliographic references and other media. The product of the users' tagging activity is an open-ended information network – commonly referred to as “folksonomy” – which can be used for navigation and recommendation of content, and has been the object of many recent investigations across different disciplines[9][10][11].

Most Social Networking Sites focus on helping users create as many friend connections as possible, but provide limited

means of sustaining connections or keeping the friends interested in each other. Users of socializing sites typically get friend invitations also from authors they don't necessarily know, or know very little about. These often include bands, communities or people wanting to network because of various reasons. All this is an effective way to encourage people to socialize more and to get connections, but the profile data in the sites needs to be discovered either manually or by proactive recommendation systems discussed e.g. in [12][13]. Several social networking, social bookmarking and blogging websites are being actively researched. While these SNSs allow searching for people with similar interest and make a friend connection, they do not provide any mechanism to determine a user's interest level in a particular activity or subject. All past research acknowledge the challenge in connecting people on web and increasing their network of friends, and sought similar solutions by suggesting them mutual friends.

2.1 The Future of Social Networks

The future of social networks on the Internet [2] suggests semantic web for linking and performing operations on object-related data gathered from different social networking websites. Friend-of-a-Friend8 project lets people share and use information about their activities and transfer information between websites. It describes people, the links between them and the things they create and do. Leveraging Social Networking Services to Encourage Interaction in Public Spaces[6] propose a system that utilizes the user generated content on socializing websites and displaying flashes of uploaded digital content on a large public display situated in the public or semi-public space like an airport or café. They do not, however, propose any methodology for making friends with similar interest level in an activity as that of the user's. The success of any SNS is directly proportional to how they can keep their users interested. [15] suggests that tags that are used to organize resources or find related resources can now potentially serve as objects around which the users can form tight connections. Research in recent years have proposed developing an object-oriented sociality on the web that links people based on shared objects of interests like jobs, sports, hobbies etc., and therefore keep the users interested.

3 Architecture

3.1 System Architecture Overview

The system consists of the following components: User Management, Blog Management, Picture Management, Interests Management, Friend management, Friend Suggestion.

User Management – Facilitates adding users to the system authenticating users.

Blog Management – Facilitates a user to retrieve, upload or modify blogs in the system. To add new blog, user clicks on the 'Add New Blog' link, adds the desired text and a title for the blog. The blog is then stored in the system. The user can later modify Blogs.

Picture Management – Facilitates a user to retrieve, upload or modify pictures in the system. To upload picture, user clicks on 'Upload Picture link. Picture can be uploaded from the desired location on the computer. A copy of the picture is then stored in the system.

Interests Management - Retrieves all interest in the system and display them in a tag cloud based on its usage. Searching an interest would display all blogs, files pertaining to that interest. It will also display interested users in a cloud form. A larger font size would mean more results were found for that user.

Friend Management – Facilitates a user to add a friend.

Friend Suggestion - The system matches a particular user's interests with other users in the system and suggests friends to user. Users are displayed in cloud form. A larger font size would mean more similarity between the user and the suggested friend..

3.2 Friend Suggestion Algorithm

The implementation of the friend suggestion algorithm is a key component of this research. The following describes the major steps of the algorithm for building the Friend Suggestion List.

- Step 1. *GetListOfUsersFromDB*
Get all the Users from the database that has the same tag names as the current user.
- Step 2. *BuildTagInfoFromDB*
For a list of users that have the same tag names as the current user, extract all the tag names and tag count from the database and build the List<TagInfo> data structure.
List<TagInfo>:=
List<TagCount,TagName>
- Step 3. *ComputeUserMatrix*
Compute the User Matrix from the List<TagInfo>
- Step 4. *NormalizeUserMatrix*
Compute the Normalized Matrix from the User Matrix
- Step 5. *ComputeUserSimilarity*
Compute User Similarity from NormalizedUserMatrix
- Step 6. *BuildFriendSuggestionList*
Build Friend Suggestion List from User Similarity in the previous step: List<FriendSuggester>:=
List<userName, Similarity, Rank>
Sort FriendSuggester by user Name.

4 Implementation

The implementation of our algorithms together with the presentation layer is called *MyBook*. A Spring framework based on MVC architecture is used to build this application. The following are the main pages provided by the presentation layer of the application:

Login Page: This page allows the users to enter Username and Password.

Home Page: This page shows the users' friends, blogs, files and popular tags. It also lets the users navigate to other pages like – AddBlog, UploadPicture, etc.

All the above modules are designed and developed using Java, JSP, Apache Tomcat, Spring Framework, and other open source technologies in the J2EE realm.

5 Experiments

Two experiments were carried out to illustrate the effectiveness of our design and implementation as follows.

5.1 Experiment 1

As shown in Figure 1, the user logs in and enters blogs and tags them. The system looks into the user's interests and suggests potential friends. As soon as the user enters more blogs and tags, the system updates the friend suggestion list.

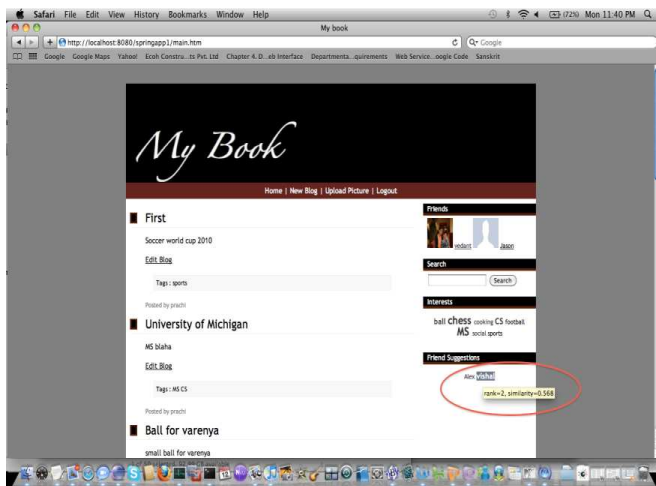


Figure 1 – Friend Suggestion

5.2 Experiment 2

Following are the steps for step Experiment2. In this experiment, we create N users and generate M Tags. For each user, we create blogs with tags. Then we run the Friend Suggestion Algorithm.

1. Create N Users for the MyBook (u1..uN)
2. Generate M tags (t1..tM)
3. For each user, create $[M/2 + \text{Random}(0, M/2)]$ blogs with

Each blog a tag in $[\text{Random}(1, M)]$

4. For each of the user, run the Friend Suggestion Algorithm and calculate the average response time in milliseconds.

In the following, Table 1 shows the time taken in generating the friend suggestion list in different scenarios such as the number of Users and number of Tags.

Table 1 – Time in Generating the Friend Suggestion List

Users	Tags	Generate the Friend Suggestion List Average Time (ms)
100	10	37.39
500	10	115.59
1000	10	233.81
100	50	103.34
500	50	506.76
1000	50	1001.19
100	100	185.37
500	100	954.41
1000	100	1934.12

6 Conclusion

The system proved 100% efficient in finding a user's interest and suggesting friends based on his/her interest during each browser session. It also updates the friend suggestion list as soon as any new blog or file is entered into the system. A manual search for friends with similar interests is not required. Friendships established by manual search may or may not lead to meaningful connection since the interest level of users in an activity cannot be determined beforehand. The system is efficient in automatically suggesting friends to a user based on his/her interest. In comparison to *MyBook*, most social networking sites suggest friends based on friend of a friend system. Facebook, MySpace and Orkut suggest mutual friendship. They also allow searching for people based on their interest but the results shown do not tell the interest level in that particular activity. It is therefore, not a viable solution to the issue addressed in this research. Delicious provides tagging and searching for data under the same tag name. It gives usernames who created that tag, but does not have profile information. It cannot be used as a utility to connect with people and socialize.

7 Future Work

As an extension to this work, we propose the scheme of auto-generation of Tags/Objects. The current implementation relies on user-entered tags to identify interest. In fact, Keyword search can be performed on the user-generated content (blogs, files etc) to auto-generate tags/objects. That can then be fed into the friend suggestion algorithm. Keyword search engine can either be run periodically on the user-generated content (blogs, files etc) to generate tags or can be performed immediately at the time the user-content gets saved. Implementation will require use of stand-alone text search engines like Lucene or full-text search operations provided by database engine (MySQL).

8 References.

- [1] K. Knorr Cetina "Sociality with Objects: Social Relations in Postsocial Knowledge Societies", *Theory Culture and Society*, 1997, Vol.14
- [2] John Breslin, Stefan Decker, "The Future of Social Networks on the Internet: The Need for Semantics", *IEEE Internet Computing*, November 2007, pp. 86-90
- [3] P. Mika. "Flink: Semantic web technology for the extraction and analysis of social networks," *Journal of Web Semantics*, vol. 3, pp. 211-223, October 2005.
- [4] L. Kahney, "Making friendster in high places," *The Wired News*, July 2003.
- [5] X. Wu, L. Zhang, Y. Yu "Exploring social annotations for the semantic web" *ACM*, 2006, pp.417-426.
- [6] T. Hammond, T. Hannay, B. Lund, J. Scott "Social Bookmarking Tools (I): A General Review", *D-Lib Magazine* 11, 2005.
- [7] A., Mathes "Folksonomies – Cooperative Classification and Communication Through Shared Metadata" 2004.
- [8] S. Golder, B. A. Huberman "The Structure of Collaborative Tagging Systems", *Journal of Information Science*, 2005, 198-208.
- [9] C. Marlow, M. Naaman, D. Boyd, M. Davis, HT06 tagging paper, taxonomy, Flickr, academic article, to read *ACM*, pp. 31-40.
- [10] C. Cattuto, V. Loreto, L. Pietronero "Semiotic Dynamics and Collaborative Tagging", *Proc. Natl. Acad. Sci. USA*, 2007, 1461-1464.
- [11] Ciro Cattuto, Alain Barrat, Andrea Baldassarri, Gregory Schehr, Vittorio Loreto, "Collective dynamics of social annotation", hal-00361199, version 2 - 30 Apr 2009.
- [12] J., Donath "Identity and deception in the virtual community", Kollock P and Smith M (Eds): *Communities in Cyberspace*, Routledge, London, 1998.
- [13] Simo Hosio, Hannu Kukka, Jukka Riekkii, "Leveraging Social Networking Services to Encourage Interaction in Public Spaces", *ACM*, December 2008.
- [14] Friend of a Friend Project <http://www.foaf-project.org/>
- [15] Gene Smith, "Tagging: People-Powered Metadata for the Social Web" Publisher: New Riders, December 2007.

Comparison of software development productivity based on object-oriented programming languages

Cuahtémoc López-Martín¹, Arturo Chavoya², and María Elena Meda-Campaña³

^{1,2,3}Information Systems Department, CUCEA, Guadalajara University, Jalisco, Mexico

¹cuahtemoc@cucea.udg.mx, ²achavoya@cucea.udg.mx, ³emeda@cucea.udg.mx

Abstract - *The reasons for measuring software productivity are to identify how to reduce software development costs, improve software quality, and improve the rate at which software is developed. In this paper, a data set of 572 software individual projects developed from 2005 to 2010 with practices based on a process specifically designed to laboratory learning environments (Personal Software Process) is used to know if there is any statistically significance difference between the productivity of developers whose projects were written using the object oriented programming languages C++ and Java. Results suggest that there is difference between projects developed in these two programming languages when software projects have been developed in a disciplined way in a laboratory learning environment.*

Keywords: Software development productivity, Object oriented programming languages, Empirical software engineering.

1 Introduction

The abstraction describes on which level the measurements software projects are carried out; there exist the following five abstraction levels [8]: organization, process, project, individual and task. This study is related to the individual level once software projects were individually developed by practitioners.

There are at least the following four options to collect as well as to report software production data [16]: developer self report, project or team manager, outside analysts or observers, and automated performance monitors. This study was related to the first option. In order to reduce bias, each developer used the same personal practices based upon Personal Software Process (PSP). The PSP was selected because the levels of software engineering education and training could be classified in the small as well as in the large software projects [1]. In the case of small software projects, the PSP whose practices and methods are used for delivering quality products on predictable schedules can be useful [6]. Moreover, it has

been suggested that a higher productivity over the entire system life cycle use to be associated with the use of a disciplined programming methodology [2]; even, productivity increases when extensive use of modern programming practices are applied (as top-down design, modular design, design reviews, code inspections, and quality-assurance programs) [17].

In accordance with the use of PSP for gathering data, some previous researches have approached their efforts to PSP automate [9] [14] and yet others have incorporated PSP concepts into their programming courses [11].

There have been diverse measures of productivity ([5] [16] [18]), in them have been indicated that the measure of productivity most commonly used is that of size over effort productivity = size / effort. That is the one used in this study; the size is measured using number of lines of code developed by unit of effort.

There have been two main directions on the study of productivity in software engineering literature [18]: (1) researches have been focused on the measure or estimation of productivity, and (2) emphasis has been laid on the discovery of methods or significant factors for productivity improvement. The approach of this study is related to second direction.

Because of the type of programming language is one of the two main factors found having significantly influence on the productivity [7], the sample of this study integrates those projects coded in C++ or Java. The hypotheses of this research are the following:

H₁: There is a statistically significant difference in the development productivity between the projects coded in C++ and those coded in Java when the projects are developed in a disciplined way in a laboratory learning environment.

H₂: There is not a statistically significant difference in the development productivity between the projects coded in C++ and those coded in Java when the projects are developed in a disciplined way in a laboratory learning environment.

1.1 Software measurement

Measures of source code size can be classified in physical source lines and in logical source lines [13]. The count of physical lines gives the size in terms of the physical length of the code as it appears when printed. Lines of code have been used by previous researches focused on productivity analysis of large projects [3] [4] [10] [16] [17] and even more recently when productivity has been related to individual projects [15].

In this study, the independent variable in the prediction models is New and Changed (N&C) code and it is considered as physical lines of code (LOC). N&C is composed of added and modified code [6]. The added code is the LOC written during the current programming process, while the modified code is the LOC changed in the base software project when modifying a previously developed project. The base project is the total LOC of the previous project while the reused code is the LOC of previously developed projects that are used without any modification.

A coding standard should establish a consistent set of coding practices that is used as a criterion when judging the quality of the produced code [6]. Hence, it is necessary to always use the same coding and counting standards. The software projects developed of this study followed such guidelines.

2 Experimental design

Because measuring software productivity presupposes an ability to construct a measurement project comparable to those employed in experimental designs for behavioral studies, it is necessary to insure that the measures employed are reliable, valid, accurate, and repeatable. It means that to measure software production implies understanding of the relationship between measurement and instrumentation employed to collect and measure data [16]. Hence, in this paper data collected were related to the same instruments (logs), phases, and standards suggested by PSP.

The experiment was done inside a controlled environment having the following characteristics:

1. All of the developers were experienced, working on software development inside their enterprises at which they were working; however, no one of them had received a course related to personal practices for developing software at individual level.

2. All developers were studying a postgraduate program related to computer science.

3. Each developer wrote seven project assignments. However, only four of them were selected from each developer. The first three projects were not considered because they had differences in their process phases and logs, whereas in latest four projects phases are the same: plan, design, design review, code, code review, compile, testing and post-mortem, and they are based upon the same logs.

4. Each developer selected his/her own programming language whose code standard had the following

characteristics: each compiler directive, variable declaration, constant definition, delimiter, assign sentence, as well as flow control statement was written in a line of code.

5. Developers had already received at least one formal course about the object oriented programming language of their choice and they had good programming experience in that language. The sample of this study reduced the bias because it only involved developers whose projects were coded in C++ or Java. One reason for selecting these kinds of languages is because object-oriented languages facilitate high productivity [16].

6. As this study was an experiment with the aim to reduce bias, we did not inform developers about our experimental goal.

7. Developers filled out an Excel sheet for each task and submitted it electronically for examination.

8. Each PSP course had a group of fifteen developers or less.

9. All of developers coincided with the counting standard depicted in Table 1.

10. Developers were constantly supervised and advised about the process.

11. The code written in each project was designed so to be reused in subsequent projects.

12. The developed projects had complexity similar to those suggested in the original PSP [6]. From a set of 18 individual projects, a subset of seven was randomly assigned to each of all developers. A brief description by project is the following:

- Estimating the mean of a sample of n real numbers.
- Estimating the standard deviation of a sample of n real numbers.
- Matrix addition integrated by real numbers.
- Summing the diagonal of a real numbers square matrix.
- Translating from a quantity to letters.
- Calculating the correlation (r) between two series of real numbers.
- Computing the linear regression equation parameters a and b ($y=a+bX$).
- Calculating z -values from a sample of real numbers.
- Calculating the size of a sample.
- Calculating the y -values from a sample of real numbers using the normal distribution equation.
- Calculating the estimation standard error (from $y=a+bX$).
- Calculating the coefficient of determination (r^2) from a linear regression equation.
- Calculating both upper and lower limits from a sample of real numbers based upon its standard deviation and mean
- Calculating the coefficient of variation from a distribution.
- Estimating the values based upon statistical empirical rule.

- Counting the physical lines of code of a software project omitting comments and blank lines.
- Both storing and searching records from a file.
- Both deleting and modifying records from a file.
- Data used in this study belong from those developers, whose data for all seven exercises were correct, complete, and consistent.

Table 1. Counting standard

1) Count type Physical/logical	Type Physical
2) Statement type	Included
a) Executable	Yes
b) No executable	
Declarations	Yes, one by text line
Compiler directives	Yes, one by text line
Comments and Blank lines	No
3) Clarifications { and }	Yes

3 Data analysis

Data from 572 individual software projects developed by 143 practitioners between the years 2005 to 2010 were used to be compared in this study (Appendix A). Once the sample of 572 software projects was developed with C++ (288 projects) and Java (284 projects), we analyzed if there was any statistical difference in their productivity values. Table 2 shows that since the p-value of the F-test is less than 0.05, there is a statistically significant difference between the productivity of the two languages at the 95.0% confidence level. This difference result can graphically be observed in the means plot of Figure 1, which shows that those projects coded in Java had a better productivity that those coded in C++ with (28 versus 25 N&C lines of code by hour).

Table 2. ANOVA for productivity by programming language

Source	Sum of squares	Degrees of freedom	Mean square	F-ratio	P-value
Between groups	1689.47	1	1689.4	8.15	0.0045
Within groups	118136.	570	207.25		
Total	119825.	571			

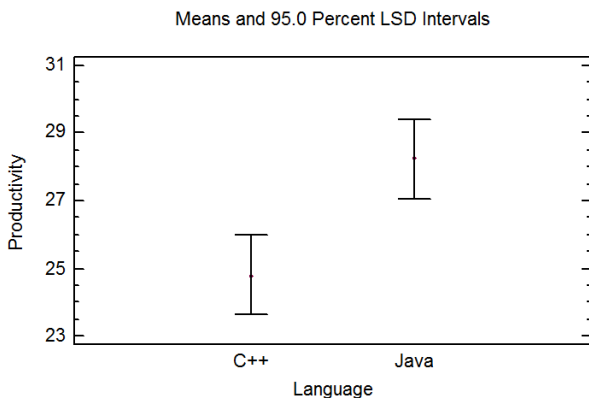


Figure 1. Plot of means for programming languages

The validity of an ANOVA is based on the analysis of the following three assumptions of residuals [12]:

1) Independent samples: Each project was developed independently and by a single practitioner, so the data are independent.

2) Equal standard deviations: In a plot of this kind the residuals should fall roughly in a horizontal band centered and symmetric about the horizontal axis (as shown in Figure 2), and

3) Normal populations: A normal probability plot of the residuals should be roughly linear (as shown in Figure 3).

Hence, the three assumptions for residuals in the productivity data set were considered as met.

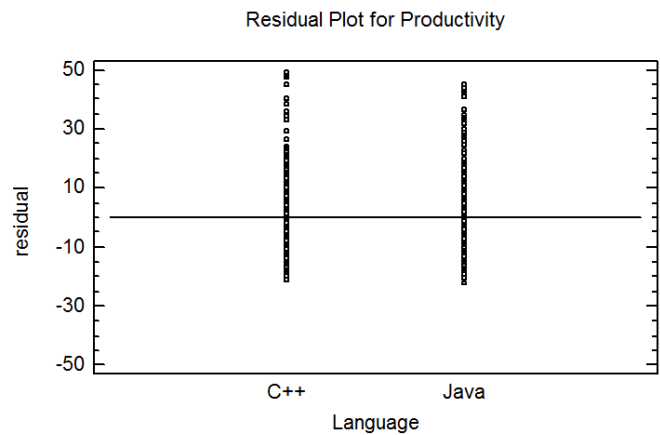


Figure 2. Equal standard deviation plot from programming languages

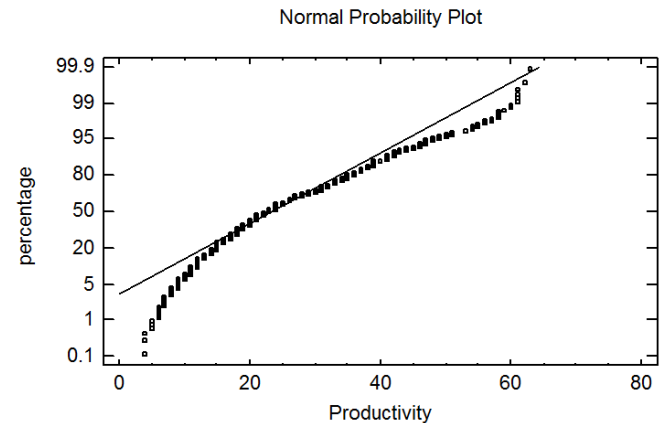


Figure 3. Normality plot from programming languages

4 Conclusions

Owning that there are relevant reasons for measuring software productivity and based upon the assumption that the programming language has influence in the software

development productivity, this study compared software projects code in two object-oriented programming languages. The software projects were developed following a disciplined process in a controlled environment.

After an statistical analysis based upon ANOVA, the accepted hypothesis is the following:

H_1 : There is a statistically significant difference in the development productivity between the projects coded in C++ and those coded in Java when the projects are individually developed in a disciplined way in a laboratory learning environment.

After ANOVA, a plot of means showed that projects coded in Java showed a better productivity than those coded in C++. This result was validated based on the three assumptions of residuals.

Future research is related to comparison between these two programming languages when they are used in industrial software projects.

5 References

- [1] Bagert D. J., Hilburn T. B., Hislop G., Lutz M., McCracken M., Mengel S. "Guidelines for Software Engineering Education". CMU/SEI-99-TR-032, ESC-TR-99-002, Software Engineering Institute, Carnegie Mellon University. 1999
- [2] Bailey, J., Basili, V. "A Meta-Model for Software Development Resource Expenditures". 5th. Intern. Conf. Soft. Engr., IEEE Computer Society, pp. 107-116. 1981
- [3] Boehm B., Abts Ch., Brown A.W., Chulani S., Clarck B. K., Horowitz E., Madachy R., Reifer D. & Steece B. "COCOMO II", Prentice Hall. 2000
- [4] Cusumano M., Kemerer, C.F. "A Quantitative Analysis of U.S. and Japanese Practice and Performance in Software Development", Management Science. Pages 1384-1406. 1990
- [5] Fenton N.E. & Pfleeger S. L. "Software Metrics: A Rigorous and Practical Approach". PWS Publishing Company. 1997
- [6] Humphrey W. "A Discipline for Software Engineering". Addison Wesley. 1995
- [7] Jiang, Z., Comstock C. "The Factors Significant to Software Development Productivity". World Academy of Science, Engineering and Technology. Pages 160 – 164. 2007.
- [8] Kai, P., 2011. Measuring and predicting software productivity: A systematic map and review. Information and Software Technology, Elsevier, Volume 53, Issue 4, pages 317-343
- [9] Johnson, P.M, Kou, H., Agustin, J., Chan, Ch., Moore, C., Miglani, J., Zhen, S., & Doane, E.J. "Beyond the Personal Software Process: metrics collection and analysis for the differently disciplined". Conference on Software engineering education and training: process and methodology. Pages 641 – 646. 2003.
- [10] Lawrence, M.J. "Programming Methodology, Organizational Environment, and Programming Productivity", Journal of Systems and Software, Elsevier. Vol. 2, Pages 257-269. 1981
- [11] Maletic J.I., Howald, A., Marcus A. "Incorporating PSP into a traditional software engineering course: an experience report". International Conference of Software Engineering Education and Training. Pp. 89 – 97. 2001.
- [12] Montgomery D. C.. "Design and Analysis of Experiments". John Wiley, 2009.
- [13] Park, R.E. "Software Size Measurement: A Framework for Counting Source Statements". Software Engineering Institute, Carnegie Mellon University. CMU/SEI-92-TR-020. 1992.
- [14] Postema, M. Dick, M., Miller, J., Cuce, S. "Tool Support for Teaching the Personal Software Process". Computer Science Education, Vol. 10, No. 2, Pages 179-193. 2000.
- [15] Rombach D., Münch J., Ocampo A., Humphrey W.S., Burton D. "Teaching disciplined software development". Journal Systems and Software, Elsevier, pp. 747- 763.. 2008.
- [16] Scacchi, W. "Understanding Software Productivity". International Journal of Software Engineering and Knowledge Engineering. Revised and reprinted in Advances in Software Engineering and Knowledge Engineering, D. Hurley (ed.), Pages 37-70. 1995.
- [17] Vosburg, J., Curtis, B., Wolverton, R., Albert, B., Malec, H., Hoben S., Liu, Y. "Productivity Factors and Programming Environments", Proc. 7th. Intern. Conf. Soft. Engr., IEEE Computer Society, Pages 143-152. 1984
- [18] Zhizhong, J., Naudé, P., Comstock, C. An Investigation on the Variation of Software Development Productivity, International Journal of Computer and Information Science and Engineering, Pages 72-81. 2007.

SESSION

SOFTWARE ARCHITECTURE, DESIGN PATTERNS + PETRI NET

Chair(s)

TBA

Enterprise Architecture and Organizational Transformation: The Human Side of Information Technology and the Theory of Structuration

Dominic M. Mezzanotte, Sr. and Josh Dehlinger
 Department of Computer and Information Sciences
 Towson University
 {dmezzanotte, jdehlinger}@towson.edu

Abstract

Enterprise architecture (EA) transforms the structure, culture, and social environment of an enterprise by introducing new processes and technologies into the workplace altering the roles, duties, responsibilities and organizational position of stakeholders. This transformation frequently affects stakeholder behavior and leads to either acceptance or rejection of an EA. Existing EA frameworks fail to recognize the significance of human behavior and its effect on EA. Therefore, a more holistic approach that includes sociologically-oriented provisions is needed. This paper advances our earlier work by exploring three factors that influence EA: organizational transformation, stakeholder resistance to change, and elicitation/use of erroneous EA requirements. Each of these issues can be addressed by implementing a sociologically-oriented approach designed to remove barriers that limit stakeholder action. The proposed approach focuses on a socio-communicative process that establishes an environment where stakeholder involvement in the decision-making aspects of EA is central.

Keywords: Enterprise architecture, organizational change, stakeholder behavior.

1. Introduction

Enterprise architecture (EA) embodies the business objectives, processes and technology infrastructure reflecting the desired incorporation and standardization requirements of an enterprise's operating environment [9]. In today's business climate, an EA represents a continuously evolving architecture aimed at improving operational efficiency and effectiveness. In the development of an EA, an EA Plan (EAP), documents the requirements that drive EA focusing on the architectural design, alignment, implementation, and deployment of new and/or enhanced technology [13][20]. EA requirements are predicated on gathering, analyzing, and validating explicit and tacit organizational knowledge. Thus, the EA focuses on

the information technology (IT) processes, related artifacts, platforms, software applications, and business and strategy to support and accomplish IT operations [12][20][28].

However, the implementation of an EA poses several potential problems requiring organizational management and an Enterprise Information Architect (EIA) to address and resolve: EA redefines the way and manner in which an enterprise functions [22] and, therefore, affects either positively or negatively stakeholder behavior.

First, EA changes the enterprise's structure, characteristics, culture and political climate of the workplace [1][22]. As a result of this transformation process, two outcomes for the EA are possible:

- It can be accepted as the new norm for the enterprise, in which case the enterprise simply moves on.
- It can have a negative effect on stakeholders and be rejected and/or modified to meet their personal goals and objectives. In this situation, the behavior of all involved in the process may be altered, and in some cases, it may literally tear the enterprise apart influencing the potential life of the enterprise by introducing factors into business operations that management may or may not be able to cope [14][15].

Second, the impact of these outcomes can produce behavioral patterns that can jeopardize the viability of the EA ending with the EA being improperly aligned with the enterprise's strategic business plan and operating model to either being partially implemented or completely abandoned [22]. The question then becomes: why didn't the changes brought about by EA work?

Answering this question is difficult as causal factors differ. For example, in our previous work [19], we identified several causes for failed EA, such as insufficient top management support, and which states that between 60% and 84% of all EA projects fail in one manner or another. This paper adds to that list by including other sources and statistics which cite causal factors such as [11]:

- Sixty-six percent of project failures are attributed to poorly defined applications (i.e.,

miscommunication between stakeholders and IT technical staff).

- Sixty to eighty percent fail because of poor requirements gathering, analysis, and management.

An analysis of these failures found that [11]:

- Fifty percent of the projects had to be rolled back out of production.
- Forty percent of the problems were identified by end-users.
- Twenty-five to forty percent of project cost was wasted on re-work.
- Up to eighty percent of budgets were consumed on fixing self-inflicted problems.

This latter group of statistics is supported by other literature attributing failure directly to erroneous requirements (i.e., organizational knowledge) [5][6][7][25], which we will collectively label hereafter as simply “poor architecture.” However, we look at the problem and resolution from a different point of view: human behavior and the impact technology has on that behavior. From this position, EA can be defined as being influenced by two separate and distinct perspectives which we will categorize as: 1) the context or environment in which EA functions; and, 2) the processes it symbolizes.

EA context is made up of sociological, organizational, and psychological elements such as stakeholder attitudes and behavior, and organizational norms, policies, politics, standards, and resources. EA frameworks (EAF), on the other hand, take on the more techno-centric aspects of EA design that consist of methods for developing the EA [5][20]. Yet, the interactions between the context of EA and its process are dependent. However, if not properly managed, they can pose a conflict in EA design in deciding the weight that should be apportioned to either context or process, potentially jeopardizing the success of the EA.

EA focuses on engineering principles and practices as a means to synchronize organizational activities and engineering modeling schemes to develop and test the architecture. The reality here lies in the fact that stakeholder requirements drive EA design [5][20][26][28]. However, the EAFs used to do the work are formulated around highly techno-centric processes and procedures based on the modalities of traditional computer oriented and computer science theories [10][12][24]. Existing EAFs aim at solving business problems from a purely technical perspective and do not include stakeholder behavior as a significant influence on EA design.

Given this perspective, each enterprise has its own characteristics, culture, and social structure which the enterprise information architect (EIA) must understand and include in the development of an EA.

For example, stakeholders are expected to adapt to new environmental conditions imposed by the EA. This influences stakeholder behavior with the assignment of new roles, duties, and responsibilities which, in some cases, they are expected to assimilate unquestionably [2][3][22]. This works well in enterprises that routinely function in a tightly controlled environment, it will not in others. [8][14][15]. In most cases however, stakeholders typically perceive these changes as a diminution of influence within the enterprise.

Therefore, the lack of continuity between EA context and process has the potential to cause conflicting views of the EA, altering the previously known stable state most enterprises and stakeholders strive for in the workplace. From this, the integration of sociologically-oriented principle with the existing techno-centric EAFs becomes a viable solution to EA design. Failure to implement such an approach leads to negative stakeholder behavior which may be observed in one of two ways [2][8]:

- They may resist the EA either overtly or covertly by exhibiting their reluctance to follow new norms, rules, and policies established by the enterprise.
- They may intentionally or unintentionally miscommunicate, mislead, and/or provide erroneous requirements as input to and thus sabotage the EA.

In either case, the EA may be jeopardized such that the enterprise reverts to the previous architecture. Reversion to the previous state is detrimental in that more efficient and effective technology is subverted and therefore enterprise growth is inhibited.

In earlier work, [17][18][19], we described several of the causal factors leading to EA failure. In this paper, we address two additional factors leading to failure: stakeholder resistance to change and “poor architecture.” These issues can be directly tied to the organizational transformation that takes place as a result of EA and the new technology it introduces. This work raises the level and significance of the impact of human behavior as a major input to EA and how that behavior is affected by technology. We examine Giddens’ *Theory of Structuration* and its application to this process [8][22].

The remainder of this paper is organized as follows. Section 2 assesses the activities used in existing EAF methodologies, their approach to and contribution to EA design, and their relationship to stakeholder behavior. Section 3 examines EA, organizational theory, human behavior, and the *Theory of Structuration* and their relationship to EA. Section 4 discusses the ramifications of not properly coordinating and controlling EA effort and concludes with some remarks.

2. Enterprise Architecture Frameworks and Stakeholder Behavior

The adoption and use of EA in large enterprises and development of complex, large-scale systems now places it at the forefront of IT and organizational business strategy [5][12]. The process aspect of EA is inclusive in its approach formulating a design and implementation plan for the project. The methodologies used in existing EAFs to describe the EA typically propose ontology for both viewing and analyzing the enterprise's current information architecture and operating environment. The focus of this effort is on how best to use technology and align it with the enterprise's strategic business plan and operating model [20].

The responsibility for developing the EA design has been that of an EIA. Until now, the requisite skill set for the EIA typically consists of a practical knowledge of technology and business practices [20]. Today, possessing just these skills alone are not enough to address a workplace environment where stakeholders no longer accept change without question but more routinely question the need for EA and the organizational transformation it brings about.

Of the many EAFs used since EA's inception, four stand out as de facto standards within the industry: the Zachman Enterprise Architecture Framework (Z|FA) [30], The Open Group Architecture Framework (TOGAF) [21], the Federal Enterprise Architecture Framework (FEAF) [4], and the Department of Defense Architecture Framework (DoDAF) [1]. Each of these EAFs feature distinct and unique approaches to EA design embodied with their own set of processes and procedures. In some cases, taxonomy defines the EAF (e.g., Z|AF) process while in the others listed above ontology describes the process [27]. Regardless of their methodologies, the strengths of the frameworks are that each imposes a disciplined regimen of processes and procedures guiding documentation of the EAP [19]. However, the weaknesses of current EAFs are that they fail to take into account [2][8][14][15]:

- The cultural effects on human behavior caused by the environment of the enterprise and the cognitive aspects of stakeholder behavior.
- The behavior and influence an individual has singularly or on and within a group.
- The social change within the enterprise resulting from the EA.
- The changes to the enterprise's political and economic systems caused by the EA.
- The social conflict that might result from the EA.

Each of these forces plays a significant role in stakeholder behavior and thus influences their capacity to contribute to the EA. For example, if we examine the cultural environment of an enterprise, stakeholder behavior mirrors organizational behavior learned over time and manifests itself based on their past and present work experiences within the enterprise [8][14]. Continuing, group behavior cannot be understood solely as the aggregate behavior of an individual though an individual may significantly influence the group's behavior or be influenced by the group [2][15]. Social change, on the other hand, is evolutionary and can occur in two opposing fashions: opened or constrained and is internally the by-product of technology and/or change in the political structure of the enterprise. External forces can also induce social change but are beyond the scope of this paper.

In an EA, the forces that affect social change are twofold: technology including the new processes introduced by the technology, and the new roles, duties, and responsibilities assigned to stakeholders [22]. These forces alone usually result in the transformation of the enterprise's political and economic structure. This force singularly can produce one salient, potent and counter-productive possibility in that it can evoke conflict within the enterprise [8][14][15].

As can be seen, EA alters stakeholder perceptions of the enterprise and as such changes their behavior which can seriously jeopardize the EA. Given this perspective, we can conclude that the techno-centric methodologies espoused by existing EAFs are deficient in neither providing mechanisms that recognize human behavior in their approach to EA nor contain any tools or processes that would mitigate adverse behavior in EA implementation [17][18][22]. Thus, the aggregation of these forces on EA elicits behavior that constricts, discourages, and limits stakeholder action and, at the same time, stifles their capacity to offer more innovative and creative approaches to problem-solving [17].

Though comprehensive from a technical point-of-view, the EAFs fail to provide for the kinds of humanistic based principles and practices such as communication, education, and training programs we believe essential to deal with stakeholder relationships, interactions, and behavior.

3. Enterprise Architecture and Human Behavior

Top-management behavior permeates through all layers of an enterprise influencing the enterprise's work environment and social structure [1][14][15].

Stakeholders, on the other hand, are purposeful systems which exhibit will which may act in concert with or oppose organizational goals and objectives and that the addition of technology to this equation [2][14][15][22]:

- Forces stakeholder to accept and adapt to new EA processes and procedures.
- Diminishes their influence and position within the enterprise.
- Alters the way in which they function within the enterprise from a consistent pattern of individual and social behavior to new ones that are less desirable.

However, enterprises must respond to competitive and shifting environmental demands by changing their existing operating environment and altering it to one focused on the use of technology to take advantage of the economic benefits to be derived from a project such as EA [12][20]. The resultant organizational transformation requires stakeholders to willingly or unwillingly accept and adapt to new ways of doing work which in turn changes their perception of the environment in which they function [2]. New rules, policies, standards, processes and procedures dictated by management to be followed and used by stakeholders typically results in new stakeholder behavioral patterns [14][15]. In many situations, organizational transformation is difficult to achieve because of these new behavioral tendencies. From a negative point of view, we can explore the effect of two candidates that can seriously affect and influence the success or failure of change, including stress and the need for EA stakeholders to learn, adapt and accept something new [14]. Stress is nothing new in any organizational setting to either stakeholders or the enterprise. However, it can take on a life of its own where technology is the prime motivation for change.

Stress caused by change is evident and easily recognized in stakeholders by their actions and behavior. Both stakeholder and organizational behavior tends toward a point where inputs, processes, and outputs remain stable with change viewed as a potential threat to the equilibrium and known state of the enterprise. Stakeholder resistance to accept change typically follows any movement away from this known state. In some cases, the stakeholder may even resort to sabotage to revert to the previous known state of equilibrium [13].

The rationale for this behavior traces to varying views of technology by people reacting to it accordingly, and for several legitimate reasons [8][14][15]:

- Low tolerance for change – the stakeholder believes the new is worse than the known.

- Misunderstanding - the stakeholder doesn't understand the reason for change.
- Power – the stakeholder has the power to ignore, obstruct, and avoid the new.
- Distrust – the stakeholder doesn't trust the enterprise's motivation for the change.

The transition to something new forces stakeholders to learn new ways to do work: processes, procedures, software, and other IT artifacts [3][22]. In most situations, this is accompanied with assignment of new roles, duties, and responsibilities [2][22]. This relearning process, in many cases, is simply beyond the day-to-day ability of some stakeholders to accept and adapt. Their tolerance level and threshold for change is limited. This results in a loss of productivity and, more importantly, negative changes in their behavioral patterns. When the change is involuntary, and imposed by internal and/or external forces (i.e., management), the change becomes emotional with stakeholders feeling a sense of disempowerment and loss of control, all adding to their feeling of stress [14][15]. Thus, these factors must be addressed by incorporating a dynamic and behavior driven approach to EA.

An appropriate theoretical lens that would enable an EA to be aligned with such an objective is Giddens' *Theory of Structuration* (ST) [8]. In ST, structure is understood to be an abstract property of social systems and in this context is not something concrete, situated in time and space, but lacks material characteristics. Structure does not and cannot exist apart from the human actors who enact and interpret its dimensions existing only in a virtual state. People, however, readily allow their actions to be constrained and limited by these shared abstractions of social structure suggesting that behavior can be strongly influenced and sometimes induced even by vague simulations of authority relationships and other organizational settings. The ability of organizational structures to elicit compliance and conformity in the absence of material constraints attests to the power of those socially constructed abstractions.

Given this perspective, structuration articulates a process-oriented theory that treats enterprises as both a product of, and a constraint on, human action. Giddens attempts to bridge the gap between the deterministic, objective and static notions of structure, on one hand, and voluntary, two realms of social order and focusing attentions on the subjective and dynamic views on the other, by positing points of intersection between these two realms. Giddens termed these as the *Institutional Realm* and *Realm of Human Action* [8]. The former represents the existing framework of rules in an enterprise derived from a

cumulative history of actions and interactions. Such a framework of rules is characterized by dimensions of signification, domination and legitimization. Signification schemes are modalities for communication within an organization and constitute organizational structures of signification. Structures of significance represent organization rules that define and inform interaction. Resources are modalities through which power is exercised in an organization and may be authoritative (i.e., extending over people) or allocative (i.e., extending over material/property). Norms are modalities that define appropriate behavior and constitute organizational structures of legitimization using which a “moral order within an organization is articulated and sustained through rituals, socialization practices and tradition” [8].

On the other hand, the *Realm of Human Action* refers to the social interaction of the humans under the aegis of the institutions. The institutions’ properties are encoded into the human actor’s stock of knowledge through the modalities of interpretive schemes, resources and norms, and influence how people communicate, enact power and determine what behavior to sanction and reward. The crux of Giddens’ theory is that this relationship is not directional but recursive. Organizational structural properties (i.e., the *Institutional Realm*) are drawn on by humans in their on-going interactions even as such use in turn reinforces or modifies the institutionalized structures. Such a recursive relation is termed as the duality of structure.

ST does not merely provide a means to understand the nature of an organization but can be applied to gain insight on the impact of the use of technology. Orlikowski proposed the *Structurational Model of Technology* (SMT) to provide a more complete model of understanding of how technology affects organizations [22]. This theory is based on the perceptions of the *Duality of Technology* and the *Interpretive Flexibility* of technology. The former posits that the socially created view and the objective view of technology is not exclusive but rather intertwined and are differentiated because of the temporal distance between the creation of technology and usage of the same. *Interpretive Flexibility* defines the degree to which users of a technology are engaged in its constitution (physically and/or socially) during its development. SMT has three components – the Human Agents, Technology and Institutional Properties of Organization. The model specifies an interactive relationship among these components that are essentially recursive in that each of these components influences and is at the same time influenced by the others. Technology is proposed to be the product of human action in that it

is created and exists through ongoing human action. Humans constitute technology by using it, while at same time making it an outcome of human actions such as design, development, appropriation and modification. However, once technology is implemented it facilitates and constrains human action through the provision of interpretive schemes, facilities and norms.

From the organizational perspective, institutional properties influence humans in their interaction with technology through: professional norms; rules of use – design standards and available resources. There is, however, a consequence of the institutional interaction with technology. They are manifested by impacting the institutional properties of an organization through reinforcing or transforming structures of signification, domination and legitimization that characterize the Institutional Realm.

In summary, the theoretical premise of these two theories is an acknowledgement that organizational structures, technology and human action are not distinct but are intertwined such that each is continually reinforced and transformed by the other [8][22]. A logical conclusion can therefore be made that an initiative such as the formulation of EA remains incomplete if it does not explicitly take into account human action. ST provides a framework, which if adopted could form a basis of a behavioral and inclusive approach towards formulating an EA. Specifically these theories provide a lens for the EIA to understand the dynamics of an enterprise and use that information to formulate an EA that is contextualized to that particular enterprise and advocated by its stakeholders.

The issue confronting the EIA is that of taking advantage of these circumstances recognizing that stakeholders are able to provide reasons for their activities, including perhaps even lying about them. However, this behavior can be managed by promoting an environment that encourages stakeholder participation in the decision-making process. EIAs are faced with three behavioral issues: the introduction of technology into enterprises, changes in stakeholder behavior resulting from technology and resistance to change with enterprises seeking equilibrium at the same time. The end-result of this behavior may result in “poor architecture”.

Successful implementation of new technology is the product of successfully navigating stakeholder behavior and the resultant influence on organizational change. For example, management practices often negatively influence stakeholder behavior because of inadequate knowledge being passed down and across enterprise boundaries. These issues can be addressed by implementing an open-ended communication’s

system where there are no boundaries, either horizontally or vertically, for sharing of knowledge, knowhow, ideas, potential problem solutions, and it provides a forum for “brainstorming.” This in effect, solves other issues such as poor communication and lack of understanding as to the rationale for EA and organizational change. It also serves as a mechanism to mitigate stress and a willingness to share and provide quality design requirements to the EA. As such, management behavior, attitudes, rules, and policies can avoid maintaining an ingrained mechanistic view of technology and approach EA from a more humanistic venue.

In this context, the actions of management and EIAs lead to changes in the way stakeholders behave. In a business context, stakeholder behavior and organizational factors contribute more to the success or failure of an EA than technical factors. Simply stated, stakeholders can be affected by IT change and are unlikely to be invested in the change if it is forced upon them without warning and input from them.

We envision an approach that highlights the impact of change on an enterprise relative to human behavior that can be utilized to enhance and extend the capabilities of well known architectural framework models used in an EA project. The approach fosters stakeholder ownership of the EA while building relationships through a coupling of EA and structuration.

4. Discussion and Closing Remarks

To manage and govern the complexity of an enterprise requires the coordination and control of activities embedded in the complex networks of techno-centric relations and boundary spanning exchanges. EA and the introduction of new and/or enhanced technology into an enterprise often results in a sociological and a political change in the hierarchical structure of the enterprise. This is evidenced by a dynamic shift in internal and perhaps external perceptions of the enterprise. Stakeholder roles, responsibilities, and duties invariably change because of new rules, policies, procedures and processes introduced by new technology. Therefore, the manner in which the EA design takes place can seriously affect acceptance and alignment of the EA by different stakeholders.

Among the many factors associated with EA failure are:

- Poor communication
- Lack of leadership
- Lack of top-management support and sponsorship

- Underestimating the importance of change and change management
- Lack of technical and business knowledge
- Poor project management

These factors are counter-productive yet they can be minimized and mitigated by providing an environment where stakeholders are involved with and are active participants in and are receptive to change. Such an environment fosters collaboration and information-sharing where stakeholders communicate both horizontally (i.e., peer-to-peer) and vertically (i.e., up and down the hierarchical organization chart) whenever and however they need to in order to solve problems and exchange knowhow and knowledge. The possibility and prospect becomes realizable if an enhanced working environment where participation in the design, decision-making, and implementation of new EA technology is welcomed and not perceived as a threat to stakeholder well-being. The benefits from such an environment can only improve workforce morale and productivity. In our increasingly digitally encoded environment, EIAs hold the potential to become major players who can assist enterprises achieve their respective goals and objectives.

In conclusion, the *Theory of Structuration* provides a means of understanding human behavior and its relationship to organizational change. *SMT*, on the other hand, addresses the effects of technology on human behavior [14]. Taken together, they conceptualize the unique opportunities for an EIA to implement an EA.

This paper progresses our earlier work [17][18][19] by expanding our exploration into the possibilities extant and the potential contribution gained as the result of using the *Theory of Structuration* along with *SMT* and thus improve on and enhance the EAF process. We consider a communication process based on human behavior the prime motivational element in behavior modification and offer it as a means to augment existing EAFs. Coupled with the communication process, future work includes expanding our research into two areas:

- To obtain a better understanding of human behavior and the influence it has on EA so as to provide a better platform to manage and govern the design process.
- To explore EA modeling schemes to assess their ability to cope with human behavior in their respective approach to requirements modeling. The focus of this effort is to better ensure the quality and reliability of design requirements input to EA.

Modifying human behavior to more readily accept organizational change represents a major component requiring design and implementation of tools and

mechanisms that facilitate stakeholder willingness to share knowledge. Quality requirements are essential to EA success. Therefore, we plan to expand our research and explore EA modeling schemes to assess and better ensure EA quality through the requirement elicitation, analysis, and specification phases of EA.

5. References

- [1] "The DoD Architecture Framework Version 2.0, DoD Deputy Chief Information Officer." Department of Defense, May, 2009.
- [2] M. Beer. *Organizational Behavior and Development*. Harvard Business Review, Harvard University, 1998.
- [3] M-C. Boudreau and D. Robey. "Enacting Integrated Information Technology: A Human Agency Perspective." *Organization Science*, 16(1):3-18, 2005.
- [4] Chief Information Officers Council. *Federal Enterprise Architecture Framework*, CIO Council, Version 1.1, August 5, 1999.
- [5] C. Ferreira and J. Cohen. "Agile Systems Development and Stakeholder Satisfaction: A South African Empirical Study." *Proceedings 2008 Conference of South African Institute Computer Scientists and Information Technologists on IT Research in Developing Countries*, pp. 48-55, 2008.
- [6] D. Galorath. *Software Project Failures Costs Billions: Better Estimation & Planning Can Help*. Filed under Project Management, June 7, 2008.
- [7] R. Gauld. "Public Sector Information System Failures: Lessons from a New Zealand Hospital Organization." *Government Information Quarterly*, 24(1):102-114, 2007.
- [8] A. Giddens. *The Constitution of Society: Outline of the Theory of Structuration*. University of California Press, 1984.
- [9] B. Iyer and R. Gottlieb. *The Four-Domain Architecture: An Approach to Support Enterprise Architecture Design*. In IBM Systems Journal, 43(4):587-597, 2004.
- [10] M. Lankhorst & H. von Drunnen. *Enterprise Architecture Development and Modeling, Via Nova Architecture*. March, 2007.
- [11] B. Lawhorn. *More Software Project Failures*. CAI, March 31, 2010.
- [12] A. Lindstrom, et al. *A Method to Assess the Enterprise-wide IT Resource for Performance and Investment Justification*. Dept. of Industrial Information Systems and Control Systems, Royal Institute of Technology, KTH, SB-100, 44 Stockholm. Sweden, 2005.
- [13] R. Lewin and B. Regine. "Enterprise Architecture, People, Process, Business, Technology." Institute for Enterprise Architecture Developments [Online], Available: <http://www.enterprise-architecture.info/Images/ExtendedEnterprise/ExtendedEnterpriseArchitecture3.html>.
- [14] A. Maslow. *Motivation and Personality*. Harper Collins, 1987.
- [15] D. McGregor. *The Human Side of Enterprise*. McGraw-Hill, 1960.
- [16] N. Melville, K. L. Kraemer, and V. Gurbaxani. 2004 Review: *Information Technology and Organizational Performance: An Integrative Model of IT Business Value*. MIS Quarterly, Volume 28, Number 2, pp. 283-322, June 2004.
- [17] D. M. Mezzanotte, Sr., J. Dehlinger, and S. Chakraborty. "Applying the Theory of Structuration to Enterprise Architecture Design." *2011 World Conference in Computer Science, Computer Engineering, and Applied Computing*, IEEE/WorldComp 2011, SERP 2011, July, 2011.
- [18] D. M. Mezzanotte, Sr., and J. Dehlinger, "Enterprise Architecture: A Framework Based on Human Behavior Using the Theory of Structuration." *International Association of Computer and Information Science, 2012 IEEE/ACIS 10th International Conference on Software Engineering Research, Management, and Applications*, 2012.
- [19] D. M. Mezzanotte, Sr., J. Dehlinger, and S. Chakraborty. "On Applying the Theory of Structuration in Enterprise Architecture." *Computer and Information Science, 2010 IEEE/ACIS 9th International Conference on Software Engineering Research*, pp. 859-863, 2010.
- [20] D. Minoli. *Enterprise Architecture A to Z*. CRC Press, New York, 2008.
- [21] The Open Group. *TOGAF Version 9*. 2009.
- [22] W. Orlikowski. "The Duality of Technology: Rethinking the Concept of Technology in Organizations." *Organization Science*, 3(3):398-427, 1992.
- [23] M. S. Poole and G. DeSanctis. *Structuration Theory in Information Systems Research: Methods and Controversies*. Handbook for Information Systems Research, M. E. Whitman and A. B. Woszczanski (eds.), Hershey, PA., Idea Group Publishing, 2004.
- [24] R. S. Pressman. *Software Engineering: A Practitioner's Approach*. 7th Ed., McGraw-Hill Series in Computer Science, New York, NY, 2010.
- [25] Roeleven, Sven and J. Broer. "Why Two Thirds of Enterprise Architecture Projects Fail." *ARIS Expert Paper* [Online], Available: http://www.ids-scheer.com/set/6473/EA_-_Roeleven_Broer_-_Enterprise_Architecture_Projects_Fail_-_AEP_en.pdf.
- [26] N. Rozanski and E. Woods. *Software Systems Architecture*. Addison-Wesley Professional, 2006.
- [27] R. Sessions. *A Comparison of the Top Four Enterprise-Architecture Methodologies*. MSDN Library, May 2007.
- [28] I. Sommerville, *Software Engineering*, 8th Ed. Addison-Wesley Publishers, Harlow, England, 2007.
- [29] B. Travica. *Information View of Organization*. Journal of International Technology and Information Management, 14 (3), 2005.
- [30] J. Zachman, *Concepts of the Framework for Enterprise Architecture*. Information Engineering Services, Pty, Ltd., 1987.

SOAs factors, criteria and metrics. SERP 2012

R. Belkhatir¹, M. Oussalah¹, and A. Viguiet²

¹Department of Computer Science, University of Nantes, Nantes, Loire Atlantique, France

²Research and Development Department, BeOtic, Rezé, Loire Atlantique, France

Abstract – *The current paper presents a semi-automated method for evaluating SOAs called SOAQE. This method corrects defects observed so far such as lacks of pertinence and accuracy for evaluation results. SOAQE takes as a starting point the McCall model, describing software quality, which led to an international standard for the evaluation of software quality (ISO 9126). This model is organized around three types of quality attributes (factors, criteria and metrics). The method consists in decomposing the whole architecture and evaluating it according to the McCall model, i.e. a list of quality factors arising from business needs, grouping criteria composed by metrics. Our experimentations led us to quantify numerically a first determining factor for SOAs, the ‘dynamism’ and some attributes of its structure: namely the ‘loose coupling’ criterion and its constituent metrics (‘physical, semantic and syntactic’).*

Keywords: SOA, quality, evaluation, factor, criteria, metric.

1 Introduction

An architectural paradigm defines groups of systems in terms of models of structures; component and connector vocabularies and rules or constraints on relations between systems [1]. We can distinguish a few architectural paradigms for distributed systems, and, among the most noteworthy ones, three have contributed to the evolution of the concerns. These are chronologically object oriented architectures (OOA), component based architectures (CBA) and **Service oriented architectures (SOA)**. First developers were quickly aware of code repetitions in applications and sought to define mechanisms limiting these repetitions. **OOA** provides great control of the **reusability** (*reusing a system the same way or through a certain number of modifications*) which paved the way to applications more and more complex and consequently to the identification of new limits in terms of granularity. These limits have led to the shift of the concerns towards the **composability** (*combining in a sure way its architectural elements in order to build new systems or composite architectural elements*). Correlatively, the software engineering community developed and introduced **CBA** to overcome this new challenge and thus, the CBA reinforces control of the composability and clearly formalizes the associated processes. By extension, this formalization establishes the base necessary to automation possibilities. At

the same time, a part of the software community took the research in a new direction: the **dynamism** (*developing applications able to adapt in a dynamic, automatic and autonomous ways their behaviors to answer the changing needs of requirements and contexts as well as possibilities of errors*) concern as the predominant aspect. **SOA** has been developed on the basis of the experience gained by objects and components, with a focalization from the outset on ways of improving the **dynamism**. Developing an **SOA** involves many risks, so much the complexity of this technology is notable (*particularly for services orchestration*) [2]. First and foremost among these, is the risk of not being able to answer favorably to expectations in terms of quality of services because quality attributes directly derive from business objectives. As these risks are distributed through all the services, the question of evaluating SOA has recently arisen. This is to identify and correct remaining errors that might have occurred after the software design stage and, implicitly, to reduce subsequent risks. Lots of tools have been created to evaluate SOAs but none of them clearly demonstrated its effectiveness [3]. The SOAQE method presented in this paper allows evaluating SOAs by combining the computerized approach and the human intervention to eliminate lacks identified with past methods. The main idea resides in automating the process to avoid time-wasting evaluations. The process consists in three principal stages detailed later in this paper. In the current paper, we first relate in the section 2 the interests of the evaluation and we also analyze the existing works. We present the SOAQE method and the stakeholders participating to the evaluation in the section 3. We finally relate the experimentation which has been done by the lab team in the section 4. We conclude this paper with a discussion comparing the SOAQE method to past ones.

2 State of the art

The SOA evaluation relates to qualitative and quantitative approaches, load prediction associated with evolutions and theoretical limits of a given architecture.

2.1 Related works on SOA evaluation

From this perspective, tools and existing approaches have shown their limitations for SOA [3]. We are currently attending the development of a new generation of tools developed by industrialists in a hand-operated way [4]. The

scale of the task has brought the academic world to tackle these issues and to try to develop a more formal and generic approach than different existing methods (*ATAM*, *SAAM*) to evaluate SOAs [3].

In any software architecture evaluation, three activities are critical to success:

1. Specifying the quality attributes deriving from the commercial needs.
 - *The current paper focuses on a finite set of technical attributes having significant impacts on the global quality, from the development process to the system produced as outcome* [5].
2. Selecting a representative assembly of stakeholders for the evaluation.
 - *In this paper, we work with a restricted and technical assembly of stakeholders listed in section three.*
3. Describing the architecture in a clear, expressive and understanding way.
 - *The present paper does not treat this part of the problematic because it has always been covered by the lab team in a previous paper* [6].

2.2 Evaluation Results

In concrete terms, SOA evaluations product a report which form and content vary depending on the method used. But, in general terms, the evaluation generates textual information and answers two types of questions [3]:

- 1- Is the architecture adapted to the system for which it has been conceived?
- 2- Is there any other architecture more adapted to the system in question?

-1- It could be said that the architecture is adapted if it favorably responds to the three following points:

- i. The system is predictable and could answer to the quality requirements and to the security constraints of the specification.
- ii. Not all the quality properties of the system result directly from the architecture but a lot do; and for those that do, the architecture is deemed suitable if it makes it possible to instantiate the model taking into account these properties.
- iii. The system could be established using the current resources: the staff, the budget, and the given time before the delivery. In other terms, the architecture is buildable.

This definition will open the way for all future systems and has obviously major consequences. If the sponsor of a system is not able to tell us which are the attributes to manage first, well, any architecture will give us the answer [3].

-2- A part of SOA evaluation consists in capturing the quality attributes the architecture must handle and to prioritize the

control of these attributes. If the list of the quality attributes is suitable in the sense that at least all the business objectives are indirectly considered, then, we can keep working with the same architecture. Otherwise, it is time work with architecture more suitable for the system.

These quality attributes may be conflictive for achieving business objectives. In such a case, the project manager must take the decision to focus on a limited set of attributes, especially if the evaluation of the architecture gives a positive result in a sector and a poor one in another sector [7]. However, provided that it is still able to favorably answer to non-functional and functional requirements of the specification, the architecture is able to exist with these rare failures. Moreover, since each sector is linked to a list of objectives and, these latter are pursued by focusing on certain quality attributes; the best way to reinforce the neglected sectors is to produce a more robust work on attributes correlated to the sector in question. The evaluation will help to indicate where the architecture fails, but the balance between the cost of the evaluation and the help it provides to ameliorate the project remains relative to any architecture. Then, an architecture evaluation does not provide answers to whether or not the architecture is adapted; if it is "good" or "bad" or if it is rated "seven out of ten"; it only tells us where the danger is. The evaluation could be applied to a single SOA or to a group of several SOAs competing to be chosen for the final system. In this last case, it first identifies the relevant business objectives needed in the comparison and then, examines available documentation for each architecture candidate. Then, it finally scores the fitness of the candidate architecture, summarizes the analysis results and provides a recommendation for the decision-making process.

2.3 Measuring the quality

It has been suggested that software production is out of control because we cannot quantitatively measure it. As a matter of fact, Tom DeMarco (1986) stated that "*you cannot control what you cannot measure*" [8]. The measurement activity must have clear objectives and a whole set of sectors need to be measured separately to ensure the right management of the software. For example, we know that the administrator must measure the software quality in order to compare projects, make provisions and set reasonable improvement objectives. In order to bring these operations to fruition, the scientific community utilizes models of quality introducing what we call software attributes decomposition.

2.3.1 Mc Call model

One of the models that have been published is the McCall model in 1977 decomposing quality attributes in three stages. This model led to the IEEE standard: *ISO/IEC 9126*. A certain number of attributes, called external (*applicable to running software*), are considered as key attributes for quality. We call them **quality factors** [3] (for example, the reliability: *ability of a system to keep operating over time*). These

attributes are decomposed in lower level attributes, the internal attributes (*do not rely on software execution*), called **quality criteria** (for example, the testability is one of the "maintainability" criteria). And each criterion is associated to a set of attributes directly measurable and which are called **quality metrics** (for example, the testability criterion is measured by the statement coverage: *evaluation of the achievable instructions percentage exerted*, the connections coverage: *evaluation of the active connections percentage*, etc.).

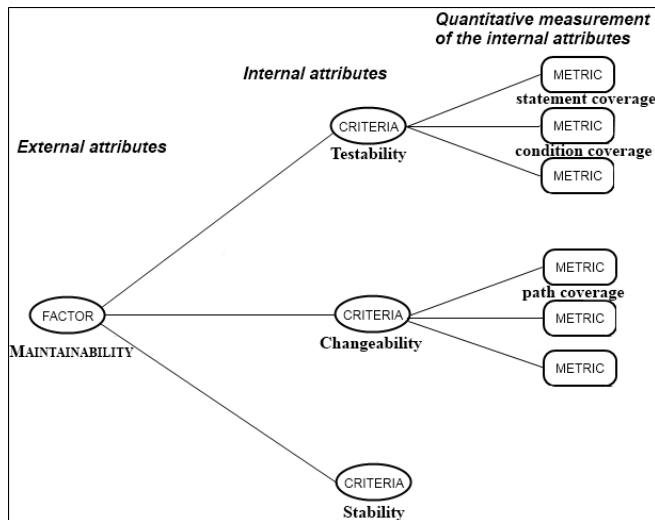


Figure 1: Mc Call model applied to the "Maintainability".

2.4 Lack of precision

Current methods of evaluation stop the quality attributes decomposition at the "quality factors" step and thus remain too vague when it comes to giving accurate measures. These methods are not precise because they cannot go further in the decomposition and consequently they cannot be automated to the point of defining a finite value for each attribute. They lead inevitably to the establishment of a brainstorming between stakeholders (*see section 3.2.4*) for the purpose of the institution of a utility tree. Because stakeholders do not have tools to quantitatively measure each quality factor chosen, they shall set up scenarios aiming to respectively solicit separately each criterion, and factor. An approximated evaluation of the architecture is then realized after having studied the system behavior while carrying out the scenarios set.

3 SOAQE

It is precisely where our work differs from those existing insofar as we wish to obtain a precise quantitative measurement for each quality factor with the SOAQE method. We especially aim to automate the process in order to avoid hand-operated evaluations pushing to solicit stakeholders for the whole evaluation.

3.1 Principle of the model

The process consists in three principal stages.

Each corresponds to a decomposition step of our quality attributes. We first identify decisive quality factors for our architecture. Then we isolate the quality criteria defining them. And finally we define quality metrics composing each criterion in order to quantify them numerically.

3.2 Steps in more details

The main idea of the SOAQE process is to evaluate in three steps the whole architecture from every metric to the set of quality factors obtained after having previously identified the business objectives. Our work is based on the architect point of view and the attributes selected are the ones considered as the most relevant among all existing.

3.2.1 Quality factors

Some authors have defined the CBA with *reusability* and *composability* [9]. Basing on previous analysis, we define the SOA with the *Reusability*, the *Composability* and the *Dynamism*.

These three attributes, that we identified as **quality factors** for SOA represent the qualitative quintessence which has directed the definition of the object, component and service paradigms. The figure 2 illustrates this analysis and offers a high-level vision of the SOA interest points.

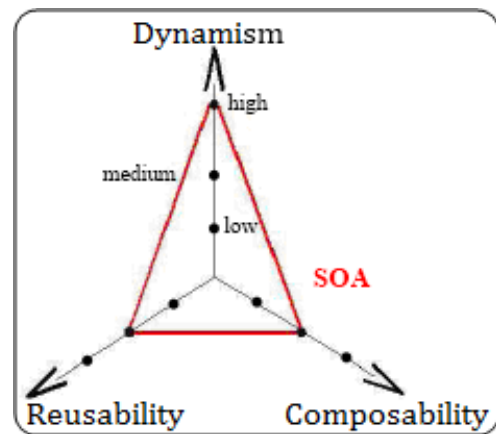


Figure 2: SOA interest points

The **first step** of the SOAQE method consists in choosing a first quality factor to study in depth and there exist a lot which could come out after the analysis of the business objectives. But we have naturally chosen to work on those identified as the qualitative quintessence for SOA. These three quality attributes (*dynamism, composability and reusability*) define each of our three paradigms to varying degrees. Moreover, there exist a hierarchical ranking propelling "dynamism" on top of SOA concerns, and this is precisely

why, we chose to especially focus deeply on this quality factor. We may record that each of the two others attributes is of major importance for the three paradigm considered (object, component or service oriented architectures).

3.2.2 Quality criteria and quality point of views (QPV)

With regards to our work and after having identified the determining factor quality for SOA (i.e. the dynamism), we were interested in the **second step** of the SOAQE process, namely discovering the criteria defining the factor on which we have gone through.

Further down the road, any factor is composed by a lot of criteria that could be looked at as part of our work.

There exist quality points of views (QPV) which group together criteria according to their characteristics. The QPV notion is fundamental because it allows company to target their evaluation on selected families of criteria and these families of criteria are common to all existing architectures.

We deliberately concentrated our work on a technical QPV (see figure 3) grouping technical criteria because we adopted the point of view of an architect (a technical stakeholder). In this light, we identified six criteria common to each of our three factors. These technical criteria gather elements having significant impacts on global quality, from the development process to the system produced.

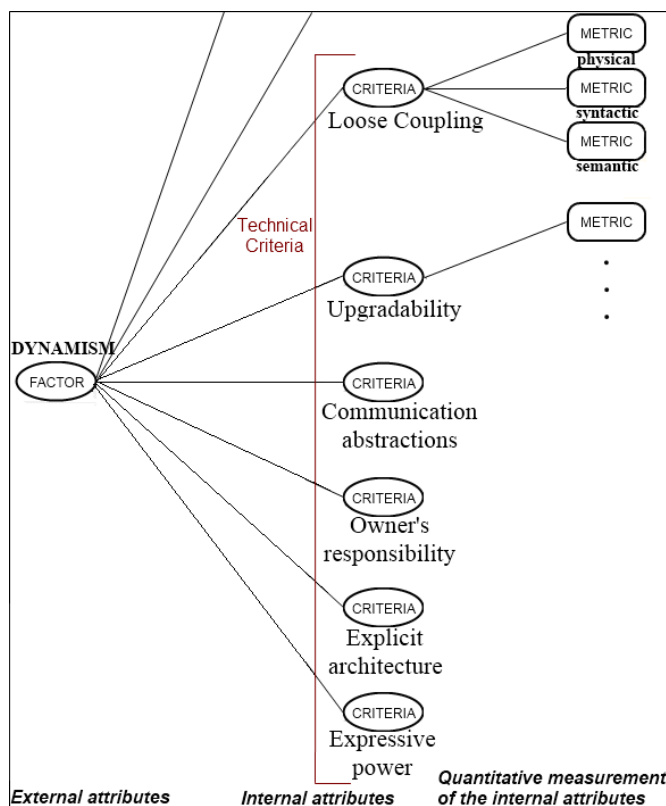


Figure 3: McCall model applied to the quality factor "dynamism".

Loose coupling: Potential of dependences reduction between services and dependences between processes.

Explicit architecture: Paradigm ability to define clear architectural application views, i.e. providing the means of identifying and clarifying functionalities associated to services composing the system.

Expressive power: Potential of paradigm expression in terms of creation capacity and optionalities. It is based on the number of processes provided to specify, develop, handle, carry out and implement services.

Communication abstractions: Paradigm capacity to abstract services functions communications.

Upgradability: Paradigm ability to make evolve its services (based on processes supporting these evolutions).

Owner's responsibility: Corresponds to the responsibilities sharing out between services providers and consumers. These responsibilities are focused on the software entity re-used in terms of development, service quality, maintenance, deployment, execution and management. This distribution expresses the degree of freedom granted to service consumers by the service provider.

Each of these quality criteria is given varying degrees of consideration according to the quality factor in question. Our previous works [5] allowed organizing hierarchically (under three distinct levels) these quality criteria for each of the three quality factors (dynamism, reusability and composability).

Consequently, we obtain the triptych of the figure 4 while considering all paradigms.

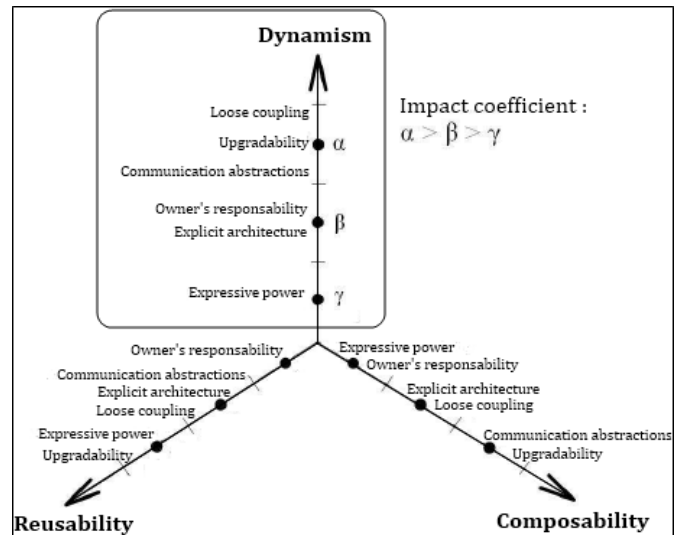


Figure 4: Expression of reusability, composability and dynamism perspectives.

While focusing on the dynamism, identified earlier as being the key quality factor for SOA, our previous work [5] allowed to conclude that the "loose coupling" criteria is of biggest importance for this factor (see figure 4), this is why we chose to concentrate in depth, on it whereas, the criterion "expressive power" is of less importance.

3.2.3 Quality metrics

The coupling is relatively well known by the community, and thus, our lab team members decided to look into the matter [5]. Correlatively, we found three quality metrics for the latter which must be considered for the **third and final step** of the SOAQE process, that is to say, the quantification and the evaluation of all the metrics (The semantic coupling: based on the high-level description of a service defined by the architect, the syntactical coupling: measures dependencies in terms of realization between abstract services and concrete services and the physical coupling: measures dependencies between concrete services really utilized, in collaboration and in execution). In this light, we can draw a triptych clearly presenting the metrics extracted and levels of acceptance for each of them (see figure 5).

Semantic coupling

High coupling: The service takes part in an essential functionality of the composite.

Low coupling: The service takes part in a nonessential functionality of the composite. The global quality is not guaranteed anymore if one or more from these functions are withdrawn. We pose that if all these nonessential functions disappear the composite becomes unusable.

Non-predominant coupling: An abstract service and a composite are in non-predominant coupling if this service takes part in a nonessential function of the composite and if the withdrawal of this function does not have any significant impact on the global quality.

Syntactical coupling

High coupling: An abstract service is in high syntactical coupling with its concrete solution if this solution (a concrete service or a composition of concrete services) represents the single possibility of realization.

Weak coupling: An abstract service and a concrete service are in weak coupling if there are several concrete alternatives to the realization of this abstract service.

Physical coupling

The physical coupling focuses on the implementation of the service. This implementation corresponds to a particular instance of the service where a choice has been made concerning concrete services to use. A unique solution has been chosen to fulfill each of the needs expressed by abstract services. It reuses existing researches [10] and it is based on measurements such as methods calls, messages exchanged, the number of linked services, commune objects and so forth. These metrics shall make it possible to identify physical dependencies between concrete services.

3.2.4 Stakeholders

Further down the road, each of the SOAQE method steps require the intervention of external evaluators (called stakeholders) to set the coefficients prioritizing factors, criteria and metrics considered. The categories of stakeholders are often the same for the SOA systems, especially when we work

with criteria from the same family. Here is a technical list of stakeholders chosen for evaluating technical attributes.

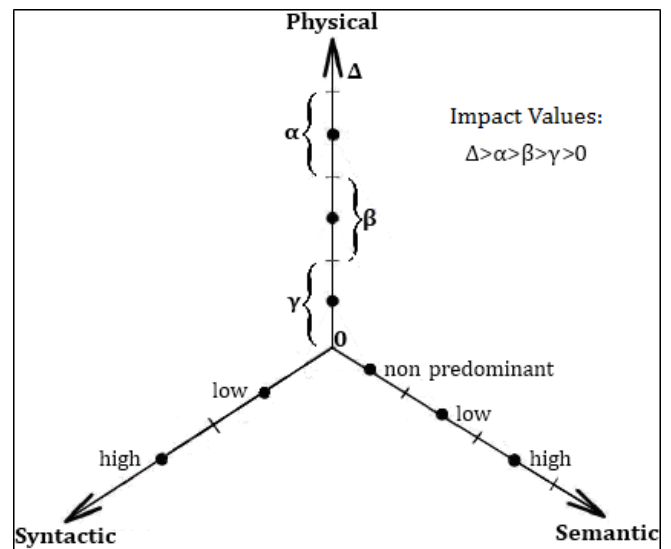


Figure 5: Loose coupling metrics

We have the **software architects** which main responsibilities include experimenting with and deciding between different architectural approaches, the **developers** which main responsibilities include implementing the architectural elements of the system according to the architecture specification; the **integrators** which main responsibilities are to ensure that the architecture and implementation conform to open and widely accepted standards; the **maintenance developers** which main responsibilities include modifying the software to correct defects and adapting the software when environmental changes occur (e.g., hardware or operating system changes). We also have the **developers of service users**: these external developers may provide input on application program interface (API) design and desired quality of service and finally the **external developers of service providers**: they may contribute requirements for interaction with their services, as well as knowledge of qualities and limitations of their systems.

3.2.5 Coefficients

Concerning the first step of SOAQE, coefficients assigned to the factors will depend on the company needs. Our works led us to conclude that for SOA and the three factors we worked with, we would allocate, according to our hierarchical ranking, a coefficient of '3' for the "dynamism" whereas we would affect the value '2' for the "reusability" and the "composability".

With regards to the second step, our works led to list the six technical quality criteria chosen under three distinct levels of acceptance, α , β and γ at which we assign respectively the values '3, 2 and 1', consequently, the "loose coupling", the "upgradability" and the "communication abstraction" will be

allocated the value '3'. The coefficient '2' goes for the "owner's responsibility" and "explicit architecture" criteria and '1' for the "expressive power".

And finally, the three metrics studied may be all assigned to the value '1' meaning that they are equally important for calculating the global coupling of SOAs.

These coefficients will be used as a basis for the following section (Experimentation). They have been affected to quality attributes as an example; however, these latter have been chosen according to the principle of proportionality validated by the lab-team. We can select other impact coefficients providing that we keep the same proportionality between the quality-attributes considered.

4 Experimentation

For the experimentation, we tempted to quantitatively measure the key quality attributes discussed in the previous sections of this paper; notably, the quality factor "dynamism", the "loose coupling" criteria and the "physical, syntactic and semantic coupling" metrics. That being said, it is important to note that the SOAQE method must be reproduced for every quality factor identified after having analyzed the objectives of the company and the set of criteria and metrics belonging to that quality factor.

4.1 Loose Coupling

Taking as a starting point an existing formula of the field of "Preliminary analysis of risks" (see formula 6.1) [11], our works led to the identification of a mathematical formula (see formula 6.2) combining the three couplings studied: semantic, syntactic and physical.

NB: The simplified formula (see formula 6.1) usually used in the automotive industry, makes it possible to measure the default risk of a car component A is the Criticality of the car component, B is the Probability of occurrence of a failure on this component and C is the Probability of non-detection of this failure.

$$R = A * B * C \tag{6.1}$$

$$Coupling = \left\{ \left\{ (a), (b), (c) \right\} * \left(\left(\prod_{k=1}^{\lambda} \prod_{i=1}^N \sum_{j=1}^i ((P_s)_{kij})^{\alpha_{kij}} \right) * C_{phys} \right) * \left\{ P_{ndetect} \right\} \right\} \tag{6.2}$$

We associate this concept of risk with our vision of the coupling. Correlatively, the quintessence of the coupling is the expression of the dependences which can exist between two elements and the principle of dependence defines that one element cannot be used without the other. Reducing the risk that the role defined by a service cannot be assured anymore is decreasing the dependence of the application in relation to this service and thus reducing its coupling. The calculation of this

risk takes into account all the characteristics influencing the coupling by redefining the three variables A, B and C according to the semantic, syntactic and physical couplings. The global coupling corresponds to the sum of the three couplings calculated individually beforehand. The lower this result is, the more the coupling is weak.

NB: The criticality $A \in [(a),(b),(c)]$ is affiliated to the semantic coupling. 'a' if the service is only associated to non predominant couplings, 'b' for non predominant and low couplings and 'c' for non predominant, low and high couplings, while 'Ps' is the probability of failure of a service.

This generic coupling formula can directly be used to quantify the quality of an architecture by weighting up each of the attributes concerned by means of the coefficients isolated after having hierarchised the attributes according to their importance. Indeed, as we already specified in section 2, we cannot automate this operation and define continuously the same coefficients for all the architectures considered because this operation is specific to the business objectives of the company.

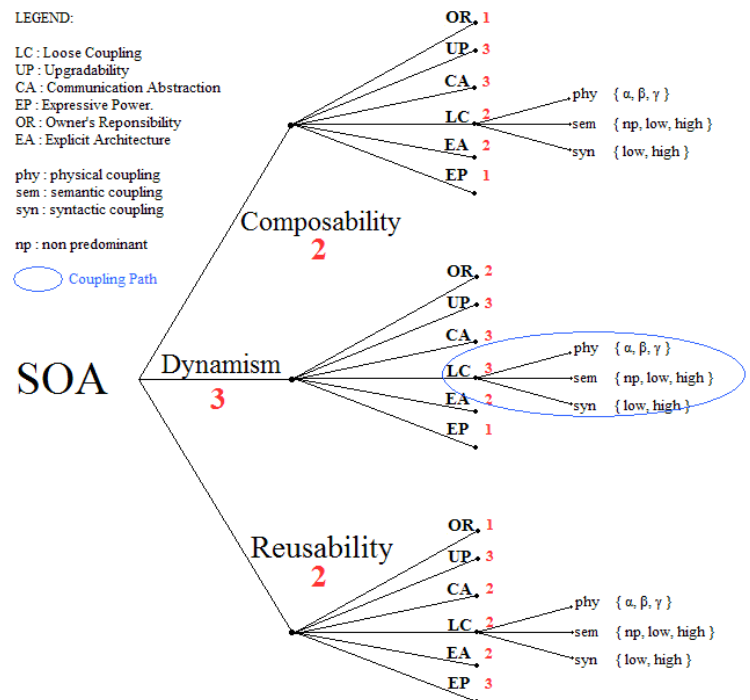


Figure 7: SOA attributes tree weighted with means of coefficients

By applying to known quality attributes the coefficients determined in the section 3.2.5, we obtain the tree of the figure 7. According to this tree, we can establish that the quantitative measure of the quality of an SOA corresponds to the sum of the quality factors dynamism, reusability and

composability, all three affected by their respective coefficients. The formula hereinafter allows calculating the whole quality of an SOA.

NB: the lower the loose coupling result is, the more the coupling is weak. Conversely, the higher the architecture quality result is, the more the quality is good.

The result of each criterion is expressed in percentage, this is why we subtract to 1 the result found.

$$SOA = 3Dynamism * 2Reusability * 2Composability$$

$$SOA = 3(2OR + 3UP + 3CA + 3(1 - LC) + 2EA + EP) * 2(OR + 3UP + 2CA + 2(1 - LC) + 2EA + 3EP) * 2(OR + 3UP + 3CA + 2(1 - LC) + 2EA + EP)$$

$$SOA = 3(2OR + 3UP + 3CA + 3 [1 - ((a), (b), (c)) * \left(\prod_{k=1}^{\lambda} \prod_{i=1}^N \sum_{j=1}^i ((P_{kij})^{n_{kij}}) * C_{p_{kij}} \right) * P_{ndetect}]) + 2EA + EP) * 2(OR + 3UP + 2CA + 2 [1 - ((a), (b), (c)) * \left(\prod_{k=1}^{\lambda} \prod_{i=1}^N \sum_{j=1}^i ((P_{kij})^{n_{kij}}) * C_{p_{kij}} \right) * P_{ndetect}]) + 2EA + 3EP) * 2(OR + 3UP + 3CA + 2 [1 - ((a), (b), (c)) * \left(\prod_{k=1}^{\lambda} \prod_{i=1}^N \sum_{j=1}^i ((P_{kij})^{n_{kij}}) * C_{p_{kij}} \right) * P_{ndetect}]) + 2EA + EP)$$

For any architecture considered, we are able to determine a finite value for the loose coupling criteria, the remaining work consists in defining a way to calculate the five others criteria in order to isolate a finite value for the quality.

5 Discussion

Because SOA implies the connectivity between several systems, commercial entities and technologies: some compromises regarding the architecture must be undertaken. Forasmuch as the decisions about SOA tend to be pervasive and, consequently, have a significant impact on the company; setting an evaluation of the architecture early in the life of the software is particularly crucial. During software architecture evaluations, we weigh the relevance of each problematic associated to the design after having evaluated the importance of each quality attribute requirement. The results obtained when evaluating software architectures with existing methods (*ATAM*, *SAAM*) are often very different and none of these latter carries out it accurately [12]. We know the causes of this problem: most methods of analysis and automatic quality evaluation of software systems are carried out from the source code; whereas, with regard to evaluation cases of architectural models, the analysis is conducted based on the code generated from the model. From this code, there exist calculated metrics, more or less complex, associated with algorithms, methods, objects or relations between objects. From an architectural point of view, these techniques can be indicated of low level, and can be found out of step with projects based on new complex architectures. The evaluation concerns qualitative and quantitative aspects, the prediction of the load associated to evolutions and on the theoretical limits of a given architecture. These architectures evaluations can be made as well on architectures under development as on existing ones.

6 Conclusion

The finality of our work is to design a conceptual framework and, in fine, a semi-automated prototype (*based on*

past methods, such as ATAM or SAAM) which could quantify with an accurate value the quality of the whole service oriented architecture. Another pursued goal consists in bringing to the customer "less abstract" documents than those proposed today. The quality concept remaining a relative one, we will target the sectors requiring a special attention by directly addressing the various development lab teams charged with the relevant functions.

7 References

- [1] D. Garlan and M. Shaw, An introduction to software architecture, CMU/SEI-94-TR-21, ESC-TR-94-21, 1994.
- [2] S. Mazumder. SOA: A Perspective on Implementation Risks. *SETLabs Briefings* publication (eg. Oct 2006).
- [3] P. Clements, R. Kazman and M. Klein, Evaluating Software Architectures: Methods and case studied, published by Addison-Wesley Professional, 2001.
- [4] A. Sangroya, K. Garg and V. Varma. SAGE: An Approach to Evaluate the Impact of SOA Governance Policies. 2010. *AINA Workshops 2010: 539-544*.
- [5] A. Hock-Koon, "Contribution à la compréhension et à la modélisation de la composition et du couplage faible de services dans les architectures orientées services" Thesis (PhD). University of Nantes, 2011.
- [6] M. Oussalah and A. Smeda. COSABuilder : an Extensible Tool for Architectural Description, 2008. *ICTTA 2008*, pages 1–6.
- [7] O. Lero, P. Merson and L. Bass. Quality Attributes and Service-Oriented Architectures, 2007. *SDSOA 2007*
- [8] T. Demarco, Controlling software projects: management, measurement and estimates, Prentice Hall, 296 pages, 1986.
- [9] I. Crnkovic, M. Chaudron and S. Larsson, "Component-based development process and component lifecycle" ICSEA'06, International Conference on Software Engineering Advances, 2006
- [10] M. Pereplechikov, C. Ryan, K. Frampton, and Z.Tari. Coupling metrics for predicting maintainability in service-oriented designs, 2007. *ASWEC:329–340*
- [11] Y. Mortureux, Preliminary risk analysis. Techniques de l'ingénieur. Sécurité et gestion des risques, SE2(SE4010):SE4010.1–SE4010.10, 2002
- [12] M.T. Ionita, D.K. Hammer and H. Obbink, "Scenario-based software architecture evaluation methods: an overview", ICSE 2002, 2002

Scalability Architecture For Processing using Microsoft .Net Remoting

Seth Nielsen and Dr. Yuke Wang

School of Computer Science, University of Texas at Dallas, Richardson, Texas, U.S.A.

Abstract - Microsoft .Net Framework provides intrinsic capabilities for distributed computing but does not provide prioritization and scaling services. As more applications and database systems convert from mainframe systems to Microsoft .Net Framework, a framework to provide these services is needed. Microsoft provides for some of these services with more advanced offerings, but with increased operational requirements and licensing costs.

This paper will define the architecture for a software-based solution that will require only .Net Framework and SQL Server to provide scaling, scheduling, prioritization, logging and load balancing services. This solution provides for horizontal scaling of defined processes that can further break themselves into sub-processes and submit these to be run on additional physical servers without the calling application being aware. The load balancing solution described here will not only attempt to locate the best fit server for the requested process, it will allow for processes to be assigned only to servers defined as capable to run the process.

Keywords: Software Architecture, Web-based Applications, Distributed and parallel systems

1 Introduction

Microsoft .Net Framework is a common software solution and provides many higher level functions. However, these solutions do not naturally provide many of the services available to mainframe systems. This includes scaling, scheduling, prioritization, logging and load balancing services. Microsoft and other vendors provide large-cost solutions, but these require large expenditures of hardware and pre-built software solutions that have large annual fees and maintenance fees. As companies move away from mainframe systems where process scaling is handled intrinsically, modern software solutions need to provide these same services.

1.1 Objectives

This paper will provide a mission-critical process scaling architecture that utilizes only Microsoft .Net Framework and Microsoft clustered SQL Server. This will allow more

software systems to provide scaling and load-balancing services for CPU, memory and query intensive processes with quick turn-around for fast and short processes at the same time.

1.2 Why is this important?

Many smaller to mid-size companies have found that the mainframe implementations are too expensive to purchase and maintain. Microsoft has provided a relatively inexpensive software solution that does not provide standard mission-critical solutions for their growing needs. This paper will describe a solution that can be implemented by the typical software development team utilizing the tools they are most familiar – Microsoft .Net Framework and Microsoft SQL Server.

1.3 Brief Summary of Existing Approaches

There are several third party solutions that currently provide scalability services, but they are either high priced or do not meet the mission critical needs of high availability.

Microsoft provides several solutions in this regard. Network Load Balancing (NLB) Services are offered as an extension of Windows Server 2008 [14]. NLB offers fault tolerance and will allow scaling to 32 servers per segment, but does not ensure that the destination server is running the service needed [16]. With no load balancing mechanism built in, the solution does not offer enough benefit for the software team not already using NLB for other purposes.

Microsoft also offers SQL Service Broker (SSB) as part of Microsoft SQL server tools. SSB provides a messaging and queuing solution that uses the same transactions utilized in data modifications to provide distributed processing [11]. However, the message queue allows communication in a first-in, first-out order so does not easily provide for the prioritization services that are needed for a scalable solution. The SSB solution also requires most of the communication logic to be stored in stored procedures on the SQL Server, which makes debugging more difficult.

Microsoft MQ (MSMQ) [17] and IBM WebSphere MQ [12] are also solutions for scalability. The intent of message queuing systems is to break down processing into very small pieces and handle each item individually within the queue. This requires a significant amount of work to then collate all

returned information together and does not work well when significant amount of preparation/data access is needed to complete the processing. Both MSMQ and WebSphere MQ are expensive solutions which require a significant amount of management and oversight [17].

1.4 Summary of the Remaining Paper

The remainder of the paper will include the following sections. Section 2 briefly describes the Microsoft solutions to be utilized by this solution and an overview of scalability. Section 3 describes the Process Scaling Architecture solution and shows how it meets the needs as discussed here. Section 4 will show the results and Section 5 provides the conclusions.

2 Background

2.1 Microsoft .Net Framework

Microsoft .Net Framework developed by Microsoft Corporation provides a consistent object-oriented programming environment using a common language runtime (CLR) and automatic garbage collection [18]. The framework provides a substantial amount of tools for development teams.

2.2 Clustered Microsoft SQL Server

Microsoft SQL Server 2008 provides data storage and retrieval for mission-critical applications that require high levels of availability and performance. Failover clustering provides protection against planned and unplanned downtime [7]. Microsoft SQL Server clustering provides for two separate physical machines to be utilized for handling all incoming requests. If one server fails, all of its services are transferred to another machine without the calling applications being aware of the transfer [15].

2.3 Microsoft .Net Remoting

Microsoft .Net Remoting provides an abstraction layer to the developer so processes can be run on remote servers without significant involvement in this communication [8]. The .Net Remoting service provides the serialization (marshaling) of data prior to sending the request to the remote machine and then performs the deserialization at the remote server before the remote process begins its processing. The remote process will run solely at the remote server performing the logic of the remote object and utilizing the resources of the remote server.

2.4 Scalability

Scalability is the ability of a software application or software systems to be able to perform in expected ways with reasonable performance as the demands of the software and systems are increased. There are two main approaches to allow for applications and systems to scale.

Vertical scaling (or scaling up) requires the hardware components of the server(s) to be upgraded or improved. This can involve upgrades to processors, memory, disks, and network adapters [10] to reduce bottlenecks to the overall application as demands increase. Vertical scaling works best when the application is limited to run on the currently defined servers.

Horizontal scaling (or scaling out) requires additional servers to be added to the environment. This spreads the application and its processing across more resources. These improvements allow for growth to happen gradually and utilize less expensive servers [10]. Applications that can handle horizontal scaling also allow for these servers to be taken off-line during maintenance without impact to the system and help provide for mission-critical demands to keep the applications running during hardware failure.

3 Possible Solution

This paper is focused on a design to provide a cost effective solution to the many needs of mission-critical process intensive applications. These applications will generally be required to fulfill requests for the following conflicting needs:

- 1) Long running processes with significant data access that will utilize the available resources
- 2) Short running processes to fulfill a single request for the user that expects results immediately

3.1 Indiscriminate Scaling Solution

One way to fulfill these differing requests is to make them both function in the same way. Adjusting long running processes into many short running processes could make them both the same, but this would cause some negative side effects.

Indiscriminately forcing all processing to be short running processes will significantly increase the overhead of long-running processes that used to share the preparation work of accessing SQL data optimized for long running processes. Running all work as short running processes could significantly increase network bandwidth and increase context switching on each server. In addition, the long running processes need to provide a single aggregated response to the application. This proposed solution would not provide for this collation of data.

3.2 Process Scaling Architecture Solution

A better solution is to create a structure that can run both the long running processes and the short running processes in a way that best fits their needs. The short running process needs to be given priority over the long running process and available resources must be reserved for these requests. The

long running process needs to be able to intelligently break itself into smaller sub-sets in a structure that can provide a single logging mechanism and a pre-defined structure for collating results back to the main process.

The Process Scalability Architecture (PSA) solution provides this staged scaling solution that allows the long running processes and short running processes to both run to their differing needs. The main calling application will simply request for the process needed to be run and PSA will submit the process to be run on the best fit server and with the scheduling and priority as requested.

The staged scaling occurs within the long running processes as it can utilize the same process request structure as utilized by the main application to request sub-process to be run for the main process. These sub-processes, called worker threads, are pre-defined to be grouped together with the main process in terms of logging and result storage.

This grouping allows the worker threads to store their results in the same structure as the main process allowing the main process to collate this information and provide a single response back to the calling application. This grouping also allows for the main process and each worker thread to report any errors, warnings or informational data into a single logging location for the end user.

Figure 1. Process Queue Log

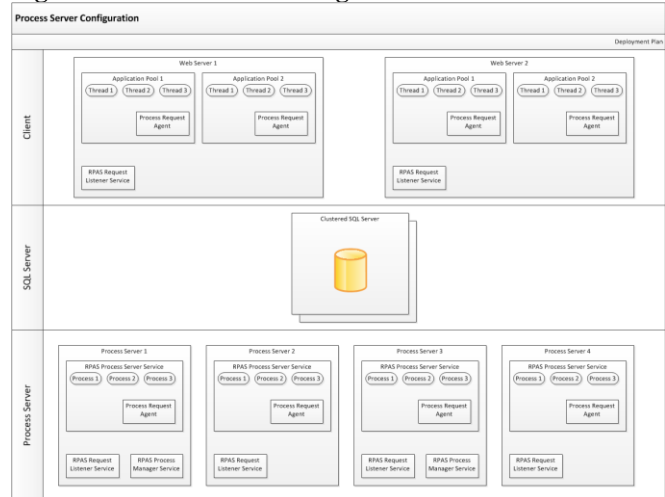
Log ID	Root Queue ID	Queue ID	Sponsor	Plan	Person ID	Severity	Time	Message
252817	8941	8941	ABC Company, Inc.	na	0	FYI	02/10/2011 11:40 AM	records to be processed: 0
252815	8941	8941	ABC Company, Inc.	na	0	FYI	02/10/2011 11:40 AM	records to be processed: 0
252811	8941	8941	ABC Company, Inc.	na	0	FYI	02/10/2011 11:40 AM	Create records based on Payments as of 2/10/2011

The logging mechanism, as is shown in *Figure 1*, will provide for all messages related to the main process to be displayed together (using the same Root Process Queue ID) but still allow the end user to see the details of which worker thread performed the process (based on Process Queue ID).

The Process Scalability Architecture is a complete solution that provides all of the services needed by mission-critical applications:

- 1) Horizontal scaling of processing
- 2) Load balancing to best fit servers
- 3) Higher prioritization of immediate requests
- 4) Scheduling to run processes at non-peak times
- 5) Consistent logging for viewing and monitoring

Figure 2. Architecture Configuration



3.3 PSA Configuration

As is shown in *Figure 2*, the solution involves three different physical groups of servers. The process server farm includes servers able to perform the processing needs of the application. The clustered SQL Server provides the data storage for process information and process server status. The Client is any software application utilizing the Process Request Agent to run a process. This could be a web farm, stand-alone application and the process servers themselves.

3.4 Process Server Service

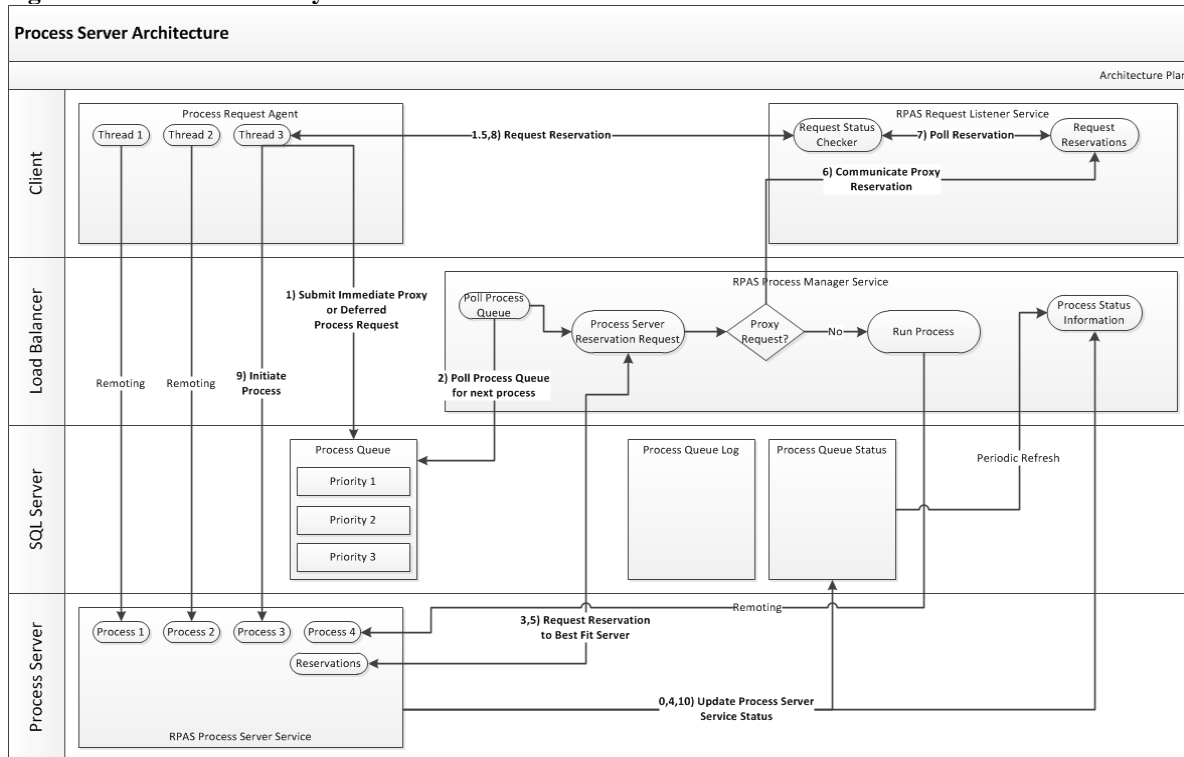
The Process Server Service will be deployed to each Process Server and will be responsible for listening for requests to run defined processes. The Process Server Service will be responsible for knowing which processes are allowed to run on the server and any restrictions on how many processes of each type can be run.

3.5 Process Manager Service

The Process Manager Service will be deployed to a subset of the available Process Servers and will be responsible for locating the best Process Server to utilize for any pending requests. The Process Manager will poll the SQL Server for both requests that are scheduled to run immediately and requests that are deferred.

3.6 Request Listener Service

The Request Listener Service will be deployed to all servers and will be responsible for communicating between the Process Manager and the application that has requested to run a process. This service is critical as it gives the Process Manager a single access point to each server even if the server is running multiple applications.

Figure 3. Process Scalability Architecture

3.7 Process Flow Overview

The Process Scalability Architecture, as shown in *Figure 3*, is separated into four logical groups: The Process Server, the SQL Server, the Load Balancer (Process Manager) and the Client. It is not represented in this diagram, but the Process Server can also play the role of the client – this is how it breaks its running process into smaller requests and scales to other process servers.

The diagram above is utilizing arrows to show all paths of communications between the entities. This communication is performed using .Net Remoting (for any communication between services) or transactions on the SQL Server (for all communication with the SQL Server).

This process flow will show the initialization of all servers, the initial request to run a process, the determination of where to run the process, and the completion of the process.

3.8 Initialization

As each service is started, it is responsible for registering itself with the SQL Server. During this registration process, it will record information about itself on the SQL server and collect information on all other servers available in the environment. This helps the service know which servers to attempt communication with, but the process flow will also handle the scenarios when new servers are added or when existing servers fail or are taken off-line.

3.9 Process Request Agent

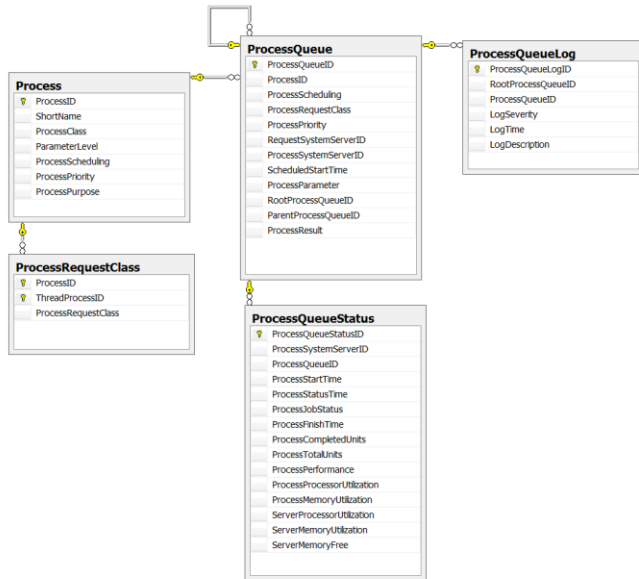
The Process Request Agent (PRA) is utilized by any client application to request a process to be run. The PRA is a middleware object that performs the necessary .Net Remoting calls to request a reservation on a process server and then runs the process for the calling application. All .Net Remoting calls are abstracted by using the Broker design pattern [6] that creates a client proxy and a server proxy for the methods utilized by the development team.

The Process Request Agent provides a single interface to allow the application to request a process to be run, define when it should be run and the parameters needed to run the process. With this information, the PRA will determine the priority of the process and submit this request to be run by recording the request in the SQL Server. If the process is scheduled to be run at a later time, then the PRA will let the application know that the request has been submitted. If the process is to be run immediately, the PRA must then remote to the Request Listener Service to await the determination on which process server to run the process.

3.10 Process Manager

The Process Manager (PM) manages both the queue of requested processes and the status and load of each process server. This queue is stored within the ProcessQueue table as shown in *Figure 4*. In combination with ProcessQueueStatus, it determines the available processes and their scheduled start time. The PM will periodically query these tables and sort the processes based on Priority and the original submission time.

Figure 4. Process Queue Storage



Once the Process Manager locates a process to be run, it will attempt to find the best-fit server to run the process. To do this, the PM must know which servers are online and available, which servers are able to run the process, the available resources on each server and what processes are currently running on each server.

These needs are handled by creating an open channel of communication between the Process Managers and Process Servers. The Process Server uses this channel of communication to update all Process Managers with its status and processing load when it starts, each time it picks up a new process and periodically while running each process. If the Process Managers or Process Servers cannot communicate with each other due to network issues or otherwise, the communication failure is logged but does not impact the processing. The Process Server status and load information is also stored in SQL tables, as is shown above, so the Process Managers can reset their status periodically or during initialization.

Of the available process servers that can run the process requested, the Process Manager will find the Process Server with the least activity and most available resources. This determination can be made by comparing the available resources of the server against the utilized resources. This paper will not detail the load balancing mechanism utilized, but the data required to make this determination is managed within the current solution.

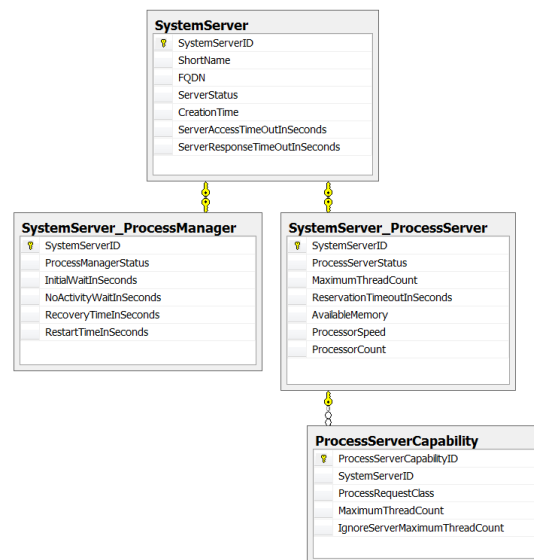
Once the Process Manager determines the best fit server, it will request a reservation from the Process Server. This reservation allows the Process Server to have the final determination of whether to run the process and allows time for the Process Manager to communicate back to the calling application. Once a reservation is returned from a Process

Server, the Process Manager communicates this back to the Request Listener Service which helps the Process Request Agent to run the requested process on the best fit Process Server.

3.11 Process Server

The Process Server's (PS) main role is to run the requested process. Along with that role is the responsibility to ensure that only the allowed processes are run on the server and within the allowed limits defined for the server. The Process Server makes this determination based on its configuration that is stored within SQL tables, as is shown in Figure 5.

Figure 5. Process Server Storage



The Process Server can be constrained to only pick up a certain number of each process type (defined by ProcessServerCapability) or can be defined to not accept any new processes once available resource levels are low. It can also reserve resources and available process capability for the short running processes so these can run at any time. As the Process Manager requests a reservation, the Process Server can still decline and the Process Manager will look to the next best server to run the process. If the reservation is granted, then the Process Server will reserve the resources or processing limits for a limited time before the reservation expires.

For each process run by the Process Server, the process can determine if and when it needs to allocate some of its work to other worker threads. Once determined, the sub-elements of the process are requested to be run by another Process Server in the same way that the main calling application requested a process to run – repeating the same process again. The only difference is that these worker threads will be linked to the

root process so all logging and activity is linked to the initial request.

While the Process Server is running the process from the calling application (or from another process), it will store all of its running information into the shared ProcessQueue, ProcessQueueStatus and ProcessQueueLog tables as discussed above. This unified storage location allows all logging information to be viewed together for the root Process requested by the end user. This allows the user to view each process's status and performance across all available servers.

3.12 High Availability

As the PSA is intended for mission-critical applications, it provides high availability. First, all elements of the architecture besides the calling application are built using Windows Services. This allows the process itself to run faster as it does not need to handle windows messaging that occurs for applications. It also allows the operational teams to configure the services to automatically start when the machine turns on and automatically restart any time the service fails due to SQL failures, etc.

Second, each of the Process Servers and Process Managers are built to handle the addition and deletion of servers without any impact to the end users' request to run a process. Each server will broadcast their status to all other servers in the environment each time they start or stop offering services. Additionally, the Process Manager attempts to maintain the status of all Process Servers in memory for faster service, but will periodically rebuild its in-memory data from the SQL.

Third, there is no single point of failure as the structure allows for multiple Process Servers and multiple Process Managers to be running at the same time. This allows the operational team to add and remove servers from deployment as needed. For any mission-critical application using SQL Server, the operational team will already have handled deploying the SQL Server as a clustered service with a possible second hot data center that will automatically become active on failure.

4 Validation

The implementation of this solution has shown very positive results. The end product provides a high available solution that can run short processes quickly, run long processes efficiently, and provide a standard mechanism to scale these long processes without any significant impact to the end user.

4.1 Case Example

Prior to this implementation, users were struggling with a process that would take 24 hours to run to completion (assuming no failure occurred). This timing was not sufficient for their needs, so they would need to manually split out the

processing into multiple submissions of lesser size. This would allow the process to complete in six hours but require an additional two hours of manual effort to split up the processing. In addition to the time required, the user could create issues by not splitting up the process correctly.

With this improvement, the user can submit the full process and the built-in scalability allows all of the initial data work to occur in the main process and allocation of 12 different worker threads to complete the process. This allows the whole process to complete in a little over 3 hours with no extra involvement of the end user. Even if there is a problem with one of the worker threads (by user error or system error), the user can then rerun a single worker thread and the logging of the rerun is included with the original process submission.

4.2 Validation By Bottlenecks

Another way to validate the success of the implementation is by reviewing the processor utilization on the Process Server before and after the implementation of this improvement. Prior to this implementation, the long running process and the manually created smaller processes would spend a significant amount of time waiting for responses from SQL requests. The queries were efficient with good execution plans, but there were a significant amount of requests. Due to this waiting time, the overall processor utilization would rarely go above 20% utilization. With these improvements, the utilization now peaks to 80%.

In addition, the SQL Server interaction increased so much with this improvement that the query and connection time out settings for the Process Server and the SQL Server had to be adjusted as the application was now making SQL Server work harder. This shows that the application and the software are no longer the bottleneck and operational improvements to hardware can now make improvements to the overall performance of the system.

4.3 Other Issues Identified

There are other possible issues in this implementation that should be reviewed when utilizing this solution.

First, remember that a 32-bit installation is still restricted to 1.5G for a single Process Server service. This means that deploying to 64-bit servers is the preferred deployment.

Second, this solution does not automatically protect against deadlocks from occurring. The Process Server Capability configuration must ensure that there are as many worker threads available as the number of main threads available. This will ensure that each main process will always have at least one worker thread to perform its work and eventually complete.

Third, starvation protection has not been built into this solution. This was not a significant enough issue for the initial implementation, but this can be added later by having the Process Manager increase the priority of all pending Process Queue records each time it does not have an available Process Server to run it.

5 Conclusions

This paper has defined a Process Scalability Architecture to be used in mission-critical applications with processor-intensive needs. This solution provides efficient solutions for common needs of these applications including: horizontal scaling, high availability, load balancing, scheduling, prioritization, consistent interfaces, and unified logging of all activity. The horizontal scaling mechanism also provides the ability for each process to also break out its processing to sub-processes on different servers. This solution has been implemented and validated against a mission-critical application and the identified issues have been corrected.

6 Acknowledgement

The authors would like to thank Craig Burma for his help and support during this work. They are also very appreciative of Ben Horstman, Daniel Wells, Mark Lansford and all of Seth's co-workers who helped to work through the issues related to this solution and to validate the design.

7 References

- [1] Ying Chen Lin, Sy-Yuan Li, Yuan-Shin Hwang. "Dynamic Load-Balancing of Jini and .NET Services," icppw, pp. 257-265, 2006 International Conference on Parallel Processing Workshops (ICPPW'06), 2006.
- [2] Erik Putrycz. "Design and Implementation of a portable and adaptable load balancing framework," pp. 238-252, Proceedings of the 2003 conference of the Centre for Advanced Studies on Collaborative research, 2003.
- [3] Rubén Mondéjar, Pedro García, Carles Pairet, Antonio F. Gómez Skarmeta. "Building a distributed AOP middleware for large scale systems," pp. 17-22, Proceedings of the 2008 workshop on Next generation aspect oriented middleware, 2008.
- [4] Simon Malkowski, Markus Hedwig, Calton Pu. "Experimental Evaluation of N-tier Systems: Observation and Analysis of Multi-Bottlenecks," pp. 118-127, Workload Characterization, 2009. IISWC 2009. IEEE International Symposium, 2009.
- [5] Yong-Cai Wang, Qian-Chuan Zhao, Da-Zhong Zheng. "Bottlenecks in Production Networks: An Overview", JSSSE 2005.
- [6] David Trowbridge, Dave Mancini, Dave Quick, Gregor Hohpe, James Newkirk, David Lavigne. "Distributed Systems Patterns", pp. 191-264, Enterprise Solution Patterns Using Microsoft .Net Version 2.0, Microsoft Corporation, 2003.
- [7] David Trowbridge, Dave Mancini, Dave Quick, Gregor Hohpe, James Newkirk, David Lavigne. "Performance and Reliability Patterns", pp. 311-336, Enterprise Solution Patterns Using Microsoft .Net Version 2.0, Microsoft Corporation, 2003.
- [8] Ingo Rammer. Advanced .Net Remoting, Springer-Verlag, New York, 2002.
- [9] Microsoft Corporation. A Guide to Building Enterprise Applications on the .NET Framework, <http://msdn.microsoft.com/en-us/library/ms954601.aspx>, 2003.
- [10] Microsoft Corporation. How to: Scale .NET Applications, <http://msdn.microsoft.com/en-us/library/ff650667.aspx>, May 2004.
- [11] Microsoft Corporation. SQL Service Broker, <http://msdn.microsoft.com/en-us/library/bb522893.aspx>, 2011.
- [12] IBM Software. WebSphere MQ: Features and benefits, <http://www-01.ibm.com/software/integration/wmq/features/>, 2011.
- [13] Microsoft Corporation. Microsoft System Center Essentials. <http://technet.microsoft.com/en-us/systemcenter/essentials/default.aspx>, November 2010.
- [14] Microsoft Corporation. Network Load Balancing Deployment Guide. [http://technet.microsoft.com/en-us/library/cc754833\(WS.10\).aspx](http://technet.microsoft.com/en-us/library/cc754833(WS.10).aspx), November 2010.
- [15] Microsoft Corporation. Microsoft SQL Server 2008: SQL Server 2008 Failover Clustering. <http://download.microsoft.com/download/6/9/D/69D1FEA7-5B42-437A-B3BA-A4AD13E34EF6/SQLServer2008FailoverCluster.docx>, June 2009.
- [16] Carol Bailey. MSCS vs. NLB: Evaluating the pros and cons, <http://www.techrepublic.com/article/mscs-vs-nlb-evaluating-the-pros-and-cons/1058353>, November 2002.
- [17] Michael Jones. Nine Tips to Enterprise-proof MSMQ, <http://www.devx.com/enterprise/Article/22314/1954>, 2010.
- [18] Microsoft Corporation. .Net Framework Conceptual Overview, <http://msdn.microsoft.com/en-us/library/zw4w595w.aspx>, 2011.

Rejuvenation Modeling in Safety Critical Real Time Systems using Stochastic Petri Nets

Bharati Sinha, Dr. Santanu K. Rath

Department of Computer Science and Engineering,
National Institute of Technology,
Rourkela-769008, India.

Abstract — *The main idea behind the increase in use of real time systems is that their performance is measured in terms of clock time and not on the basis of logical time which imposes timing constraints on the performance evaluation of such systems. It is a fact that any software with aging becomes prone to failure. Mostly the complex systems are safety critical systems whose reliability is the most important performance measure. Such systems need to be safeguarded from the effects of aging. This could be done either by debugging the errors or through preventive maintenance like rejuvenation. Although the downtime cost due to rejuvenation may make it appear expensive, but it enhances the reliability of the software. This paper studies the effect of rejuvenation on a real time system from the estimated values of performance measures based on SPNP tool.*

Keywords – Real time system; rejuvenation; stochastic petri nets; reliability; efficiency

1 Introduction

The safety and reliability of software systems could either be independent or related. When they are independent the system enters fail-safe state whenever it fails i.e. failure causes no damage. However when they are related, the systems become safety critical systems where failure can cause severe damage. Real time software systems generally have safety critical application where a single flaw or disruption could result in huge monetary loss; such applications need an almost perfect run where the chances of error are minimal even with continuous usage. A real time system has to consider real clock time in its functioning i.e., the behavior of the system is time dependent [1]. These systems are used in places where the performance of the system is analyzed based on actual quantitative time e.g., computerized air traffic control, where a delay in signal beyond permissible limit is equivalent to failure. The main factor in assessing performance is reliability rather than cost, associated during whole software life cycle.

The present day system failures are mostly due to software failure rather than hardware failure [2], which could be due to design anomalies, improper usage by users or deployment in changing environment. Such faults could be corrected either by debugging them when they occur or

through preventive measures. The former involves transient fault recovery which is corrective in nature but it results in incurring huge cost as the recovery starts only when fault occurs. It brings much overhead on time, cost and effort, so it needs to be reduced. The later method is preventive in nature which preemptively stops the continuously running application to avoid failure, called as rejuvenation [2]. When any system works continuously for a longer period of time, errors might creep in or it may require adaptive changes which require fault recovery. As real time systems are very sophisticated and have to deliver high performance; such periods of revival also needs to be considered for the overall system reliability estimation. Rejuvenation involves stopping the system in a periodic manner, freeing it from bugs that might have crept in and restart it from the initial state. This planned shutdown of system to prevent failure would be more economical in terms of cost to crash failure.

Rejuvenation could be best explained considering two basic concepts i.e., time based and prediction based rejuvenation [4]. Time based rejuvenation is considered when rejuvenation interval is of predetermined discrete value. Prediction based rejuvenation is done when the system enters a state of vulnerability where it is prone to failure and is rejuvenated.

The system could be modeled either as a discrete time Markov chain (DTMC) or continuous time Markov chain (CTMC) depending upon the case whether the system is absorbing or irreducible. Absorbing systems are those which have terminating execution. Irreducible systems are those which are continuously running applications. The system initially is considered fault free with all the modules working properly. The underlying Markov chain of the system has to be generated from the SPN model of the system for which reward function has to be incorporated in the structure of SPN Model. The SPNP tool [20] is used to define reward function after which it can be transformed to Markov model. The performance of the system may be analyzed in both steady as well as time variant states. This paper studies the effect of rejuvenation when it becomes vulnerable to estimate the reliability of the system. For reliability computation the probability of failure is calculated and from it, the reliability of the system is estimated.

The outline of the paper is as follows: section II describes the foundations of proposed methodology by explaining

basics of real time system, software reliability, need for rejuvenation and Stochastic Petri Net. Section III describes the experimental system where a system is considered initially with complete functioning to represent its functioning normally and then with rejuvenation interval. The reliability models with rejuvenation using various SPN models for different situation are discussed. Thereafter section IV describes the result which shows the effect of rejuvenation on reliability. Section V concludes the paper with scope for future work showing the effect of rejuvenation on reliability of the whole system.

2 Basic concepts

2.1 Real Time System

Real time systems are those in which system behavior is described by quantitative time; time being considered as clock time [1]. The performance of real time systems depends heavily on deadlines in terms of time. Real time systems have wide applications in digital and control systems. Based upon the type of task, real time system could be classified into hard, firm or soft real time system.

- Hard real time systems are those which have to perform within certain fixed time interval else they are considered failed e.g. robot.
- Firm real time systems also have fixed time for performance but the system does not fail in case of delay just the results are discarded e.g. video conferencing.
- Soft real time systems also have timing constraint but the time interval here is not an absolute value rather the average value which is why the systems efficiency lowers when more tasks have delayed completion time e.g. web browsing.

As time plays crucial role in the working of real time systems certain timing constraints are imposed on the system which could be classified as:

- Performance constraints: being imposed on the system in its response to the user.
- Behavioral constraints: being imposed on user's interaction with the system.

These two constraints behave differently based on the following criteria:

- Delay: This defines the minimum time difference required between two events.
- Deadline: This defines the maximum time interval which is permissible between two events.
- Duration: This defines period over which system must perform; it could be of minimum or maximum kind where either the system should have a longer duration than the specified minimum or shorter duration than the specified maximum.

Real time tasks are classified as:

- Periodic task: These tasks are repeated over fixed interval of time known as the period of the task. Periodic task may exist either from the instant of system initialization or during system application.
- Sporadic task: These tasks occur at random instants. Sporadic tasks can be denoted by

$$T_i = (e_i, g_i, d_i) \quad (1)$$

Where for any task T_i , e_i is its worst case execution time, g_i is the minimum time difference between two instances of T_i while d_i is relative deadline of T_i .

- Aperiodic task: Aperiodic tasks are similar to sporadic as they too can arise at random instant. The difference between them is that there need not be any time interval between two instances of an aperiodic task also the deadline is not an absolute value but average value or is expressed statistically.

Scheduling is the process of appropriate sequencing of jobs in order to get the optimal results. Real time scheduling represents the process of sequencing of real time tasks in proper order so as to meet the deadline. The time instants at which the scheduler has to decide upon the next task for execution are known as scheduling points. Schedulers are activated at these scheduling points to determine the next task. They can be classified on the basis of scheduling point as:

- Clock driven scheduling: In these schedulers the scheduling point is determined from clock interrupt.
- Event driven scheduling: In these schedulers the scheduling point is determined from occurrence of certain events.
- Hybrid scheduling: These schedulers use both clock interrupt and events to determine their scheduling points.

Another manner of classification is based upon how the tasks are selected for scheduling. The types of schedulers are

- Planning Based: This accepts the task only if it can meet its deadline and not interrupt the execution of other tasks.
- Best effort: This accepts the task without any test and makes a best effort to meet its deadline.

The other category of classification is based upon the kind of processor used and could be of following types:

- Uniprocessor based
- Multiprocessor based
- Distributed based

To meet the timing constraints of real time systems, tasks require efficient scheduling. The timing constraints can be modeled using stochastic petri nets.

2.2 Software rejuvenation

Software systems are being used in critical areas where study of reliability is quintessential for researchers as well as practitioners. However when any software is run for longer time, certain errors tend to creep in with time which is due to aging. Huang et.al [2] detected the effect of aging in telecommunication billing applications. Avritzer and Weyuker [3] reported aging in telecommunication switching software causing gradual performance degradation. The errors which are not debugged in the testing phase are mostly transient in nature whose root cause cannot be determined. They can be rectified either with a reactive approach or through preventive measures. In the reactive approach the error is rectified only when it occurs and might cause crashing of the system in critical applications. The preventive measure is to resurrect the fault before it occurs. Software rejuvenation was proposed by Huang et.al. [2] where the system is stopped for certain time interval, revived and then allowed to run again.

Various modes of rejuvenation have been proposed related to the criteria on which rejuvenation interval should be determined [10]. Huang et.al [2] considered the continuous time Markov chain (CTMC) to model system behavior for rejuvenation. They considered failure model in which the system could move from its initial state to a vulnerable one. It may either cause failure or it could undergo rejuvenation. The system could be restored in its initial state either after rejuvenation or after its recovery from failure. Garg et.al [4] introduced periodic rejuvenation and modeled the system using Markov regenerative stochastic petri nets to represent interval between successive rejuvenation of model. The fine grained software rejuvenation model [13] considers the degradation to be made of a sequence of failures that causes crashing when it exceeds threshold value.

Rejuvenation could be done in following ways depending upon the rejuvenation interval [6]:

- **Time Based Rejuvenation:** This is the simplest form of rejuvenation where the system is rejuvenated after fixed intervals of time. Here the system is stopped cleaned and restarted after fixed intervals of time irrespective of other consideration. This concept works well under normal circumstances but could cause severe problems if the system is executing some critical tasks.
- **Load and Time based Rejuvenation:** The system could be rejuvenated after its rejuvenation interval but only if it is free. This policy could be modified to consider the various conditions under which system cannot be rejuvenated even after the rejuvenation interval. As in case of clustered systems [14], where rejuvenation is either delayed in peak time or carried out during initial peak time but delayed later on.

In this paper the effect of rejuvenation is being considered. The system is put under rejuvenation when it enters a vulnerable state. It is modeled using Stochastic Petri Net (SPN) with SPNP tool [25] to estimate its reliability in case of time based and load and time based rejuvenation.

2.3 Software reliability

Software reliability gives an estimate of the probability of the system functioning properly under various circumstances. It is estimated by calculating the failure probability of the system, and then taking its complement. When a system is modeled using SPN, the reliability of the system is calculated by first estimating the failure probability due to various reasons and then summing them up for all the failure probabilities. Its complement is found out for estimating reliability.

The reliability of a system can be computed from the value of failure rate $\gamma(t)$, which is random and follows exponential distribution. The reliability of the system would be

$$R(t) = e^{-\gamma(t)} \quad (2)$$

While modeling the system with SPN the failure rate is modeled by associating firing times with each timed transition. When any software is identified to be fault free, the state associated with it is assigned a reward rate of value equal to "1". For reliability computation the expected instantaneous reward rate is to be computed [5]. The expected reward rate in steady state is given as:

$$E[X] = \sum_{k \in \tau} r_k \pi_k \quad (3)$$

Where τ is the set of tangible marking, π_k is steady state probability of tangible marking and r_k is the reward rate in marking k .

The expected instantaneous reward rate at time t is given as:

$$E[X(t)] = \sum_{k \in \tau} r_k \pi_k(t) \quad (4)$$

where $\pi_k(t)$ is the probability of marking k at time t .

2.4 Stochastic Petri Nets

Stochastic Petri Nets (SPN) is used for modeling performance of complex software systems [20]. They are a variant of Petri nets. They are represented as bipartite graphs, but have the time consideration which makes them suitable for performance and reliability modeling. Like Petri nets they also consist of sets of places and transition; but unlike simple petri nets the transition here could be timed or immediate. Any transition is enabled when there is at least one token in each of its normal set of input places and no token in any of its inhibitor places. The firing of enabled transition transfers resources from input places to output places depending on weights associated with the corresponding arcs. The transition in SPN could be of three types:

- Immediate with no time delay.
- Timed with exponentially distributed firing time.
- Timed with generally distributed firing time.

When the SPN consists of either immediate transition or timed transition with exponentially distributed firing time, it is said to be Markovian in nature where the future events depend only upon the present state and not on the past. It could then be used to generate underlying Markov chain and solve it for computing various performance measures [21]. However if it consists of even one timed transition with

generally distributed firing time, the Markov property does not hold well. To solve this problem certain instances of time are needed where the past state may not be considered. These instances are known as regeneration points where the future events depend only on the present state.

Timed transitions are used to signify events which are time consuming where the enabling of transition denotes the beginning of the activity while the firing of the transition denotes its completion. Immediate transitions are used to represent instantaneous events which occur with no delay. Immediate transitions thus have a sort of precedence over timed transitions. When a system is modeled as SPN, its various performance characteristics can be obtained as a time average limit of markings of the net [22]. SPN is suitable for modeling applications which involve randomness and probability as well as time consideration. So these are efficient for modeling real time applications. There are various functionalities [21] in stochastic petri nets which make them more suitable for real time system modeling:

- Variable cardinality arc: The cardinality of an arc could be varied so as to denote the minimum number of tokens to be present in input places for a transition to be enabled.
- Priorities: The priority of transition could be defined to establish priority relationship between them the one having higher priority will be preferred to that with lower priority.
- Guard: A transition could have an associated guard function. It is evaluated only if there is possibility of the transition being enabled in that marking. Then transition is enabled only if the guard function is evaluated to be true. It is used for representing constraints which are difficult to be represented graphically in terms of input, inhibitor and output arcs.

The stochastic process is calculated by a marking process $\{M(t), t > 0\}$ which is obtained by constructing the reachability graph of the net [23]. First of all the reachability set has to be determined i.e., the set of possible future states in the system. Then from the initial marking M_0 , reachability graph can be constructed by connecting arcs when a transition enabled in one marking reaches another marking. Even if one immediate transition is enabled the marking is said to be vanishing otherwise tangible.

3 Rejuvenation model

To consider the effect of rejuvenation on reliability an example of safety critical system i.e. Air traffic control (ATC) system is being considered. The various activities related to aircraft functioning are modeled using SPN. The reliability of ATC is computed first without considering the case of rejuvenation and then subsequently the case of rejuvenation. Table 1 gives the description of places in the model and Table 2 gives the description of each transition.

Table I. Place description

Place	Description
P0	Aircraft
P1	ATC
P2	Source ground controller
P3	Source local controller
P4	Departure controller
P5	Radio controller
P6	Approach controller
P7	Destination local controller
P8	Destination ground controller
P9	System down
P10	System rejuvenation

Table II. Transition description

Transitions	Description
T0	Flight plan sent
T1	Secondary clearance issued
T2	Flight progress strip(FPS) sent to ground controller
T3	Undock
T4	Runway assigned
T5	Local controller called
T6	Taxi Aircraft
T7	Clearance issued
T8	Transponder signal sent
T9	Tracked on radar
T10	Radio controller called
T11	Radio controller contacted
T12	Radio controller updated about positions
T13	FPS sent
T14	FPS transferred
T15	Request to land
T16	Parameters adjusted
T17	Local controller called
T18	Clearance awaited
T19	Clearance issued
T20	Ground controller called
T21	Runway requested
T22	Runway assigned
T23	Aircraft landing
T24	Taxi and dock aircraft
T25	Vulnerability in flight plan
T26	Failure on rejuvenation
T27	Success on rejuvenation

In Figure 1 Air Traffic Control system(ATC) is modeled where initially the mechanically checked aircraft sends it flight plan to the air traffic controller. They on verifying the flight plan issues secondary clearance to flight and sends Flight Progress Strip(FPS) to ground controller. The aircraft then contacts ground controller for undocking

who assigns specific runway upon request. The ground controller calls local controller. Aircraft requests for taxi to local controller and when clearance is issued taxis on runway for passengers to arrive. Aircraft sends transponder signal to departure controller who tracks it on radar. Departure controller calls the radio controller. Aircraft then contacts the radio controller who regularly updates the flight position as tracked on radar. Accordingly flight sends FPS to Radio when it is approaching the destination. The radio controller sends updated FPS to flight. Aircraft then requests approach controller for landing. Approach controller sends the adjusted parameters to flight and calls local controller. Aircraft request local controller for clearance of landing, when granted local controller calls ground controller. Aircraft requests ground controller for runway when runway is assigned it lands. The local controller then asks the aircraft for taxi and docking. However if the flight plan is faulty aircraft may become vulnerable to failure and move to down state.

In Figure 2 rejuvenation is considered once the system become vulnerable the system on rejuvenation may come to its initial state or to down state. The two models are simulated to estimate system reliability. A flawless flight is considered to be one where the aircraft taxis on the destination runway safely. First the simulation is done for the model in Figure 1 and its reliability is computed. Then simulation is done on the model in Figure 2 and its reliability is estimated. The results show the reliability estimates in both the models. In these models only one reason of failure has been considered when the flight plan is not right, the system may become vulnerable and fail moving to down state. If rejuvenation is considered a vulnerable system upon rejuvenation has greater chances of being corrected and moved to its initial state or may even then fail. Rejuvenation is considered to improve system reliability.

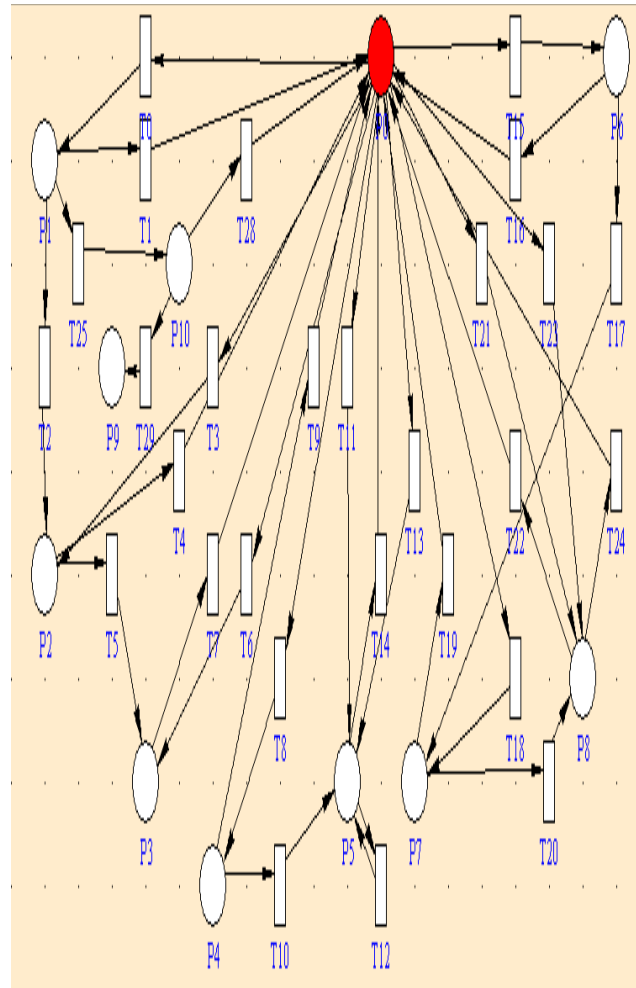


Figure 2.ATC model considering rejuvenation

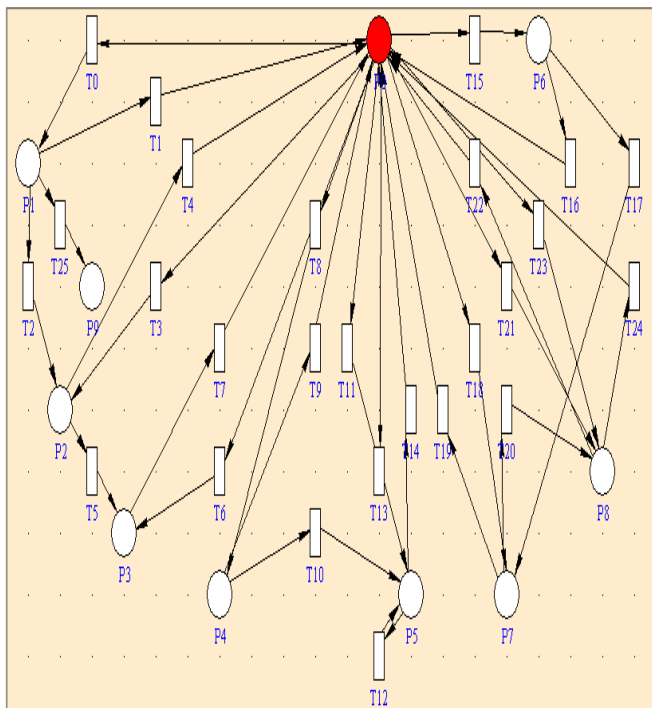


Figure 1.Model for reliability estimation of ATC

4 Results

The graph in Fig. 3 shows variation in reliability over time. The system is said to have successfully completed flight if it lands at the destination airport with permission from the ground controller i.e. T24 is enabled. This is possible only when the flight plan is correct otherwise the system enters a state of vulnerability and fails, moving to down state. When rejuvenation is considered, the vulnerable system may move to its initial state rather than moving to down state which is why reliability is improved when rejuvenation is considered in system modeling. Reliability of the system in both cases is compared. Table III indicates the variation in reliability values, under both the circumstances.

Table III. Result comparison

Sl.no	Time	Reliability value without rejuvenation	Reliability value with rejuvenation
1	1	0.160454904396	0.162636016104
2	10	0.14499093067	0.153886816717
3	20	0.142916906879	0.15139063949
4	30	0.142821837245	0.151310725531
5	40	0.142804328654	0.151308373726
6	50	0.142789410231	0.151308304329
7	60	0.142774578066	0.151308301708
8	70	0.142759750211	0.151308301331
9	80	0.142744923987	0.151308301301
10	90	0.142730099305	0.151308301301
11	100	0.142715276163	0.151308301325

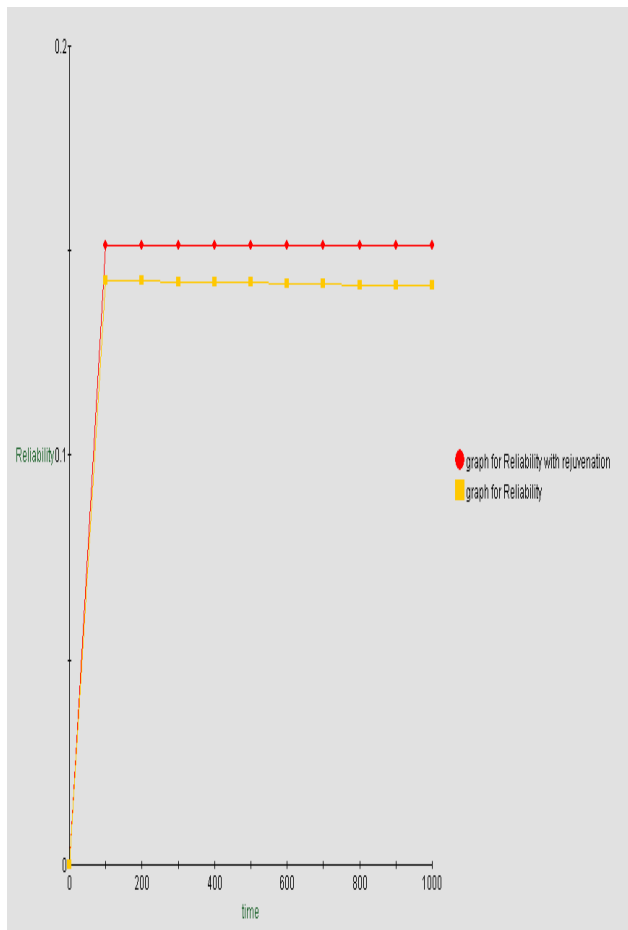


Figure 3. System Reliability

5 Conclusion

Research indicates reliability estimation is a core area of software performance engineering and for safety critical real time application; it becomes even important as stakes are very high. Any software with aging needs to be revived periodically to satisfy the newer requirements in order to make the system compatible. Hence the effect of rejuvenation on reliability is of prime importance. Here the results show that the reliability of the system improves when system is rejuvenated after which it tends to assume a constant rate. These results are from preliminary research and need to be probed further.

These models consider a single reason for failure but more accurate estimates could be obtained if other factors are considered. These models could be extended to include sensitivity analysis for fault tolerance and for modeling more complex systems considering dependencies between the modules of the system.

References

- [1] Jane W.S.Liu, *Real-Time Systems*, Pearson Publication 2006.
- [2] Huang, C.Kintala, N.Kolettis and N.D.Funton, "Software Rejuvenation: Analysis, Module and Application", In Proc. Of IEEE Int'l Symp.on Fault Tolerant Computing, IEEE Computer Society Press, 1995, pp.381-390.
- [3] A.Avrizter and E.J.Weyuker, "Monitoring Smoothly Degrading Systems for Increased Dependability", *Empirical Software Engineering*, 1997, pp.59-77.
- [4] S.Garg, A.Puliafito, M.Telek and K.S.Trivedi, "Analysis of Software Rejuvenation using Markov Regenerative Stochastic Petri Net", In Proc.- Int'l Symp. On Software Reliability Eng., 1995, pp.24-27.
- [5] S.Garg, Y.Huang, C.Kintala, and K.Trivedi, "Time and Load Based Software Rejuvenation: Policy, Evaluation and Optimality", In Proc.- 1st Fault-Tolerant Symp., 1995, pp. 22-25.
- [6] S.Garg, A.Puliafito, M.Telek and K.S.Trivedi "On the Analysis of Software Rejuvenation Policies", In Proc. Annual Conference on Computer Assurance (COMPASS), June 1997.
- [7] Vibhu Saujanya Sharma and Kishore.S.Trivedi "Quantifying software performance reliability and security" In: Elsevier 2006 , pp.493-508.
- [8] Jogesh.K.Muppala "Stochastic reward net for reliability prediction" In: research paper sponsored by National Science Foundation.
- [9] M. Grottke and K. S. Trivedi "Fighting Bugs: Remove, Retry, Replicate, and Rejuvenate ", *IEEE Computer*, , 2007, pp. 107-109.
- [10] T. Dohi., K. Goseva-Popstojanova and K. S. Trivedi , "Analysis of Software Cost Models with Rejuvenation", In Proc.- IEEE Intl. Symposium on High Assurance Systems Engineering, November 2000.
- [11] R.Chillarege, S.Biyani and J.Rosenthal, "Measurement of failure rate in commercial software" In Proc. Of 25th symposium on fault tolerant computing , June 1995.
- [12] S. Garg, A. Puliafito, M. Telek, K.S. Trivedi, "Analysis of preventive maintenance in transactions based software systems", In *IEEE Transactions on Computers*, 1998, pp. 96-107.
- [13] A.Bobbio, and M.Sereno, "Fine Grained Software Rejuvenation Models", Proc.3rd IEEE Int'l Computer Performance & Dependability Symp., IEEE Computer Society Press, 1998, pp. 4-12.

- [14] D.Wang, W.Xie and K.S.Trivedi "Performability analysis of clustered system with rejuvenation under varying workload", In:Elsevier,2006, pp.247-265.
- [15] M.Grottke, L.Li, K.Vaidyanathan and K.S.Trivedi "Analysis of software aging in a web server" In:IEEE transactions on reliability September 2006, pp.411-420.
- [16] Rivalino Matias, Jr., Paulo J. F. Filho, K. S. Trivedi, and Pedro A. Barbetta "Accelerated Degradation Tests Applied to Software Aging Experiments", IEEE Transactions on Reliability, Vol. 59, Issue 1, March 2010, pp. 102-114,.
- [17] K. Vaidyanathan, R. E. Harper, S. W. Hunter and K. S. Trivedi. "Analysis of Software Rejuvenation in Cluster Systems", In Proc. of the Joint Intl. Conference on Measurement and Modeling of Computer Systems, ACM sigmetrics 2001/performance 2001, Cambdrige, Massachusetts,June 2001.
- [18] KalyanramanVaidyanathan,Richard.E.Harper,Steven.W.Hunterandad, Kishore.S.Trivedi, "Analysis and implementation of software rejuvenation in cluster systems"Research paper supported by IBM.
- [19] A.Avritzer, A.Bondi, M.Grottke, K.S.Trivedi and E.J.Weyukur "Performance assurance via software rejuvenation: monitoring statistics and algorithm", In proceedings of International Conference on Dependable Systems and Network,2006 , pp.435-444.
- [20] M.Marson,.Balbo, G.Conte, S.Donatelli, G.Franceschinis,"Modelling with Generalised Stochastic Petri Nets".
- [21] V.G. Kulkarni, *Modeling and Analysis of Stochastic Systems*, Chapman & Hall, 1995.
- [22] Peter.J.Haas, *Stochastic Petri Nets Modelling,Stability,Simulation*, Springer, 2002.
- [23] M.Ajmone.Marsan,S.Donatelli and F.Neri, *GSPN model on multi servermultiqueue system*, In:IEEE Computer society press,December 1989,pp.19-28.
- [24] G.Chiola "A software package for the analysis of generalised stochastic petri net models" In Proc.of international workshop of timed petri nets,July 1985, pp.136-143.
- [25] G.Ciardo,J.Muppala, K.S.Trivedi, "Stochastic petri net package" In Proc.of international workshop on petri net and performance models", December 1989, pp.142-150.

Efficient Design Pattern Selection and teaching by BPL technique.

S.Sarika
Research Scholar
Computer Science & Engg
Sathyabama University
Chennai.
sarish_sar1@yahoo.co.in

Dr.T. SasiPraba
Dean-Publications & Conferences
Sathyabama University
Chennai.

Abstract:

Design patterns are a proven way to build high-quality software. A pattern is a named abstraction from a concrete form that represents a recurring solution to a particular problem. The number of design patterns is rising rapidly, while training and searching facilities seems not to catch up. Code reuse has attracted lot of attention but design patterns are knowledge reuse. In this paper we discuss several problems in teaching design patterns and also an experiment prototype to adopt design patterns with the help of users' previous history. The effective training can be achieved by using PBL approach. Software agents will be used to extract information from the WWW. These services will be given as web service to the user for their convenience in learning and adopting design patterns.

Key words: *Design patterns, design pattern adoption, PBL approach.*

I. INTRODUCTION

In all engineering disciplines it is important to develop new systems from existing, already proven reusable elements. This will enable the engineers to use well known approaches and best practices. Reuse has become an essential and important strategy and software development area is not different. The reuse of concrete software elements such as functions, classes and components is already well established and practiced on a daily basis. But if we observe reuse at higher levels of abstraction i.e. software patterns, the reuse is still not well established. Patterns are not code reuse instead, solution/strategy reuse and sometimes, interface reuse. The field of software design patterns has seen an explosion in interest since 1995 with the publishing of the book "Design Patterns: Elements of Reusable Object- Oriented Software", written by GoF. Nowadays, design patterns are widely used in software development and have become a

fundamental skill that is required for all Software Engineering professionals. Design Patterns is generally regarded as a difficult area to learn as well as teach because it is too abstract to comprehend. A survey conducted by the MS Patterns and Practice Group indicated a low adoption of design patterns among practitioners – respondents estimated that no more than half of the developers and architects in their organization use software patterns. Therefore bridging the gap between the pattern expert communities and the typical pattern user is critical for achieving the full benefits of software patterns. Several hundred software patterns have already been published. With the increasing size of software system, the understanding and maintenance is more difficult and time consuming. And the development document does not exist, or there is no up to date. The source code has become the only resource for understanding software system which is a structured organization of the source code. A design pattern needs to be instantiated before its use. To be able to instantiate the pattern and to attain its intended benefits, designers are expected to have a good understanding and experience with design patterns, which is not evident to acquire [10].

Design patterns are too abstract; they are difficult for students to understand [1]. If we use conventional spoon-feed teaching strategy in the teaching process of this subject, the knowledge of design patterns will become fragile knowledge, and will not help to form real application capability for students. As we mentioned above, it is an urgent and tough task for us to give full considerations of the characteristics of students, to choose appropriate textbook, and to adopt proper teaching strategy to improve the teaching effect of the software design patterns subject. This paper concentrates on the discussion of the teaching strategy for software design patterns subject. We

adopt problem-based learning (PBL) teaching strategy with a variation in our teaching process.

II. SOFTWARE DESINGS PATTERN

Design patterns reside in the domain of modules and interconnections. At a higher level there are architectural patterns that are larger in scope, usually describing an overall pattern followed by an entire system.

Algorithm strategy patterns addressing concerns related to high-level strategies describing how to exploit application characteristic on a computing platform Computational design patterns addressing concerns related to key computation identification.

Execution patterns that address concerns related to supporting application, including strategies in executing streams of tasks and building blocks to support task synchronization.

Implementation strategy patterns addressing concerns related to implementing source code to support program organization, and the common data structures specific to parallel programming.

Structural design patterns addressing concerns related to high-level structures of applications being developed.

III. INTERACTION DESIGN PATTERN

Interaction design patterns are a way to describe solutions to common usability or accessibility problems in a specific context.^[1] They document interaction models that make it easier for users to understand an interface and accomplish their tasks

Elements of an interaction design pattern

For patterns to be helpful to the designers and developers who will make use of them, they need to be findable and readable.

Common Elements

Though pattern descriptions vary somewhat, many pattern libraries include some common elements:

Pattern Name: Choosing a clear and descriptive name helps people find the pattern and encourages clear communication between team members during design discussions.

Pattern Description: Because short names like "one-window drilldown" are sometimes not sufficient to describe the pattern, a few additional lines of explanation (or a canonical screenshot) will help explain how the pattern works.

Problem Statement: Written in user-centered language, this communicates what the user wants to achieve or what the challenge is to the end-user.

Use When: "Context of use" is a critical component of the design pattern. This element helps people understand situations when the design pattern applies (and when it does not.)

Solution: The solution should explain "how" to solve the problem, and may include prescriptive checklists, screenshots, or even short videos demonstrating the pattern in action.

Rationale: Providing reasons "why" the pattern works will reinforce the solution, though time-pressed developers may prefer to ignore this explanation.

Optional Elements

Pattern libraries can also include optional elements, depending on the needs of the team using them. These may include:

Implementation Specifications: A style guide with detailed information about font sizes, pixel dimensions, colors, and wording for messages and labels can be helpful for developers.

Usability Research: Any supporting research from usability tests or other user feedback should be captured. This can also include feedback from developers, customer service, or the sales team

Related Patterns: The pattern library may include similar patterns, or it may be organized into a hierarchy with parent and child patterns.

Similar Approaches: Since there are likely to be many possible solutions to this problem, teams may want a place to capture similar alternatives.

Source Code: If the code is modular enough to be reused, then it can be included in the library as well

IV. RELATED WORK

Since 1995, when design patterns are introduced there are lot of works done to improve the design patterns adoption. In general there are three main categories of descriptions: informal representations, semiformal representations based on graphical notations such as UML and various formal representations which also include notations using semantic web technologies. There are plenty of attempt made to ease the selection of design patterns. Still Design Patterns are selected by the architects with the help of their expert knowledge manually.

Design patterns are well explained in the book “Design Patterns: Elements of Reusable Object-Oriented Software”, which is respected as the Bible of object-oriented designers. Many other books related to software design patterns have the same kinds of problem. Some of them have made explanation of each kind of design patterns, thus are easier to understand, but still lack good teaching cases. They are more suitable for reference materials than textbooks. Design patterns are too abstract and so it is difficult for the beginners to understand.

V. PROBLEM BASED LEARNING APPROACH

Problem-based learning (PBL) as a teaching strategy and curricular design was pioneered at Case Western Reserve University in the early 1950s and in the 1960s by Howard Barrows at McMaster University in Canada. It is a deep “learn by doing” approach and is conducted successfully in the medical discipline. Kaufman asserts that “PBL has been one of the most successful innovations in medical education and has established its credibility.” PBL is a good teaching approach that was verified by many teachers of other disciplines

around the world. The fundamental principle supporting the concept of PBL is that the problem is the driving force that initiates the learning. This way of learning encourages a deeper understanding of the material rather than superficial coverage.

The Six core characteristics of problem based learning are [1]:

1. Consists of student-centred learning
2. Learning occurs in small groups
3. Teachers act as facilitators or guides (referred to as tutors)
4. A problem forms the basis for organized focus and stimulus for learning
5. Problems stimulate the development and use of problem solving skills
6. New knowledge is obtained through means of self-directed learning

Within the PBL environment the problem acts as the catalyst that initiates the learning process.

VI. IMPLEMENTATION WITH HELP OF SOFTWARE AGENT

A software agent is a piece of software that acts for a user or other program in a relationship of agency, which derives from the Latin *agere* (to do): an agreement to act on one's behalf. Such "action on behalf of" implies the authority to decide which (and if) action is appropriate

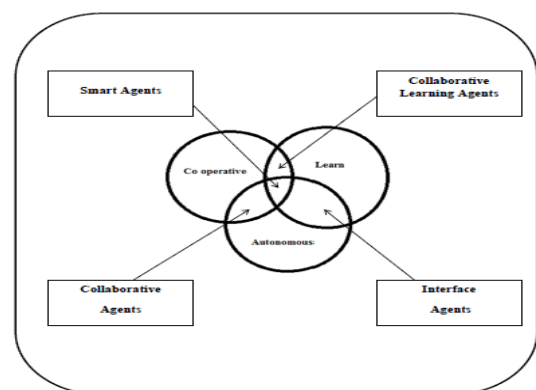


Fig 1: Agent Interaction with Environment

Agents are not strictly invoked for a task, but activate themselves.

Agents may reside in wait status on a host, perceiving context.

Agents may get to run status on a host upon starting conditions.

Agents do not require interaction of user

Agents do may invoke other tasks including communication

Design issues

Interesting issues to consider in the development of agent-based systems include:

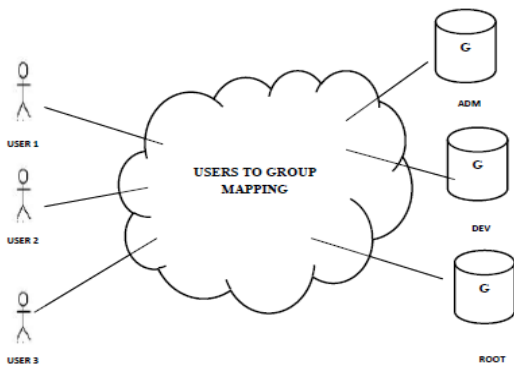


Fig 2: Agent Interaction with Environment

How tasks are scheduled and how synchronization of tasks is achieved.

How tasks are prioritized by agents.

How agents can collaborate, or recruit resources.

How agents can be re-instantiated in different environments, and how their internal state can be stored.

How the environment will be probed and how a change of environment leads to behavioral changes of the agents

Functionalities and design

The design patterns adoption and teaching can be improved with the following three modules.

1. Search module
2. Proposing module (Recommendation approach)
3. Training module (PBL approach)
4. User and data maintenance module.

Search Module

Search module is a fully indexed text based search service which will give details about 23 Design patterns given in GoF and this is open to the other open source patterns too. This will be a user friendly text search. All the users can be allowed to consume this service. The search will be done in a repository where the patterns will be present with pattern name, wheretouse, benefits, and related patterns, example code. Search can be implemented using ORM (Object Relation mapping) which will be more efficient.

Training Module

Training module is a help service which will contain real time examples with diagrams to teach design patterns. This module will also have a chat application which will enable PBL based design patterns learning. The chat application enables the novice programmers to communicate with the experts present online. This will eliminate the abstract of design pattern concepts. Also this service will contain the detailed information about the design pattern which includes Pattern name and classification, intent, Also known as, Motivation, Applicability, Structure, Participants, Collaborations, Consequences, Implementation, sample code, Known uses, Related patterns.

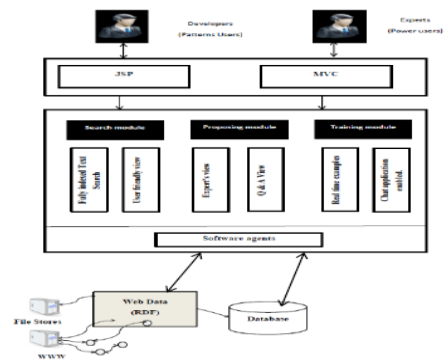


Fig 3: System Architecture

Proposing Module

Proposing module is a proposing service which will help in selecting patterns for the application. This will suggest the design patterns to the user depending upon the scenario given by the user. The user scenario will be searched in the repository and the results will be the design patterns' name along with the rank given to the design pattern by the users. This suggestion is done by using modified Page Ranking algorithm. If the users' scenario is not found in the repository then a notification will be sent to the expert user group. There is provision given to the experts to add the appropriate design pattern for the requested user scenario. Once the requested design patterns' detail is added in the repository then an email notification will be given to the user regarding the update.

Modified Page Ranking algorithm

PageRank is a probability distribution used to represent the likelihood that a person randomly clicking on links will arrive at any particular page. But here we slightly modified the algorithm to suggest the design patterns for the novice programmers. The algorithm will be as follows:

Initially all the Design Patterns will be given a rank of 1.

$$R(DP_i) = 1 \text{ for } i=1, 2, 3...23$$

The Rank of the design patterns will be increased by the number of positive feedback given by the user on the basis of relevance they encounter while searching with the scenario. Every time the user gets satisfied with the result suggested then the rank of the particular design pattern (R_n) will be incremented by 1.

$$R_n(DP) = R(DP) + 1$$

The rank of the design patterns will be considered when suggestion is made. The design pattern with the highest rank among suggested list of result will be shown in the top of the suggestion.

User and data maintenance Module

This is the module which is responsible for maintain responsible for maintaining the user and data repository. Here repositories will be maintained to hold the information about design patterns. This module also describes the authentication for the users who are all consuming this web service. The admin will usually add and remove the design pattern related information in the repository.

VII RESULTS AND DISCUSSION

The search module will be used by the users

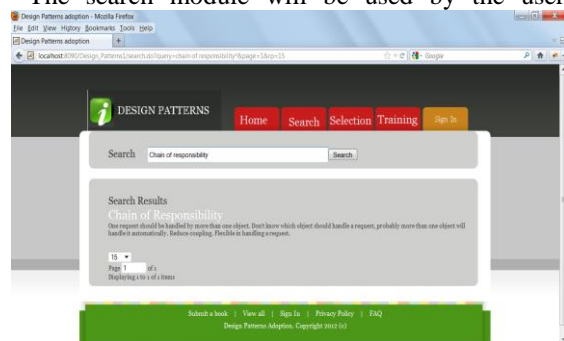
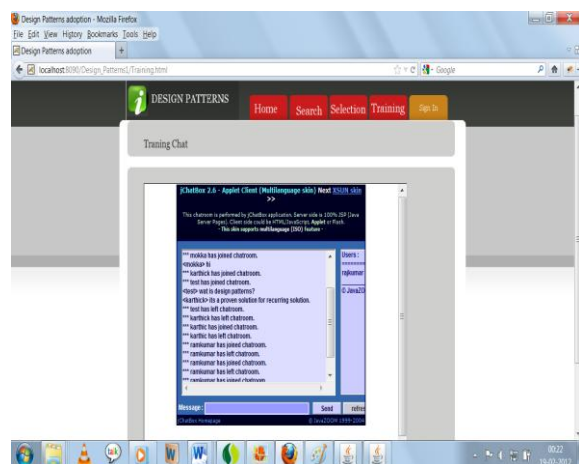


Fig: 2: Search Module showing search results search details about the standard design patterns. The proposing module will be suggesting the design patterns depend upon the scenario given. The accuracy of the suggestion will increase as the web service is used more.



The PBL based training will be done through the chat application where chat room will be created for discussing exclusively about the design patterns. This is entirely different from the traditional method of learning and will be more useful to the new programmers.

VIII FUTURE WORK

It would be reasonable to develop additional component that would enable user to verify if particular candidate was well selected. Service of that kind might help developers capable of selecting design patterns on their own to verify their selections. The design patterns can be documented into formal representations so that it can be processed automatically by the system in the future. Plug-in can be given to the user in their development IDE itself. Plug-in architecture is a well-known which got lot of attention by the developers across the world. Plug-in will be developed by invoking the above said services for Eclipse IDE to make sure the design patterns adoption and learning tool is readily available to the novice programmers in their hands. The plug-in will be helpful to programmers in learning the design patterns with the help of real world examples and also allow the users to interact with the experts from their IDE itself.

IX CONCLUSION

This will reduce the gap between the software developers and the usage of design patterns. The search and finding the design patterns for the software development will be readily available to the novice programmer. The PBL method is really a deeper learning method as it says "learn by doing". So the power of Design patterns can be utilized to a greater extend.

REFERENCES

[1] Barrows, H.S. (1996). Problem-based learning in medicine and beyond: A brief overview. In Wilkerson, L & Gijsselaers, W.H. (eds). New directions for teaching and learning, no.68. Bringing problem-based learning to higher education: Theory and practice, 3-13. San Francisco: Jossey –Bass

[2] Core J2EE Patterns, <http://java.sun.com/blueprints/corej2eepatterns>.

[3] Improving Design Pattern Adoption with Ontology-Based Design Pattern Repository, Luka Pavli_, Marjan Heri_ko, Vili Podgorelec University of Maribor, Faculty of Electrical Engineering and Computer Science Smetanova 17, SI-2000 Maribor, Slovenia {luka.pavlic, marjan.hericko, vili.podgorelec}@uni-mb.si

[4] L. Rising, "Understanding the Power of Abstraction in Patterns", IEEE Software, July/August 2007, Vol. 24, No. 4., pp. 46-51.

[5] T. Jenkins, "A participative approach to teaching programming", Proceedings of the 6th annual conference on the teaching of computing and the 3rd annual conference on Integrating technology into computer science education: Changing the delivery of computer science education, ACM New York, NY, US, 1998, pp.125-129.

[6] Woods, D. R., Problem-based Learning: how to gain the most from PBL, Waterdown, Ontario, 1996.

[7] Ignacio Navarro, "A recommendation system to support design pattern selection", IEEE symposium on visual Language and Human-centric Computing, 2010.

[8] Niklas Pettersson, Welf Lowe, and Joakim Nivre, "Evaluation of Accuracy in Design Pattern Occurrence Detection" IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 36, NO. 4, JULY/AUGUST 2010.

[9] Cheng Zhang and David Budgen, Member, IEEE, "What do we know about the effectiveness of Software Design Patterns?" IEEE TRANSACTIONS ON SOFTWARE ENGINEERING.

[10] Nadia Bouassida , Afef Kouas, Hanene Ben-Abdallah, "A Design Pattern Recommendation Approach", IEEE TRANSACTIONS, 2011.

[11] B. Huston, "The effects of design pattern application on metric scores," *Journal of Systems and Software*, vol. 58, pp. 261–269, Sept. 2001.

[12] K. Schelfhout, T. Coninx, A. Helleboogh, T. Holvoet, E. Steegmans, and D. Weyns. Agent implementation patterns. In *Proceedings of the OOPSLA 2002 Workshop on Agent-Oriented Methodologies*.

[13] A. Blewitt, A. Bundy, and I. Stark, "Automatic verification of Java design patterns," *Automated Software Engineering*, 2001. (ASE 2001). *Proceedings. 16th Annual International Conference on*, pp. 324–327, 26–29, Nov. 2001

[14] Y. Aridor and D. Lange. Agent design patterns: Elements of agent application design. pages 108.115. ACM, 1998.

[15] E. Billard. Patterns of agent interaction scenarios as use case maps. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 34(4):1933.1939, 2004.

[16] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal. *Pattern-oriented software architecture: a system of patterns, volume 1*. Wiley India Pvt. Ltd., 2008.

[17] M. Cossentino, P. Burrafato, S. Lombardo, and L. Sabatucci. Introducing pattern reuse in the design of multi-agent systems. *Agent Technologies, Infrastructures, Tools, and Applications for E-Services*, pages 107.120, 2002.

[18] A. Drogoul, T. Chu, E. Amouroux, V. An, and N. Doanh. *Agent based simulation: definition, applications and perspectives*. 2008.

[19] J. Ferber. *Multi-agent systems: an introduction to distributed artificial intelligence*. Addison-Wesley, 1999.

Application of Petri Nets to XPDL Workflow Optimization

L. Osuszek

Institute of Informatica, Silesian University, Sosnowiec, Poland

Abstract - This paper describes the conversion of XPDL workflows into Petri Network Modeling Notation. The result of such operation is analyzed for time consumption optimization. This shows how performing such operation results in tangible savings and economical benefits.

Keywords: Business Process, Workflow, XPDL, BPM, Petri Network, PNML, Petri Network Modeling Notation, optimization, P8 Business Process Management, P8 BPM

1 Introduction

This article presents the potential of the Petri Net model used for the optimization of business processes. There are several alternatives to model business processes. One modeling alternative is the Business Process Modeling Notation (BPMN).

For each company and organization, business processes and the related decisions are the key element which provides the momentum for their operations. The management of workflow and information within process paths has a major impact on the speed, flexibility and quality of decision-making processes. This is why the acceleration and optimization of processes is decisive for the success of any organization.

Processes involve people, systems and information. The maximum efficiency is possible only if all of these elements interoperate in an automated environment. Note also that optimized processes enable a faster response to the changing market situation and to new customers' demands while guaranteeing compliance with applicable regulations. In short, better processes contribute towards continuous improvement of the efficiency of company's operations, and therefore, allow gaining competitive advantage in the industry. One of the factors which make these tools increasingly popular is the tracking, analysis and simulation of processes. With the monitoring of work progress, with in-depth analysis of current and historical processes, and with the verification of changes to processes prior to their implementation in a production system, these functionalities guarantee more accurate business decisions. Additionally, they enable fast implementation of best business practices, guarantee unified processing, and reduce the total cost of system ownership with reusable process definitions.

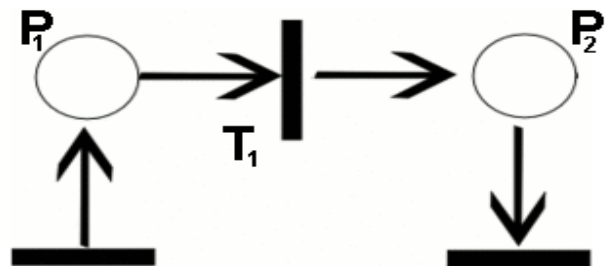
Usually, such business processes are developed with a graphical tool which allows easily defining, designing and administering business processes for business and IT users. It offers a powerful process definition environment enabling accurate implementation of complex, interrelated processes with broad functionalities.

The aforementioned advantages of Business Process Management (BPM) are widely spoken of, but not enough attention is paid to the optimization problem. Today's tools offer the functionality of business process optimization. However, expert knowledge is required to use them efficiently. With their experience backed up by software-based simulations, the consultants who operate such tools are able to specify the way of optimization of the process concerned. The weakness of the BPM description language is the inadequate description model.

Only few business process description models enable the use of process map optimization methods and analyses (process paths in the form of a graphical map) to improve the timeliness (accelerate), to ensure the optimum use of resources, or to save money.

The apparatus for modeling communications with automata [1], introduced by Carl Petri in 1962, is a tool perfectly suited for the optimization of business processes. In the simplest version, the Petri Net comprises of "places," "transitions," and directed arcs. Such a net may only be used to describe a system as a static connection of possible states. The graphical representation of the net is a bipartite graph in which the nodes represent places (signified by circles) and transitions (signified by bars).

Figure 1. Petri Net diagram



The Petri Net is a 4-tuple $C=(P,T,I,O)$, where $P = \{ p_1, p_2, \dots, p_n \}$ is a finite set of **places**, $T = \{ t_1, t_2, \dots, t_m \}$ is a finite set of **transitions**, $I : T \rightarrow P^*$ is the input function, and $O : T \rightarrow P^*$ is the output function. The sets of places and transitions are disjoint: $P \cap T = \Phi$.

The value of the $I (t_j)$ function is the collection of input places of the t_j transition.

The $\# (p_i , I (t_j))$ notation means the number of occurrences of the p_i place in the $I (t_j)$ collection.

The value of the $O (t_j)$ function is the collection of output places of the t_j transition.

The $\# (p_i , O (t_j))$ notation means the number of occurrences of the p_i place in the $O (t_j)$ collection.

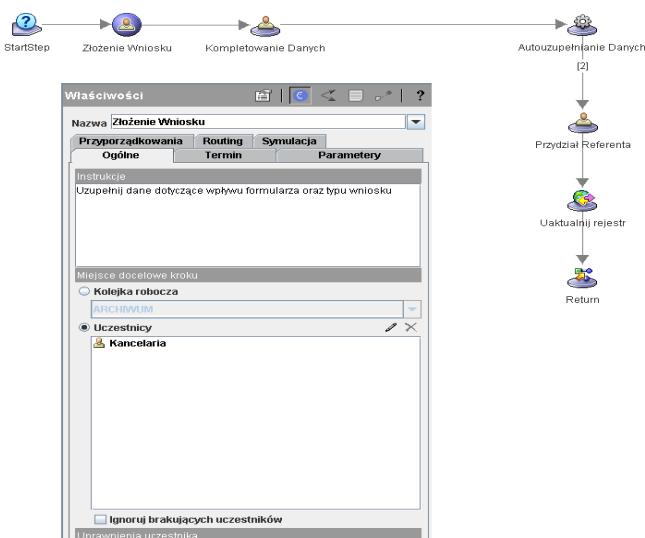
The nodes of the graph are connected with directed arcs so that no two places and no two transitions are connected directly. The reachability tree, the matrix equations and the invariants of places are the methods of Petri Net analysis which enable optimization of business paths.

2 Business process optimization

2.1 Definition of the workflow path

The example of the path optimized in this article is a sub-process of the map of handling a settlement request supported by BPM tools. Figure 2 below shows a part of the business process handling procedure: the presented sub-process of handling the electronic request comprises of six steps. Some of them are performed automatically.

Figure 2. Diagram of the process map with general properties of a process step

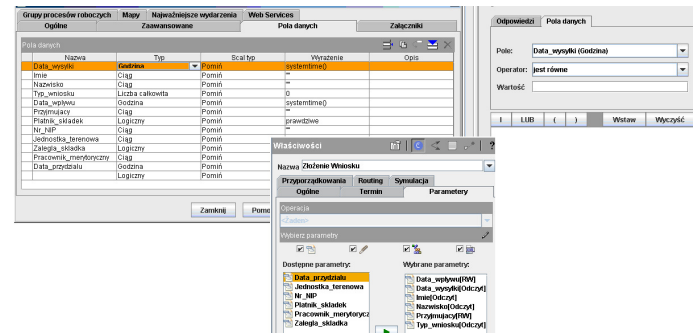


(e.g. the Auto-completion of Data where the connection with customer's external systems is established to retrieve the

information, while others are handled manually by specific actors [process participants]). The BPML model significantly differs from the mathematical model. It focuses on the definition and description of tasks assigned to participants, and on the making of appropriate decisions. However, that model does not enable the exact analysis in terms of the optimization of resources, time or costs involved in performance of the concerned process. On the presented sub-map, the first step (Submitting the Request) is performed by the Office participant. The task of that participant is to complete the corresponding process data and to transfer the current instance of the process to the next step.

The data necessary to handle the business process, and the logic of transitions between specific steps, are defined with task flow variables.

Figure 3. The definition of process variables, process properties available in the specific step, and definitions of conditional transitions between the steps



2.2 Conversion to the Petri Net model and optimization

The graphical map of the process is stored as a definition in an XML file. The open structure of the standard of the description of business paths allowed for the construction of a tool which translates the aforementioned structure to PNML (Petri Network Modeling Language). That tool uses the intermediate form which provides a metadata description of the map of the source business process.

Figure 4. Intermediate form of the source map of the process

```

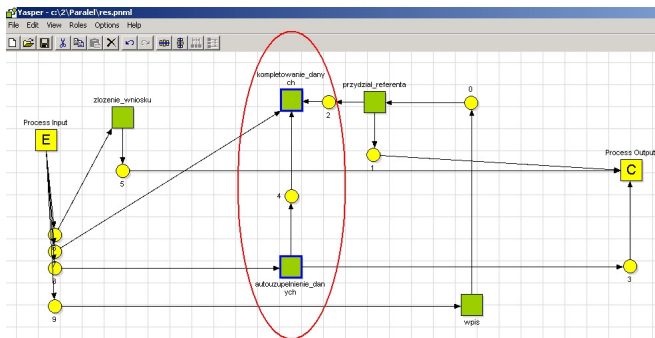
input.pep | test.txt | _input.txt
1 data_wysylki nazwisko imie typ_wniosku data_wplywu przyjmujacy platnik_skladek nr_NIP
2 zlozenie_wniosku data_wysylki imie nazwisko typ_wniosku ; data_wplywu przyjmujacy
3 kompletowanie_danych imie nazwisko ; nr_NIP jednostka_terenowa platnik_skladek
4 autouzupelnienie_danych imie nazwisko platnik_skladek ; zalegla_skladka
5 przydzial_referenta jednostka_terenowa typ_wniosku ; prac_merytoryczny data_przydzialu
6 wpis prac_merytoryczny data_przydzialu ; koniec
    
```

The conversion of information on steps, transition sequence, resources and variables included in the BPM map resulted in an equivalent notation based on Petri Nets. The basic PNML model was used [10].

Article includes tool for conversion between XPDL maps created in P8 Process Designer and PNML languages. Maps saved in Petri Network Modeling Notation could be open Yasper editor (one of the most popular Petri Network graph editor www.yasper.org).

The mathematical model developed as described above was subject to a simulation of flow enabling the optimization of transitions. The experiment resulted in a Petri Net map presenting the optimized BPM model:

Figure 5: Process map presented in the PNML Petri Net notation



Tools like Yasper gives powerful capabilities of simulation and optimization of Petri graph. The following property was used in the example analyzed above:

(1) $a1 \rightarrow a2$, p^* output parameters $a1$, p^* input parameters $a2$

(2) $a2$ does not depend on p^*

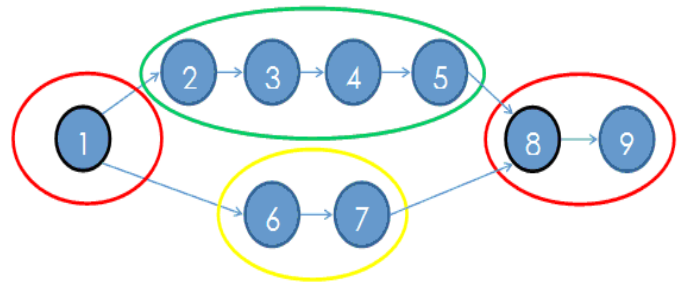
$a1 \rightarrow a2$, $a1||a2$

The program analyzed the relationships between specific steps. It has examined the dependencies between the sequence of steps and the data necessary for the performance of tasks specified in various steps, and identified the steps which may be performed in parallel. BPML imposes a specific manner of acting/thinking of the process. The conversion of the map to the Petri Net model allows for optimization of the business process. The resulting optimization of process timeliness provides measurable business advantages (acceleration, economies).

3 Experiments

The translation of the original process map to the equivalent PNML-based Petri Net model enabled performance of a complex analysis to determine the process steps which may be handled in parallel [11, 15]. The optimization algorithm was as follows:

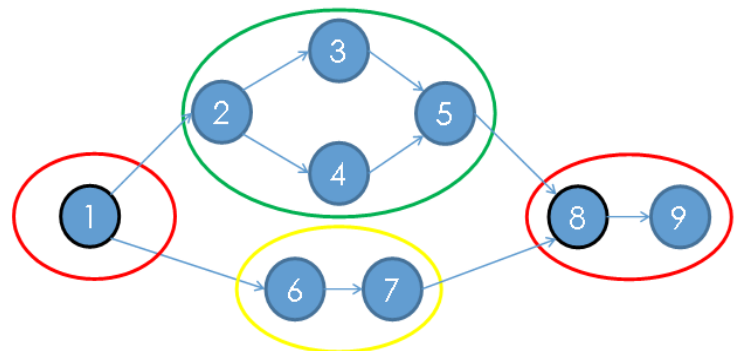
Arranging the elements into groups,



Parallel performance of steps within the groups

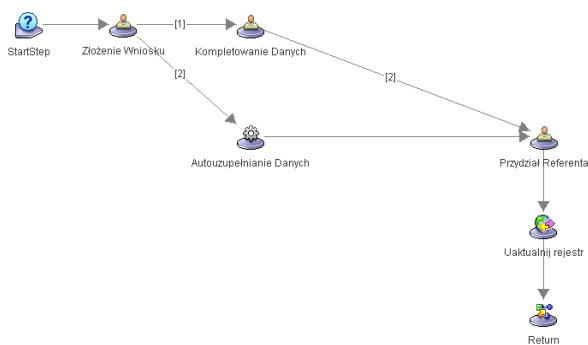


Reunification of groups



The conversion of the PNML-based process map to a definition readable by business process management systems resulted in a BPM path optimized for the performance time with steps handled in parallel. Optimized in Yasper map shows P8 process designers exact places where optimization could be applied. In this particular scenario, renders where to apply parallel processing for speeding up the process execution.

Figure 6: The optimized output process map



4 Conclusions

The experimental research presented in this article demonstrates that the Petri Net model may be efficiently used to map business processes used in BPM software, and to subsequently analyze and optimize these processes. Previous scientific studies on the aforesaid matter included only a few theoretical attempts to translate some processes described in BPMN 1.2 to the Petri Network Modeling Notation (PNMN) which corresponds to the standard Petri Net model. This document demonstrates the feasibility of the translation of process maps to the Petri Net model, and of the sophisticated optimization of business processes.

The paper presented an idea for applying Petri Network based optimization to business process designs. By developing a formal business process model and orienting it to Petri Modeling Notation, the generation of optimized business process map was facilitated and demonstrated using the real example. What make the business process optimization problem distinctive is its highly constrained nature and the fragmented search space that has a significant impact on locating the optimum solutions. It is shown that state-of-the-art Petri Network based optimization algorithms produced satisfactory results by generating and preserving optimal solutions on process designs. That provides an adequate alternative optimized process designs for the business analyst to decide the trade-offs between the different objectives. The results presented here are indicative and encouraging for further research in the area of business process optimization.

This series of articles presented approach differs significantly from the typical process optimization model. As opposed to the traditional approach of optimization, according to which the expert-matter knowledge is necessary - methods described in the articles is more automatic. With the use of traditional optimization methods it could bring significant values to business..

5 References

- [1] Harald Störrle Models of Software Architecture - Design and Analysis with UML and Petri-Nets, ISBN 3-8311-1330-0.
- [2] R. Dijkman, M. Dumas, B. van Dongen, R. Kaarik, and J. Mendling. Similarity of business process models: Metrics and evaluation. Working Paper 269, BETA Research School, Eindhoven, The Netherlands, 2009.
- [3] Best, Eike, Devillers, Raymond, Koutny, Maciej. "Petri Net Algebra".2001, XI, 378 p. 111 illus., Hardcover ISBN: 978-3-540-67398-9
- [1] B. F. van Dongen, R. M. Dijkman, and J. Mendling. Measuring similarity between business process models. In Proc. of CAiSE 2008, volume 5074 of LNCS, pages 450–464. Springer, 2008.
- [2] Suzana Donatelli, H C M Klejn, APPLICATION AND THEORY OF PETRI NETS 1999. Lecture Notes in Computer Science, 1999, Volume 1639/1999, 7
- [3] H. Chen and A. Mandelbaum, "Discrete flownetworks: Bottleneck analysis and fluid approximations," Math. Oper. Res., vol. 16, pp. 408–446, 1991.
- [4] Tadao Murata: "Petri Nets: Properties, Analysis and Applications" Proceedings of the IEEE, vol. 77, No. 4, April 1989.
- [5] A. Giua and E. Usai, "High-level hybrid Petri nets: A definition," in Proc. 35th Conf. on Decision and Control, Kobe, Japan, Dec. 1996, pp.148–150
- [6] W.M.P. van der Aalst. Verification of Workflow Nets. In P. Azéma and G. Balbo, editors, Application and Theory of Petri Nets 1997, volume 1248 of Lecture Notes in Computer Science, pages 407.426. Springer-Verlag, Berlin, 1997
- [7] M. Jungel, E. Kindler, and M.Weber. The Petri Net Markup Language. In S. Philippi, editor, Proceedings of AWPN 2000 - 7thWorkshop Algorithmen und Werkzeuge f ur Petrinetze, pages 47.52. Research Report 7/2000, Institute for Computer Science, University of Koblenz, Germany, 2000.
- [8] T. Murata. Petri Nets: Properties, Analysis and Applications. Proceedings of the IEEE, 77(4):541.580, April 1989
- [9] A. Hirschall, S. Artishchev, Enabling e-Business Transaction Support in a Petri-netBased Workflow Description Language, Faculty of Technology and

Management, Department of Information and Technology,
P.O. Box 513, NL-5600 MB, Eindhoven, The Netherlands

[10] WIL M. P. VAN DER AALST ET AL, Workflow Patterns: On the Expressive Power of Petri-net-based Workflow Languages, h10_2004

[11] R. S. Aguilar-Saven, "Business process modelling: Review and framework," *Int. J. Prod. Econ.*, vol. 90, pp. 129–149, 2004.

[12] WIL M. P. VAN DER AALST ET AL, Pattern-Based Analysis of BPML and WSCI, 2004.

Modelling Rail-road Crossing Control using OPN

Arghya Ghosh¹, Ranjan Dasgupta²

¹Student, Dept. of Computer Science and Engineering,
National Institute of Technical Teachers' Training and Research, Kolkata, West Bengal, India

²Dept. of Computer Science and Engineering,
National Institute of Technical Teachers' Training and Research, Kolkata, West Bengal, India

Abstract - This paper deals with Object PetriNet to model the 'rail-road crossing' problem. Here we discuss the rail-road crossing problem from two perspectives - one from the level crossing side and the other from light signal control. Both the issues are modelled by using Object PetriNet and safeness condition has been identified.

Keywords: PetriNet, Object PetriNets, Safeness conditions, Traffic signal control.

1 Introduction

Modelling of complex systems is a fundamental requirement for understanding the complexity of the system and also helps in designing appropriate solution for the system. PetriNet [1][2][11] is one of such graph based tools used by the researchers in this regard. Several variations of PetriNets [12] are available to address problem of different level of complexities.

In this paper, we discuss the 'Rail-road Crossing' problem and modelled it with the help of Object PetriNet (OPN) [6][7][9]. OPN is very suitable here as the basic logic of signal control for both rail & road are the same and hence one can inherit the features of the others. Moreover, as several functions need to be triggered at the same time, like change of signal from green to red at the rail side need to be passed to the road side to change the position of level crossing to down state, the use of methods help in designing such sequence of operations.

2 Rail-road Level Crossing problem

One of the operational goals of the train control system is the safe train movement through the crossing of a railway and road traffic.

2.1 Spatial relationship of Rail-road Crossing

In case of Automated Level Crossing Systems (ALCS)[13][5], the spatial relationship, connecting the level crossing with the train can be derived through analysis of the regular behaviour. Figure -1 shows the pictorial view of the railroad area. It is described by the following three

operational states (i) No train in influence area, (ii) train in the approaching area and (iii) train is passing the approaching area (i.e. passing the approaching level crossing also).

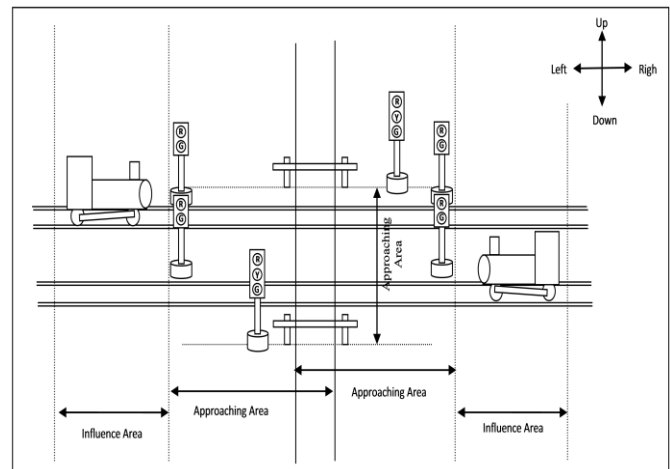


Figure 1 Spatial relationship of railroad crossing

2.2 Operational view of Rail-road Level Crossing

When a train is in influence area, it switches on the trackside control unit by radio communication. Consequently, the level crossing control first switches the traffic light for the road traffic to "yellow" and later to "red". After a predefined time, the lowering of the gate will be started. When the gate has reached its down position, the level crossing is at safety. In a sufficient distance to the danger area, the train interrogates the status of the level crossing. When the train gets the statement that the level crossing is at safety, it passes the Approaching Area (AA). After leaving the level crossing, the train activates a switch-off contact and induces the annulment of the protection.

This approach only ensures the monitoring of railway track. It does not ensure the roadside monitoring. However, there is always one possibility; at the time of closing the level crossing gate [3] due to some unavoidable reason, one vehicle can be trapped on the railway track between two gates. Figure 2 shows pictorial scenario of the same.

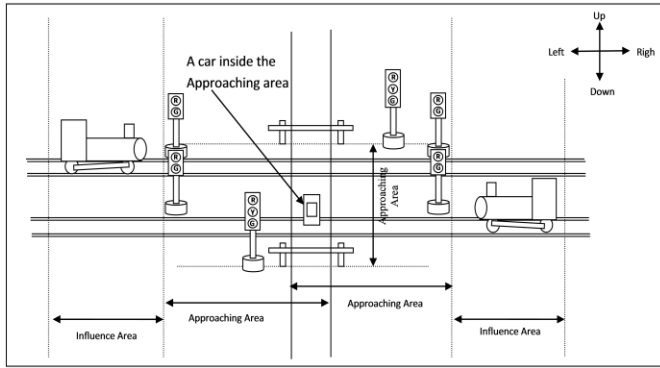


Figure 2 A typical scenario of railroad crossing

3 Control logic for Rail-road Level Crossing

Let,

LT_{in} :Number of Left side Train inside the Approaching Area

RT_{in} :Number of Right side Train inside the Approaching Area

LT_{out} :Number of Left side Train leaving out from the Approaching Area

RT_{out} :Number of Left side Train leaving out from the Approaching Area

Now, whenever a train has reached the Influence Area (IA), the communication between the train and the control station is established and when the train leaves out from the Approaching Area (AA) then also the control station is informed.

Within a particular time span (t) (i.e. time span starts when train just reach the IA and ends with living out from AA.) it can be said that –

$m1 : (LT_{in} + RT_{in}) - (LT_{out} + RT_{out}) > 0$ infers , there is at least one train inside the AA.

In case of roadside:-

Let,

UV_{in} : Number of Up side vehicles inside the road side Approaching Area

DV_{in} : Number of Down side vehicles inside the road side Approaching Area

UV_{out} : Number of Up side vehicles living out from the road side Approaching Area

DV_{out} : Number of Down side vehicles living out from the road side Approaching Area

Then $m2: (UV_{in} + DV_{in}) - (UV_{out} + DV_{out}) > 0$ infers, there is at least one vehicle inside the road side AA within the time span t.

3.1 Safeness of Rail-road Crossing

Whether there is at least one vehicle inside the road side Approaching Area then definitely no train gets the permission to enter the rail side Approaching Area and vice versa .

The collision avoidance constraints can be written as

- (i) If $m1 = 0$ And $m2 = 0$ then Safe
- (ii) If $m1 > 0$ And $m2 = 0$ then Safe [Assuming there is no train vs. train collision]
- (iii) If $m1 = 0$ And $m2 > 0$ then Safe [Assuming there is no vehicle vs. vehicle collision]
- (iv) If $m1 > 0$ And $m2 > 0$ then Unsafe

3.2 Compatible issues of light signaling

We also must take care of compatibility issues between rail light signaling and road light signaling. In rail light signaling [13], there are different states of light signaling, in this paper, two states light signaling (red & green) is considered, where Green light indicates- proceed at line speed. Expect to find next signal displaying green. Red indicates stop. The compatible matrix of different light signaling is given below.

	R_t	G_t
R_r	1	1
G_r	1	0
Y_r	1	0

Where R, G and Y indicate the Red, Green and Yellow light respectively and the suffix r and t indicate the roadside and train side signaling respectively. The Yellow light of the roadside prepare to find next signal displaying red.

In the compatible matrix, one (1) represents the possibility of occurrence of the states of lights for both road side & rail side. That is, when the road side signal is Red then there is the possibilities of Red or Green signal in rail side. On the other hand zero (0) means there is no possibility of occurrence. i.e. when Green light is in the road side then the train side signal cannot have the green light signal high.

4 Object Petri Nets (OPNs)

Object Petri Nets belongs to the class of high level Petri Nets. OPNs extend CPNs in number of significant ways. In Object Petri Nets the Petri Nets or subnets [10][14] are defined by a class, which can be instantiated in a number of different contexts. Each class can contain data fields, functions and transitions. The objects of a class can represent tokens. Methods or functions share the transition, and they have bindings of variables and evaluated at a particular point in time. OPNs are capable to explain the definition of a method of a class. It supports inheritance property[8], and reusability, which reduce the designing complexity as well as effort at design level. The ability to

define function with read-only access to the current state of a subnet is another significant extension of OPNs over CPN. Formal definition [6] of OPNs is in bellow.

OPNs An OPN is an 11-tuple

OPN = (P, T, F, NPF, I, G, C, AE, NPAE, TO), where

- (i) P is a finite set of places
 $P = OP \cup NP$ where
 – OP is a set of (ordinary) places
 – NP is a set of named places,
 where a mapping name : $NP \rightarrow PlaceId$ is defined
- (ii) T is a finite set of transitions, $P \cap T = \emptyset$
- (iii) F is the set of arcs (the flow relation),
 $F = DF \cup NF$ where
 – DF is a set of (deterministic) arcs,
 $DF \subseteq (P \times T) \cup (T \times P)$
 – NF is a set of non-deterministic arcs,
 $NF \subseteq (P \times T) \cup (T \times P)$
- (iv) NPF is a set of arcs which are adjacent to the place name of a named place,
 $NPF \subseteq (NP \times T) \cup (T \times NP)$
- (v) T is a set of object types
- (vi) I is an initialization function,
 $I : P \rightarrow \mathcal{E}$ such that $\forall p \in P : Var(I(p)) = \emptyset$
- (vii) G is a guard function,
 $G : T \rightarrow \mathcal{E}(Bool)$
- (viii) C is a code function,
 $C : T \rightarrow \mathcal{E}_{MS}$
- (ix) AE is an arc expression function,
 $AE : F \rightarrow Arc(\mathcal{E})$
- (x) NPAE is an arc expression function for arcs $\in NPF$,
 $NPAE : NPF \rightarrow \mathcal{E}(PlaceId)$
- (xi) TO is a timeout function,
 $TO : T \rightarrow IN$

where

– \mathcal{E} denotes the set of all expressions, $\mathcal{E}(T)$ the set of all expressions $E \in \mathcal{E}$

such that $Type(E) = T$;

– $Arc(\mathcal{E})$ the set of multiset expressions .

wrapper function $AAE : F \cup NPF \rightarrow Arc(\mathcal{E})$, where

$AAE(f \in F) = AE(f)$ and $AAE(f \in NPF) = NPF(f)$

5 OPNs model for Rail-road Level Crossing

To achieve the safeness rules of railroad level crossing and non conflicting operations of the different light signal control, OPNs models are divided in two section (i) OPN models of level crossing system and (ii) OPN models of light signal system.

Transitions are drawn as rectangles, (in one transition there are more than one methods can be called.) and ovals are representing the places. Data field generalizes Petri Net places and are therefore drawn as ovals. Their annotation indicates the type, the identifier, an optional initial value (preceded by a character '=') and an optional guard

(preceded by a character '['). Functions are drawn as rectangles to emphasize that they share transition semantics. The exporting of fields and functions are indicated by drawing them with a double outline.

5.1 OPN models of Level Crossing System

To implement the safeness rule of the railroad crossing, two classes are modeled (i) LevelCrossing class and (ii) RailRoad Level Crossing control class.

5.1.1 Level Crossing class

In this class two data members are used, -

String data | "UP" || "DOWN" :- 'data' of String type with only two possible value "UP" or "DOWN" to store the value of the current state of the level crossing gate.

Int Counter:- A integer type counter variable to count the number of vehicles inside the Approaching Area of road side or the number of trains inside the Approaching Area of rail side.

And four methods are used, -

insideAA(String info): Method insideAA(string data) used to inform of incoming and outgoing vehicles or train in Approaching Area of the both traffic side or road side. If the argument value of the method is "IN", then it increases the value of the counter variable. If the argument value is "OUT", then it decreases the value of the counter variable.

Boolean insideIA(): The method insideIA() is informed of incoming and outgoing vehicles or trains in Influence Area of the both traffic side or road side. It returns Boolean type, if the returned value is TRUE, then there exists a vehicle or train in the Influence Area of road side or train side respectively, else there is no vehicle or train inside the Influence Area.

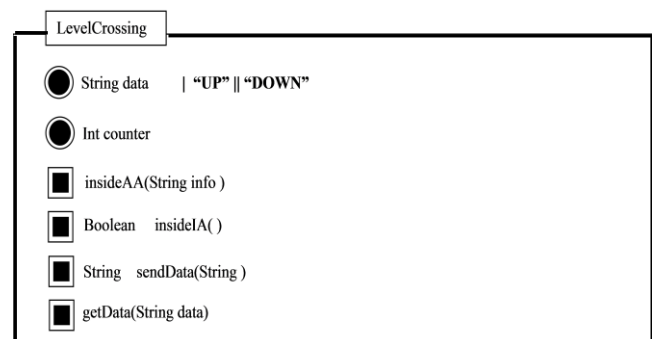


Figure -3 LevelCrossing class.

String sendData(String): Method sendData(String) is used to send the information of the outside of the class to communicate with the other objects of the same or different class.

getData(String data): This method is used to receive the information of the other objects of same or different classes. Figure -3 shows the description of the LevelCrossing class.

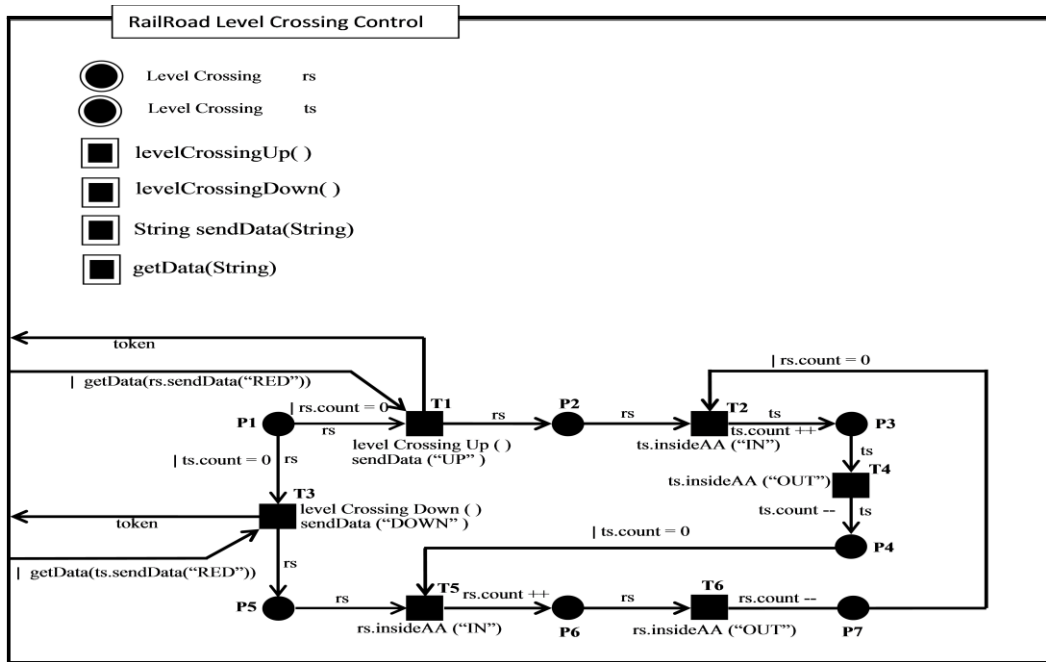


Figure -4 RailRoad Level Crossing Control class

5.1.2 RailRoad Level Crossing Control class

In this class two data members are used. -

LevelCrossing ts: Level Crossing type object variables to represent road side control.

LevelCrossing rs: Level Crossing type object variables to represent rail side control.

And, four methods are used.-

levelCrossingUp(): To change the state of level crossing gate from close (Down) to open (Up).

levelCrossingDown(): To change the state of level crossing gate from open (Up) to close (Down).

String sendData(String): To send the information of the level crossing state (i.e., “UP” or “DOWN”) to communicate with other object(s) of the same or different class and return that state as string.

getData(String): To get the information of the states of the different light signaling (such as, road light signaling & train light signaling)

RailRoad Level Crossing Control class is responsible for operating the level crossing gates safely (i.e. gate can only be opened when no train is inside the train side Approaching Area. And must be closed when train is inside the train side Approaching Area and no vehicle is inside the road side Approaching Area). Following constraints are considered for maintaining the safeness of the level crossing, in the model of the RailRoad Level Crossing Control class.

1) The level crossing gate can move in upward direction, only when rs.count = 0, (i.e., there is no train in the AA of

the rail side) and the “Red” light status of the rail side light signaling (i.e., getData(rs.sendData(“RED”)) must be in high state where rs representing the object of the Train Light Signal class).

2) If the level crossing gate status is open and rs.count = 0 (i.e. no train in AA of the rail side), then only it is possible for entering vehicles inside the AA of the road side and fire the transition t2 to call ts.insideAA(“IN”) and increases the value of ts.count by one.

3) If there is at least one vehicle inside the road side AA then only transition t4 is fired to call the method ts.insideAA(“OUT”) for each leaving out vehicle from the road side AA and decrease the value of ts.count by one.

4) The level crossing gate can move in downward direction, only when ts.count = 0, (i.e., there is no vehicles in the road side AA) and the “Red” light status must be in high state of the road side light signaling (i.e., getData(ts.sendData(“RED”)) where ts representing the object of the Traffic Light Signal class).

5) If the state of the railroad level crossing gate is closed and ts.count = 0 (i.e. no vehicles in AA of the road side) then only it is possible for entering trains inside the AA of the train side and fire the transition t5 to call rs.insideAA(“IN”) and increases the value of ts.count by one.

6) If there is at least one train inside the road side AA then only transition t6 is fired to call the method rs.insideAA(“OUT”) for each leaving out train from the rail side AA and decrease the value of rs.count by one.

Figure – 4 shows the description of the RailRoad Level Crossing Control class. For example :-

Let, initial states of the level crossing gates are closed (i.e. the gates are in down position). Then definitely the "Red" light signal of the road side traffic is in high state and also the train side light signal can have the "Green" state high. Now to change the state of the level crossing gates, there must be required the "Red" light signal to be in high state at the rail side light signaling (i.e. `getData(rs.sendData("RED"))`, where `rs` represent the rail side traffic signaling object) and also `rs.count=0` (i.e., no train is in rail side AA). Then only the transition `t1` can fire and `levelCrossingUp()` method is called and send the current state (using `sendData("UP")`) after the up operation of the level crossing gate has been performed. In this class, to communicate with the other objects of same or different classes, two transitions variable `rs` and `ts` are used. Now, to move vehicles inside the AA of the road side the places `p2` and `p7` must be required to be in active state, i.e., transition `t2` can be fired only when the level crossing gate is in open state and no train is in AA of the rail side. After firing the transition `t2`, for each vehicle inside the road side AA, `ts.insideAA("IN")` method is called and `ts.count` is increased by one. The transition `t4` can fire if the vehicle is leaving out from the road side AA and then calls the method `ts.insideAA("OUT")` for each leaving out vehicle from road side AA and also decrease the value of `ts.count`.

On the other hand, if the level crossing gate is required to change the state from open to close for giving clearance to the train(s). Then there must be required the "Red" light signal is in high state at the road side light signaling (i.e. `getData(ts.sendData("RED"))` where `ts` represent the road side traffic signaling object) and also `rs.count=0`, then only the transition `t3` can fire and `levelCrossingDown()` method is called and send the current state (using `sendData("DOWN")` method) after the close operation of the level crossing gates. Now to fire the transition the places, `t5`, `p5` and `p4` must be required to be in active state, i.e., there is no vehicles inside the road side AA and the level crossing is in closed state and for each train the method `insideAA("IN")` is called and also increase the value of `rs.count` by one. Transition `t6` is fired for each leaving out train from the rail side AA and also decrease the value of `rs.count` by one.

5.2 OPN models of Light signal system

To design with non conflicting conditions of different light signaling (road side / train side), in this paper three classes are defined, (i) 'Traffic Light Signal' class, (ii) 'Train Light Signal' class which inherits the properties of the class 'Traffic Light Signal', and (iii) Light Signal Control class.

5.2.1 Traffic Light Signal class

In this class, four data members are used. -
 Boolean Red: it can take only true/false to represent state (ON/OFF) of the Red light.

Boolean Yellow : it can take only true/false to represent state (ON/OFF) of the Yellow light.

Boolean Green: it can take only true/false to represent state (ON/OFF) of the Green light.

String Colour: it is used to hold the name of the current high state (ON) of a particular light.

And three methods are used. -

`getData(String)` : It has no return type and takes one argument as String type for getting the information from the other objects.

`String sendData(String Colour)`: It has one String type argument and returns the String value to communicate with other objects of the different classes.

`changeTo(Boolean Colour, Boolean Colour)`: Method `changeTo(boolean, boolean)` has two arguments of Boolean type and it can change the state of the light signal

Figure-5 shows the description of the class.

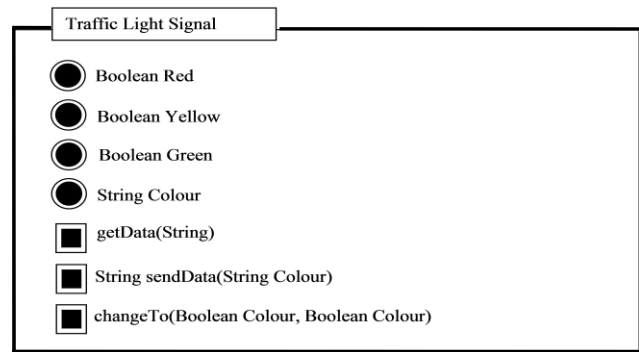


Figure -5 Traffic Light Signal class

5.2.2 Train Light Signal

In the train light Signal control, most of the properties are similar of traffic signal control [4]. Therefore, Train Light Signal class inherits the property of Traffic Light Signal. The grey shading identifies those features or components, which are inherited without any change; while the black shading indicates components, which have been introduced by this class or which overrides inherited components. Here only `changeTo` method is overridden and change the states of light Green/Red). Figure 6 shows the description of TrainLightSignal class.

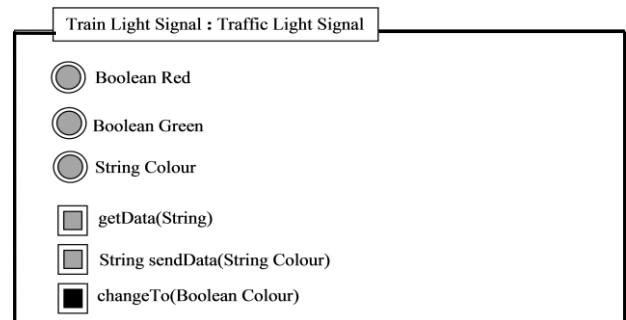


Figure 6 Train Light Signal class

5.2.3 Light Signal Control class

In Light Signal Control class, the ts and the rs are two objects of two classes Traffic Light Signal and Train Light Signal respectively. Figure 7 shows the OPN model of the class, where the traffic side yellow light is initiated with high value (i.e. ts.Yellow = true). From that place there are two possible transitions, viz -

(i) change Yellow(Y) to Red(R), - calls the method `changeTo(Y, R)` which changes the Red (R) state value to true (i.e. Red = true) and the states Green(G), Yellow(Y) become low (i.e. ts.Yellow = False; ts.Green = False). `sendData("RED")` method is also called for sending the data to communicate with the object of class Train Light Signal (i.e. rs). In this model, the transition variables are tr and rs.

(ii) change Yellow(Y) to Green, - calls the same method with the different value. i.e. `changeTo(Y, G)`. `sendData("Yellow")` is also called to send the information of the change. Same thing is applicable for changing the state Red to Green.

For example, once the Red gets high value, `rs.getData("RED")` is called and it then change the rail signal red to green (assuming no other constrains are there) using `rs.changeTo(G)` method. At the same time `rs.sendData("Green")` is called. Whenever the `rs.changeTo` method is called the communication is required to inform the states of the lights with the other objects of the different classes and also avoiding the conflicting conditions. Here the transition variable rs ultimately communicate with the Level Crossing system and road side light signaling system.

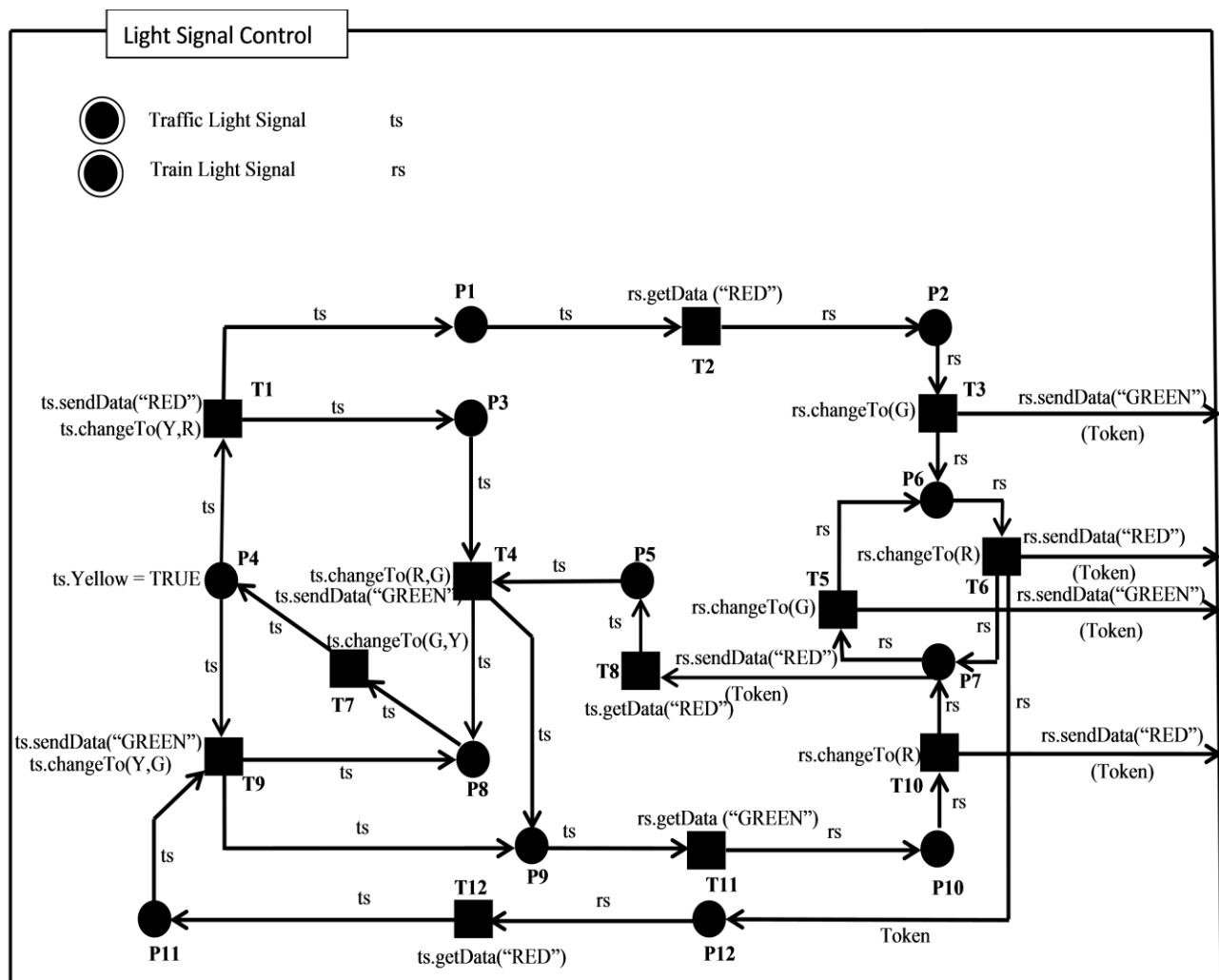


Figure 7 Light Signal Control class

Same things are happened when the `rs.getData("GREEN")` method is called by firing the transition. This transition is only fired when Red light signal of the rail side become active (i.e `getData("RED")`) and `ts.Red=TRUE` . In figure – 7, all possible transitions for making the light signal control conflict free for both traffic and rail side light signaling have been modeled.

6 Conclusions

Applicability of OPN in Rail-road Crossing has been shown in this paper. Model for both level crossing and light signaling have been developed and use of Object Oriented concept has been applied while designing the model. A simple mathematical form for safeness condition has also been proposed which in turn will help in understanding the system. This can also be extended to parallel Rail-road crossing by using the concept of inheritance.

7 References

- [1] Yi-Sheng Huang; Ta-Hsiang Chung; Che-Ting Chen. "Modeling traffic signal control systems using timed colour Petri nets". *Systems, Man and Cybernetics*, 2005 IEEE International Conference, pp 1759 - 1764 Vol. 2, 2006.
- [2] George F. List, "Modeling Traffic Signal Control Using Petri Nets"; *IEEE Transaction on Intelligent System*, Vol. 5, No.3, 2004.
- [3] Tso-Hsien Liao, Guo-Shing Huang, Yi-Sheng Huang "Modelling critical scenarios in parallel railroad level crossing traffic control systems using statecharts". *ROCOM'11/MUSP'11 Proceedings of the 11th WSEAS international conference on robotics, control and manufacturing technology, and 11th WSEAS international conference on Multimedia systems & signal processing*, World Scientific and Engineering Academy and Society (WSEAS), 2011
- [4] George F. List, "Modeling Traffic Signal Control Using Petri Nets"; *IEEE Transaction on Intelligent System*, Vol. 5, No.3, 2004.
- [5] Diana, F.; Giua, A.; Seatzu, C., "Safeness-enforcing supervisory control for railway networks ", *Automation Science and Engineering*, IEEE Transactions, pp 99 – 104, vol.1,2002.
- [6] Tom Hovoet, Pierre Verbaeten, "Usng Petri Nets for Specifying Active Objects and Generative Communication", *Concurrent OOP and PN, LNCS 2001*, Springer Verlag Barlin Heidelberg, pp 38-72, 2001.
- [7] Charles Lakos, " Object Oriented Modelling with Object Petri Nets", *Concurrent OOP and PN, LNCS 2001*, Springer Verlag Barlin Heidelberg, pp. 1-37,2001.
- [8] Costa,S.A.D. ,Guerrero,D.D.S., deFigueiredo,J.C.A. ;Perkusich A. "Inheritance Issues in Object-Oriented Petri Net Models" *Systems, Man, and Cybernetics*, 1998. 1998 IEEE International Conference , Vol. 1, pp 196 – 201, 2003
- [9] Bastide, R. ; Sibertin-Blanc, C. ; Palanque, P. " Cooperative Objects : A Concurrent, Petri-Net Based, Object-Oriented Language", *Systems, Man and Cybernetics*, 1993. 'Systems Engineering in the Service of Humans', Conference Proceedings., International Conference, Vol. 3, pp 286 – 291, 1993.
- [10] Perkusich, A. ; de Figueiredo, J.C.A. " Design of Distributed Track Vehicle Systems Applying a High-Level Object Oriented Petri Net Methodology", *Systems, Man and Cybernetics*, 1995. *Intelligent Systems for the 21st Century.*, IEEE International Conference, Vol. 1 pp 389 – 394, 2002.
- [11] Ballarini, P. ; Djafri, H. ; Dufлот, M. ; Haddad, S. ; Pekergin, N. " Petri Nets Compositional Modeling and Verification of Flexible Manufacturing Systems", *Automation Science and Engineering (CASE)*, 2011 IEEE Conference, pp 588 – 593, 2011
- [12] Watanabe, H.; Tokuoka, H.; Wu, W.; Saeki, M. "A Technique for Analysing and Testing Object-oriented Software Using Coloured Petri Nets", *Software Engineering Conference*, 1998. *Proceedings. 1998 Asia Pacific*, pp 182 – 190, 1998.
- [13] Wei Zheng ; Mueller, J.R. ; Slovák, R. ; Schieder, E. "Function modelling and risk analysis of automated level crossing based on national statistical data", *Informatics in Control, Automation and Robotics (CAR)*, 2010 2nd International Asia Conference, Vol. 2, pp 281 – 284, 2010.
- [14] Rodrigues, C.L.; Guerrero, D.D.S.; de Figueiredo, J.C.A. "Model Checking in Object-oriented Petri Nets" *Systems, Man and Cybernetics*, 2004 IEEE International Conference, Vol. 5, pp 4977 – 4982, 2004.

SESSION

BATTLESPACE REPRESENTATION FOR AIR, SPACE, AND CYBER

Chair(s)

Dr. Vince Schmidt

Battlespace Representation for Air, Space, and Cyber

James McCracken¹, and Denise Aleva²

¹The Design Knowledge Company, Fairborn, Ohio, U.S.A.

²Air Force Research Laboratory, 711th Human Performance Wing, Human Effectiveness Directorate, Battlespace Visualization Branch, Wright Patterson AFB, Ohio, U.S.A.

Abstract – We present a summary of a body of work executed over the last eight years addressing battlespace representations in the domains of air, space, and cyber. We couch this presentation in an historical context and relate it to design principles as well as user-centered design processes. We summarize the work in each of the domains and conclude with some thoughts about supporting real work in applied settings.

Keywords: visualization, agile work environment, work-centered support systems, air, space, cyber

1 Introduction

Visualization has a long history. Its value is well illustrated by the work of Dr. John Snow. Between 1831 and 1854, Great Britain suffered four major outbreaks of cholera. Throughout this 20-year period, and despite volumes of recorded data on casualties, the situation baffled the local authorities as well as the scientific community. Snow created an overlay of the victims in a single neighborhood, and observed that the epicenter of the epidemic converged at a water pump on the corner of Broad Street and Cambridge Street.¹ The map can be viewed at <http://www.ph.ucla.edu/epi/snow/highresnowmap.html>.

Similarly, battlespace representations have a long history of evolution and use. One of the authors once worked with a senior leader in the Cockpit Integration Office at Wright-Patterson Air Force Base (WPAFB) who frequently declared his NUTS theory: “Nothing under the sun.” The theory was based on the notion that technology changes, but the ideas stay the same. Thus, 3D displays, while a current technology trend, were used in WWII; they were cardboard models of targets carried in the bombers to allow bombardiers to see how the target would look with current shadows. See Figure 1 for an example.

Command and control has changed as well as targeting. Figure 2 illustrates a WW II-era command center built in tunnels near Dover, England. The plotting table was used to keep track of aircraft movements as radioed in by spotters. The concepts, in both cases, 3D models and command and control centers are thus demonstrated to not be novel; the concepts stay the same, but the technology to support them is ever-changing. The ever-changing technology landscape leads developers of Battlespace representations to both stay

abreast of current technology and to seek invariants that can be used to shape or guide designs.



Figure 1. A cardboard model used by bombers.

Command and control has changed as well as targeting. Figure 2 illustrates a WW II-era command center built in tunnels near Dover, England. The plotting table was used to keep track of aircraft movements as radioed in by spotters. The concepts, in both cases, 3D models and command and control centers are thus demonstrated to not be novel; the concepts stay the same, but the technology to support them is ever-changing. The ever-changing technology landscape leads developers of battlespace representations to both stay abreast of current technology and to seek invariants that can be used to shape or guide designs.

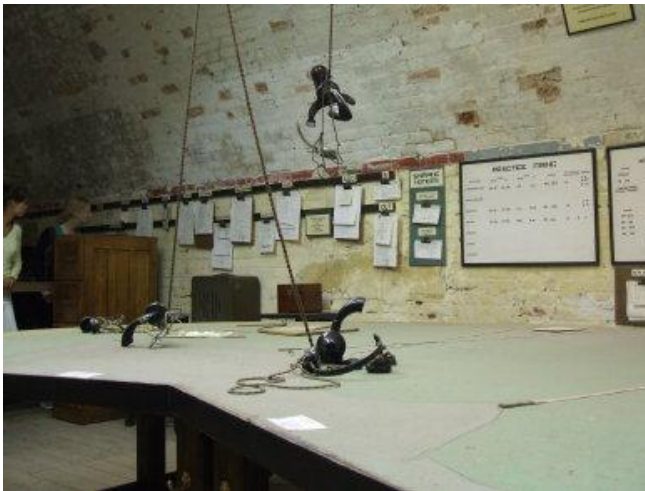


Figure 2. WWII plotting table in Dover tunnels 1.

2 Stable Design Principles

What is invariant is the human perceptual and cognitive apparatus that underlies the sense-making process when using current display technology and techniques. Thus, the relationship between and among displayed objects, their color, texture, and contours and the use of those relationships and properties in telling a story, communicating a situation to the user or operator, is critical. Display or graphics designers must be deeply aware of the needs of their users.

Also invariant are the notions of visual momentum, visual search, isomorphism, vernier acuity, Stroop effect, and Gestalt closure. Graphic or interface design can either induce or reduce visual search, dependent on the task at hand and the relationship between and among display elements. The Stroop effect, which manifests itself as interference in the processing of the names of colors written in colors other than those of the color being named, illustrates verbal and visual conflicts in processing information.

Strategies such as Shneiderman's "Overview first, zoom and filter, then details-on-demand."² or McCracken's progressive refinement in which the broad context is conveyed in the upper left of the display, and successively refined left to right and top to bottom, with the bridge between context and data being appropriately designed levels of abstraction/levels of detail.³

Yet another concept is the notion of surface structure vs. deep structure of the interface. This concept addresses issues such as system transport delays, which can manifest themselves at the user interface and thus impact the user's experience. Ignoring the effect of time delays on the delivery of data to dynamically updating displays can have deleterious, even catastrophic results (think Ground

Collision Avoidance Systems) GCAS). All of these elements (and more) are the underpinning of good display design. In order to support "connecting the dots" or making information available "at-a-glance" these human perceptual and cognitive factors must be addressed. Their consideration and integration in the interface development process are critical for success. This was as true in Snow's day as it is today.

3 Air Force UDOP Family of Systems

The Air Force Research Laboratory (AFRL), 711th Human Performance Wing (711th HPW), Battlespace Visualization Branch (AFRL/RHCV) has sponsored a series of Small Business Innovative Research (SBIR) efforts dating back to 2004 aimed at developing Command and Control (C2) Battlespace Representations for Air, Space, and Cyber. The Design Knowledge Company has won five of those contracts. These include the Satellite Threat Evaluation Environment for Defensive Counterspace (STEED), Local/Distributed Operational Collaborative User-Centered Support System (LOCUS), 4D NETcentric Framework For Operational C2 Environments (4D NETFORCE), Air Operations Community (AOC) System Engineering Toolkit (ASET), and Air, Space and Cyber User-Definable Operational Picture/Common Operational Picture (ASC&U/C). These are summarized in Figure 3. Space Weather Information Fusion Technology (SWIFT), Bridge for Usable Collaborative Knowledge Integration (BUCK-I), and Visualization of Information for Satellite Tactical Applications (VISTA) are plug-ins to the core UDOP technology and support battlespace operations in air, space, and cyber. These capabilities have matured to the point of being the work environment for the JMS program of record, and being used in the 2011 Joint Expeditionary Force Experiment and Advanced Concept Event. An overview is provided in Figure 3.

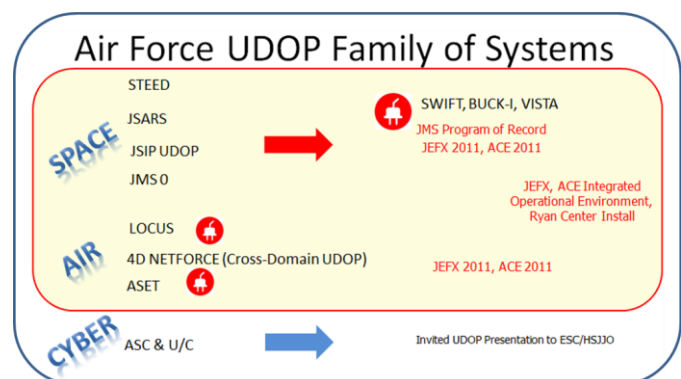


Figure 3. Family of Agile Work Environments

4 User-Centered Design

The United States Air Force has a long history of user-centered design. Each of the projects described began with an understanding of the user population. Cognitive task

analyses were performed, pertinent documents reviewed, users interviewed, and work processes observed. The key to our success is that TDKC cognitive and human factors engineers work closely with our software development team to ensure that we build a work environment that focuses on a single, unified user interface to multiple systems that is tailorable for individual users. This integrated team approach is critical because our JMS experience has demonstrated that legacy tools as well as many different vendors may be contributing services to support diverse functionality, each having different styles and thus requiring the cognitive and perceptual integration task to be done in close collaboration with the UDOP software development. We next describe space, air, and cyber visualizations in the form of Agile Work-centered Environments (AWEs) developed for AFRL/RHC.

4.1 Space

Space assets are critical to our military superiority, thus making them targets for attacks. These attacks were difficult to detect using current (2003) tools available at Space Operations Squadrons (SOPS) or Joint Space Operational Center (JSpOC) because the degradation effects may be gradual and appear to be isolated. Improved user work-centered technologies were needed to assist the operator in detecting otherwise imperceptible trends and to rapidly gather the information needed to report the incident. Neural networks were being refined for counterspace operations and could detect some trends, but this technology lacked the flexibility needed for warfighters to perform their own analyses. Current (2003) technology lacked advanced collaboration methods to communicate results of analyses across echelons; an example of this is communicating a finding from SOPS to the 14th Air Force. To address this problem, AFRL/RH sought innovative proposals to develop a Work-Centered Support System (WCCS) that bridged the gap between the analyst and the numerous information sources available, as well as collaboration across echelons. The resulting STEED project evolved through a series of steps into the Joint Space Operations Center (JSpOC) Mission System (JMS) UDOP. An example of one of the JMS UDOP perspectives is shown in Figure 4. JMS is built on a net-centric, Service-Oriented Architecture (SOA). STEED employed a variety of visualization approaches to build situation awareness support. These include 2D and 3D geovisualizations using NASA WorldWind, standard graphical user interface tools such as pick lists and tree structures, task-customized small multiples, and various time-based views, among others.

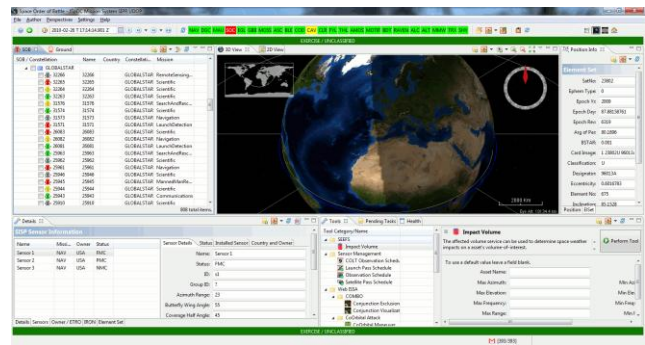


Figure 4. Using the Space Order of Battle Perspective to select a GLOBALSTAR asset.

4.2 Air

The 4D NETFORCE effort assessed the requirements to transform the existing TDKC STEED into a Command and Control (C2) environment to support work in Air Operations Centers (AOCs), providing integrated 2D and 3D spatial and temporal displays. The result of our investigations, working with the AOC experts from Teledyne-CollaborX, was that this goal was fully achievable. We identified open source tools to integrate various information sources such as InfoWorkSpace (IWS), email, databases and electronic chat, and created an ability to run Microsoft products such as Word, Excel, and PowerPoint inside of the 4D NETFORCE framework; this is critical because many AOC products are captured using Microsoft (MS) tools. A sophisticated data loader supports the import of data from a variety of data sources, including legacy stove-piped systems, JSeries messages, and net-centric services. 4D NETFORCE has been used at the 2011 Joint Expeditionary Force Experiment (JEFX) and Advanced Concept Event. It incorporates all of the UDOP core technology found in JMS. A sample perspective from the 4D NETFORCE capability is shown in Figure 5.

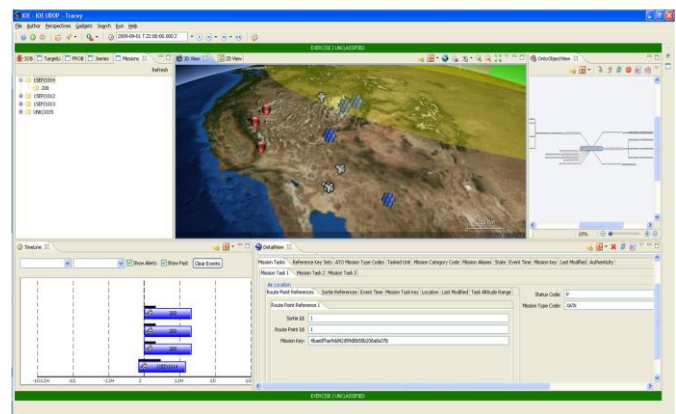


Figure 5. Air picture with order of battle, map, timeline and tasking views.

4.3 Cyber

Major General Ellen Pawlikowski, while Deputy Director of the National Reconnaissance Office (NRO), was asked at the 2009 AMOS conference about the biggest threat to space assets. Her answer was “Cyber.”

Focusing on visualizations, data handling and design, and operator aiding, the ASC&UC project is extending the existing UDOP capability into a comprehensive command and control (C2) visualization space and shared collaboration space for Air, Space, and Cyber warfighters. Meeting this challenge, The Design Knowledge Company (TDKC) is building a principled framework for visualization to ensure visual coherence over the three domains. Developing new innovative collaboration and visualization components that can integrate across space, air, and cyber domains will yield a robust and adaptable framework for lowering the cost of current and future developments and provide a much-needed synergy across development. A preliminary example of a cyber perspective is illustrated in Figure 6.

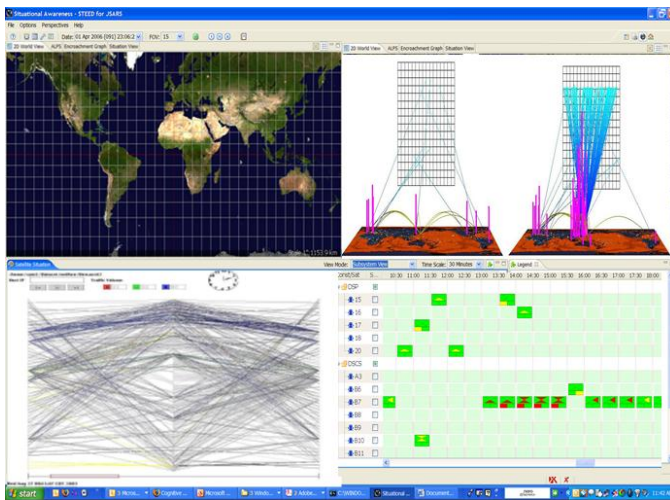


Figure 6. Cyber perspective showing map, data, and small multiple visualizations.

4.4 Cross Domain Work Environment

TDKC has derived a toolkit and framework from the components developed across all of the projects described above. The Agile Work-centered Environment (AWE) core contains the capability that is supported across all domains. AWE allows us to very rapidly integrate domain-specific data sources to develop tailored work environments. TDKC cognitive engineering processes define the information and knowledge required to support work₄; our design knowledge guides the structure of the user interface, and our software

development processes and usability testing ensure reliable, usable end products.

Using the AWE toolkit and accompanying processes, TDKC is building an AWE supporting situation awareness for the nuclear arsenal. The AWE toolkit brings the capability to develop what Lt. General Bob Elder, at the time Commander, 8th Air Force, called game-changing integration across domains, illustrated in Figure 7.5

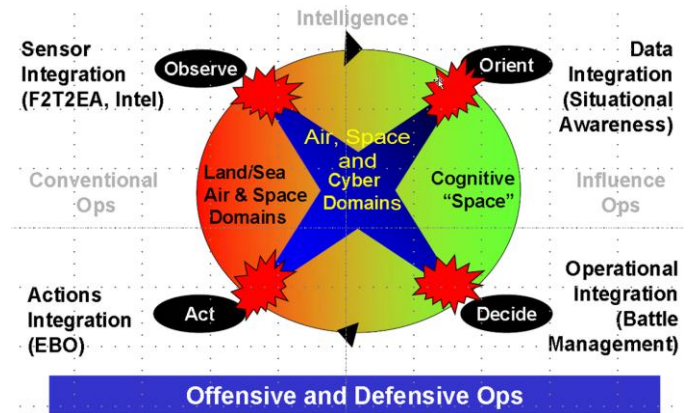


Figure 7. General Elder’s Conception of Game-Changing Integrated C2.

5 Conclusions

Figure 8 borrows another graphic from Gen Elder’s 2007 NDIA presentation. This graphic shows the 2007 vision of moving our command centers from having a multitude of stovepiped tools and data bases each with its own display to a set of services that can share data and present a single User Defined Operational Picture. In 2006-2007, there existed in the AOC some 88 different stand-alone tools. During a CTA conducted with JSpOC operators, the authors identified more than 75 different tools in use.

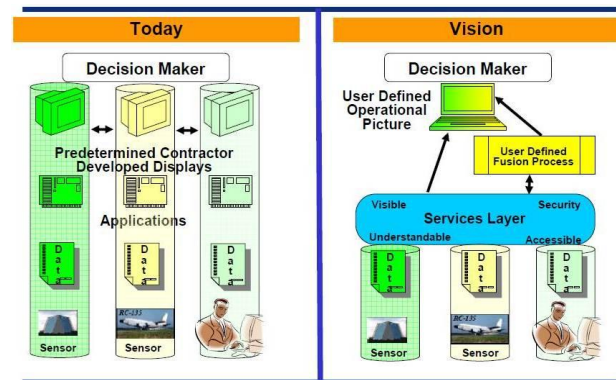


Figure 8. Evolution from legacy stovepiped systems to services-driven UDOP.

We've made considerable progress, particularly on the technology side. SOA has allowed us to move away from stovepiped tools and databases. However, this doesn't make decision maker's task any easier unless the data is fused and presented in a way that facilitates sensemaking. Bringing all this data together requires much more than simply hanging legacy tools or new applications on a SOA. Roth et al assert that "In too many cases, the introduction of computer technology has placed greater requirements on human cognitive activity. ...systems that provide access to more data or that automate more processes do not necessarily simplify the user's task or guarantee improved performance."6 Somehow must get from an overabundance of data to decision quality information. This requires correlating data in time and space, putting the right data together, dealing with conflicting data as well as metadata, and presenting the data to the decision maker in a way that is easily understandable.

Much of data fusion takes place in the grey matter of the human. Understanding how humans fuse data to reach conclusions is an important area of research. Visualizations are for the purpose of allowing humans to understand something – to get meaning. Designers must understand the applied work space to know what meaning is needed; they must have a deep understanding of the real work. Issues include more than providing individual pieces of information, but providing context and integrating the information in a meaningful way. We must also consider facilitation of shared understanding for collaboration.

Creating visual representations for individual applications is easy; creating a holistic operator environment is difficult. Operators and decision makers need the UDOP to be a holistic information composition expressed thru interactive visualization. The technology solution should amplify the human cognitive expertise and minimize interaction overhead.

6 References

- [1] Summers, Judith. Soho -- A History of London's Most Colorful Neighborhood. London: Bloomsbury, pp. 113-117. (1989).
- [2] Shneiderman, Ben. The Eyes Have It: A Task by Data Type Taxonomy for Information Visualizations. In Proceedings of the IEEE Symposium on Visual Languages, pages 336-343, Washington. IEEE Computer Society Press. (1996).
- [3] McCracken, James R. Unpublished internal training manual. The Design Knowledge Company. Fairborn, OH. (2011).

[4] McCracken, James R. Questions: Assessing the structure of knowledge and the use of information in design problem solving. Published doctoral dissertation. The Ohio State University, Columbus, OH. (1990).

[5] Elder, Bob. Air Force Cyberspace Command NDIA 2007 DIB Infrastructure Protection Symposium. Found at: http://www.ndia.org/Resources/OnlineProceedings/Pages/7030_DIBCIPConference.aspx (2007).

[6] Roth, Emilie, Patterson, Emily, and Mumaw, Randall. Cognitive Engineering: Issues in User-Centered System Design. In J. J. Marciniak (Ed.), Encyclopedia of Software Engineering, 2nd Edition. New York: Wiley-Interscience, John Wiley & Sons. (2002).

Automated Streaming Imagery and Filter Selection

Brian P. Jackson and A. Ardeshir Goshtasby

Department of Computer Science and Engineering, Wright State University, Dayton, Ohio

Abstract—A method for consolidating video streams from multiple visible and infrared cameras by analyzing their information theoretic properties is proposed. The following assumptions are made: 1) The videos are already registered at the static background. 2) The videos can be in the same or in different modalities. 3) Objects of interest are known ahead of time. Various methods for analyzing different modality images are discussed and compared. Experimental results show that a method incorporating mutual information, cascade classifier, and discrete wavelet transformation can effectively fuse multiple video streams into a single highly informative stream for a human observer or an automated surveillance system.

Keywords:

Image fusion, mutual information, human detection, MASINT

1. Introduction and Background

Human observers or automated vision systems are sometimes required to make a decision based on information received from multiple video streams. An abundance of camera modalities makes it possible to overcome environmental obstacles such as fog, smoke, or darkness, enabling an observer to react to cues in any one of the video streams. We require the ability to combine the most useful parts of different video frames captured simultaneously into a single frame for viewing or analysis. This capability can be achieved by providing the following functions: comparison of information in corresponding image regions, detection of the presence of objects of interest in a region, and fusion of image parts from different modalities.

In this paper, the objects of interest are considered to be humans. The problem of human detection has been addressed before. Of particular interest are techniques that are conducted in real time and work on moving targets. Pai et al. [5] detected pedestrian at crossroads in videos captured by stationary cameras.

Viola and Jones [8] developed a technique for visual object detection based on an adaptively boosted cascade of Haar-like features. The method has been implemented and used in face detection for various applications. Another popular method proposed by Jia and Zhang [3] uses the histograms of oriented gradients to detect pedestrians. Both methods can be trained to detect objects of interest.

Gualdi et al. [2] proposed a method for detecting humans in a video from a fixed-camera, focusing on areas in a frame where motion is present to reduce the search space

for human detection. They used covariance descriptors to train the presence of humans in an image.

When objects of interest are not present in an image, a comparison should still be made between corresponding areas in different video streams. Our method is influenced by the work of Culibrk et al. [1], which compares different saliency features to assess video quality. Statistical and information theoretic features were evaluated and used as markers to measure the quality of a video.

Fusion of regions from different images has been addressed before. In particular, methods to fuse multi-focus and multi-modal images based on wavelet transform have been successful. Zhang and Blum [9] used the Discrete Wavelet Transform (DWT) to fuse edges and regions in images in order to detect concealed weapons. Lewis et al. [4] extended this method to fuse image regions using semantic rules, each emphasizing a particular property in source images.

A more simplistic approach attempts to highlight areas of interest. Such an approach was analyzed by Rasmussen et al. [6] for its effect on human performance in a wilderness rescue setting. In an experiment, subjects were asked to detect objects of interest while responding to audio cues. The benefits of using a fused video scheme were compared to a side-by-side display of multimodal imagery, concluding that fused video has a smaller cognitive load on human operators.

The method discussed in this paper focuses on multimodal registered images. First humans in an image are detected. Then, a usefulness measure based on information gain and foreground motion analysis is computed. Next, a selection grid for image regions is generated. Finally, images are fused in the wavelet domain, focusing on an implementation that enables real-time use.

2. Experimental Design

2.1 Data

Two primary datasets were used in the experiments. The first dataset contained twenty-five frames in visible and IR modalities taken from the Octec dataset. The dataset represents a scene containing smoke and a person running behind the smoke. The second dataset contained two hundred frames from a handheld camera, showing a person walking in a snowy scene.

The two datasets represent different use cases. The Octec set is an example of multimodal data containing intentionally concealed human presence, during which the two modalities contain dissimilar information. The handheld camera dataset



Fig. 1: The two datasets used in the experiments.

is a much more simplistic use case, which is useful in examining the output of the system in a more familiar environment, containing no suspicious behavior.

2.2 Experimental Framework

A framework for processing video data was developed. The framework is capable of computing video statistics and information measures, carrying out filter operations, and manipulating spatial and temporal intensity histograms.

2.3 Usefulness Measures

Many regions in an image do not contain humans. However, it is still required to compare and evaluate the usefulness of corresponding regions in different modalities to decide which portions of an image to include in the fused image. Scenes with complicated backgrounds may contain information about structures and environment necessary for the planning of an action. Moving objects that do not fall into predefined objects of interest may also be relevant to an operator. Rather than attempting to classify every object in the image domain, a statistical measure is used to assess the relative usefulness of individual image regions across modalities.

Usefulness is defined in terms of statistical region-based measures and object classification. If one modality contains

information about an object of interest, its information is displayed to the user regardless of the statistical properties of image regions containing it. In the absence of objects of interest, the modality maximizing a combination of measures defined by the user is displayed.

2.4 Region Comparison

One approach to comparing multimodal imagery is to examine the information theoretic properties of corresponding regions. For the initial experiment, mutual information is computed between corresponding regions of neighboring frames in each input video stream. Frames i and $i + 1$ are divided into square tiles of $2^n \times 2^n$ pixels. For each tile a joint probability distribution $p(X, Y)$ and marginal probability distributions $p(X)$ and $p(Y)$ are computed from the two tiles in frames i and $i + 1$. Then, mutual information from one tile to the next is computed from

$$I(X; Y) = \sum_{y \in Y} \sum_{x \in X} p(x, y) \log \frac{p(x, y)}{p(x)p(y)} \quad (1)$$

The modality with minimum mutual information (or, equivalently, maximum information gain) represents the most informative source video for a given location and time. Other measures, including variance, range of intensities, or

contrast of regions can be computed, but such measures are not very meaningful when comparing across modalities.

2.5 Object Detection

For detecting humans, an adaptively boosted cascade of Haar-like classifiers is used. The implementation is similar to Viola's [8]. From a large bank of simple, wavelet-like filters, a supervised training process adopts as small number of filters that can recognize human shapes. Each filter is scale-invariant, using a technique called Integral Images, whereby a simple summation across intensities in the image domain reduces the computation of filter responses to the constant time. The filters are applied one at a time to regions of interest that have been hypothesized to have humans, and any negative responses cause the classifier to reject the hypothesis. If all filters yield a positive response, then the area is classified as positive for human presence.

Because the classifier can be applied rapidly in many different scales and positions in each frame of a video, the classifier is applied independent of the tiles examined by the region comparison subsystem. The results of such a classifier are returned as bounding rectangles for regions containing object presence with a confidence at or above a certain threshold.

The classifier is currently applied to only the visible spectrum. The consequence of this limitation is examined in Section 3.2.

2.6 Image Fusion

The result of region comparison and human detection processes is a grid of stream selection or blending values across the image domain. The final representation of these values is dictated by the image fusion rules. Originally, the fusion was to be accomplished by blending or selecting the input streams according to their relative usefulness as given by the region comparison subsystem, then highlighting the humans in the output.

Because the meaning of an intensity value may be different in different modalities, the same object may be represented with very different intensities in different video streams. As a result, fusion by intensity blending was replaced with a wavelet-based fusion scheme that combines intensities based on spatial frequencies in images, as detailed in Section 3.4.

3. Experimental Results

3.1 Mutual Information

Figure 2 shows heatmaps of the mutual information in corresponding frames from the Octec dataset. In frames containing smoke, the regions corresponding to the smoke are predictably uniform, and their relative usefulness is small. As a result, the regions of the heatmap with high values represent regions of the image domain where the

visible modality is preferred by the system, but regions with low values represent regions where the infrared modality is selected.

The information content of each modality can be modeled as a Gaussian distribution whose mean and variance can be estimated. By normalizing the information content, a weight factor is applied to control the relative a-priori importance of certain cameras. The mutual information measure is made more reliable by adding a linear combination of other statistical features, yielding a more general usefulness measure.

3.2 Foreground Motion Analysis

Background subtraction is commonly applied to both object detection and object tracking. If the system is to allow camera motion, the video frames must be registered at the background to enable background subtraction. Because consecutive frames can contain translation and rotation, with negligible scale, a simple registration can be applied using the Fourier Transform-based method of Reddy and Chatterji [7].

By reducing translation, rotation, and scale differences to phase differences in the Fourier domain, the differences are estimated from the cross power spectrum of the transformed frames. By registering neighboring frames, a long sequence of images is resampled into one common background for analysis.

Inaccuracies in registration produce two types of artificial motion: 1) motion noise due to sub-pixel inaccuracy, and 2) high-frequency environmental motion, such as the movement tree leaves and branches by wind. Highly complicated scenes will have more noise of both types than simple scenes due to the abundance of sharp intensity changes and the presence of motion in the background. This problem is solved by finding the average motion at each pixel and using that information to distinguish object motion from the motion noise caused by subpixel registration error and motions of trees, bushes, and grass (Figure 3).

3.3 Human Presence Detection

A small training dataset is insufficient for recognizing humans in all situations. Datasets for the adaptively boosted cascade classifier are subject to many potential problems if the dataset does not contain both a variety of poses and a variety of lighting conditions related to the expected target environment. Insufficient training datasets exhibit overfitting. The datasets may also exhibit high sensitivity to changes in pose (especially the changes caused by unusual target behavior, such as climbing, leaping, running, or crawling, which are not frequently addressed in pedestrian datasets). The datasets may also exhibit a low power of discrimination, which may result in a large number of false positives centered on tall, thin objects.

While more plentiful sources of data play a large part in determining the adequacy of the training procedure for

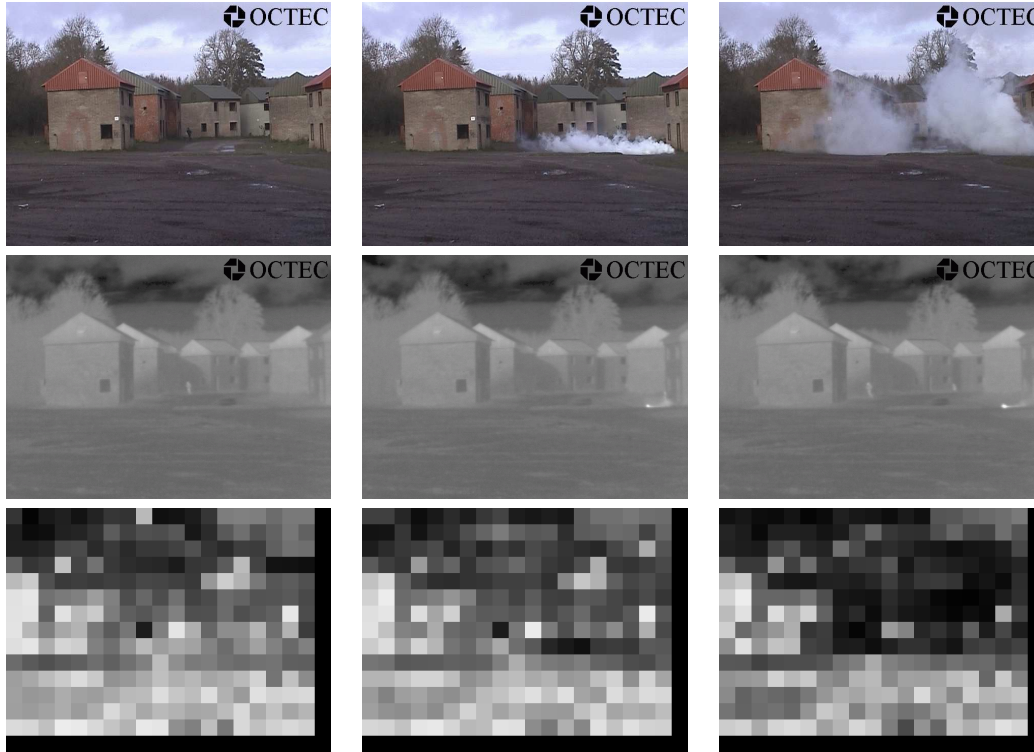


Fig. 2: Heatmaps of mutual information in the Octec dataset.

human detection, the problem of insufficient discriminatory power may be partly addressed by limiting the regions considered by the classifier to those that exhibit unusual motion, as described in Section 3.2. By determining the intersection of bounding boxes produced by the human classifier with the areas of highly unusual motion highlighted by the motion analysis subsystem, the contours representing human motion may be exposed and displayed, as in Figure 4. This novel approach provides two benefit: it reduces the false positives created by the original classifier, and emphasizes the object's outline as it moves across the image domain.

3.4 Discrete Wavelet Fusion

Originally, a fusion method based on intensities was developed. Given the tilewise mutual information in stream i at pixel location (x, y) as $I_i(x, y)$, a stream selection function is defined as

$$f_i(x, y) = \begin{cases} 1 & I_i(x, y) = \min_j I_j(x, y) \\ 0 & \text{otherwise} \end{cases} \quad (2)$$

By representing these functions as images, and performing Gaussian blurring with a standard deviation proportional to the size of the selection tiles, a blending function is obtained which emphasizes one particular source at the center of each tile, but blends between the streams at selection boundaries.

In many cases, merely blending the intensities of multi-modal imagery is insufficient. Even after histogram manipulation, visible and IR video streams are too dissimilar for a weighted addition of intensities to be helpful to an operator as shown in Figure 5.

A more effective fusion is achieved by wavelet fusion. While, in general, wavelet fusion is a global approach, by proper manipulation of the wavelet coefficient, local effects are achieved.

Certain image properties can be emphasized by transforming images by Discrete Wavelet Transformation (DWT), manipulating the obtained coefficients, and inverting the transformation (Figure 6). Particularly, contrast of the fused video output can be adjusted for display.

By adjusting the low-frequency coefficients of an image, its global intensity can be increased or decreased. By adjusting the high-frequency coefficients, an image's contrast can be amplified. Similarly, adding the wavelet coefficients of two images produces an image containing contours and regions belonging to both upon inversion of the transform (Figure 7). This highlights objects not visible in the intensity-based fusion results, such as the silhouette of the person concealed by the smokescreen.

During processing of an image by DWT, decreasing-frequency coefficients are generated at power-of-two scales in various known locations of the image domain (Figure

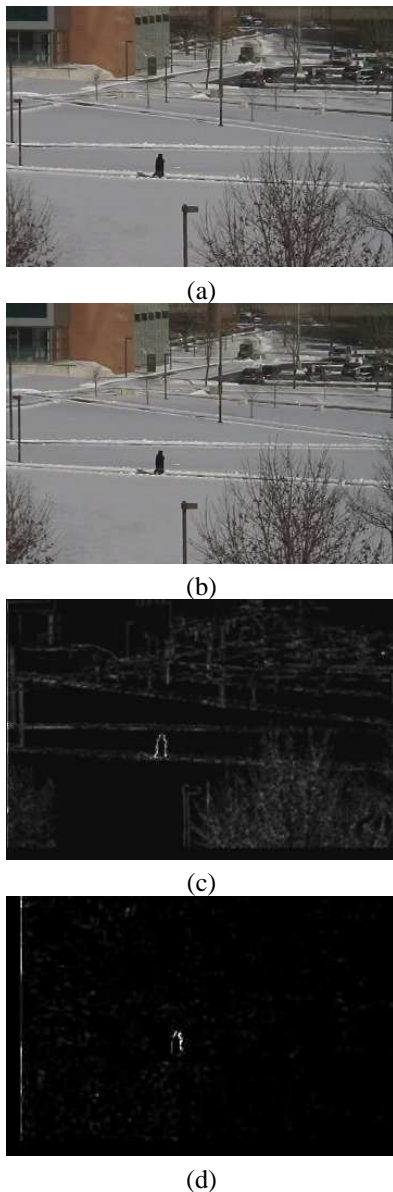


Fig. 3: (a), (b) Two consecutive frames in a video, showing a pedestrian. (c) Background subtraction of the frames. (d) Detected motion after suppressing motion noise.

8). This allows each region to be adjusted in wavelet space, despite the separation of different scales of wavelet coefficients.

In this manner, local results of region comparison is kept local. Our future research includes determination of descriptive rules that can effectively guide fusion of multimodality images for viewing and automated analysis.

4. Summary

A system for combining video imagery from multimodal sources is proposed. Figure 9 represents the pipeline cur-

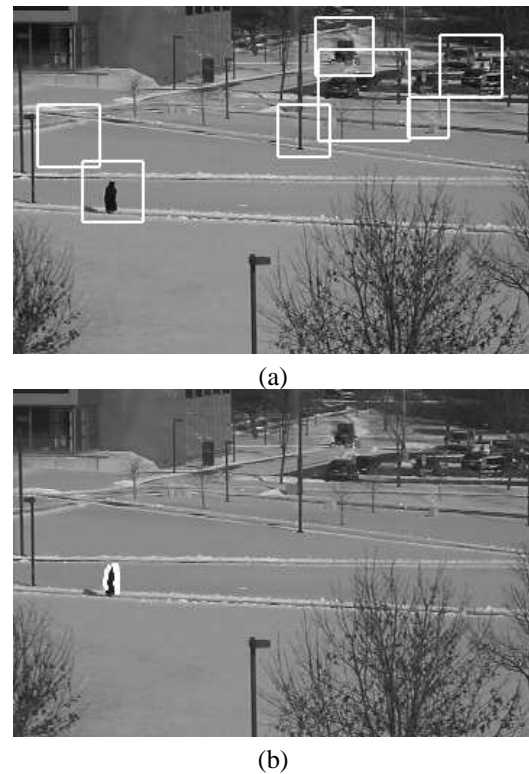


Fig. 4: (a) Human detection with a poorly-trained classifier. (b) Human detection when aided by motion analysis.

rently employed by the system. The region comparison subsystem produces both a selection grid aligned to regions of wavelet coefficients in the fusion process and de-noised motion for use by the human detection subsystem. The human detection subsystem produces a set of unaligned bounding boxes containing hypothesized humans, adding to the selection grid guided by wavelet coefficient manipulation. After video frames are decomposed by DWT, the selection grid and human presence values are used to produce a set of combined wavelet coefficients, which are then inverted to produce a fused output video.

Experimental results are shown demonstrating region comparison and object detection using motion features. A purely information theoretic approach to the comparison of video streams is made more flexible by adding image statistics and classification to the process. Insufficient classification power in object detection is mitigated via foreground motion analysis.

5. Conclusions and Future Work

There are three challenges to overcome in this project: 1) comparison of corresponding regions in dissimilar modalities, 2) classification of objects of interest, some of which may be concealed in one or more input streams, and 3)

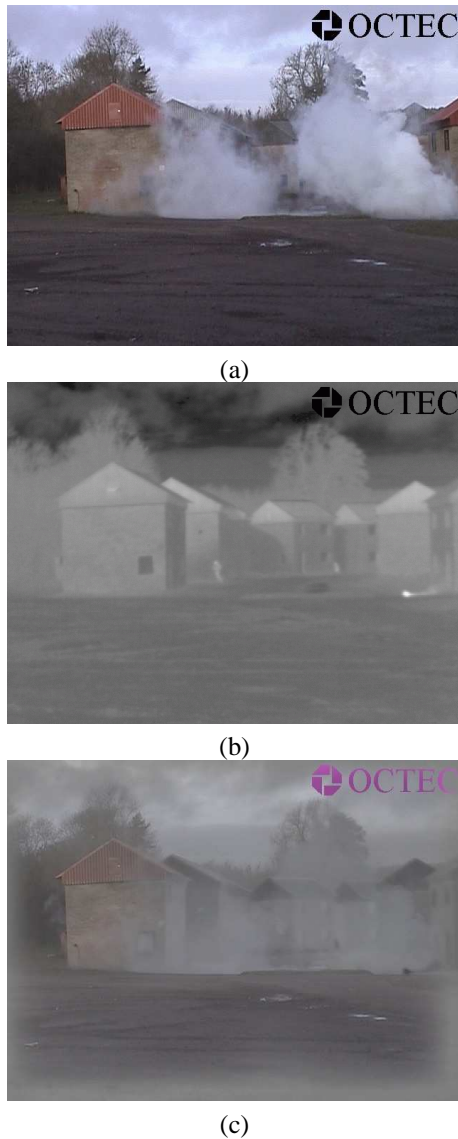


Fig. 5: The results of an intensity-based fusion.

fusion of the results in a form that is easily interpreted by a human operator.

Comparison of corresponding regions is achieved through information theoretic measures. It may be useful to add motion statistics to the region comparisons, giving the usefulness measure a temporal dimension. Classification of objects of interest according to unusual motion features and a cascade of Haar-like classifiers is also believed to be promising. The main difficulty will be in compiling a training dataset that is both sufficiently large to allow for accurate classification and diverse enough to address people in various poses of interest.

Fusion of multimodality video streams is another challenging problem that has been examined in the past. The challenges in our problem have not been adequately ad-

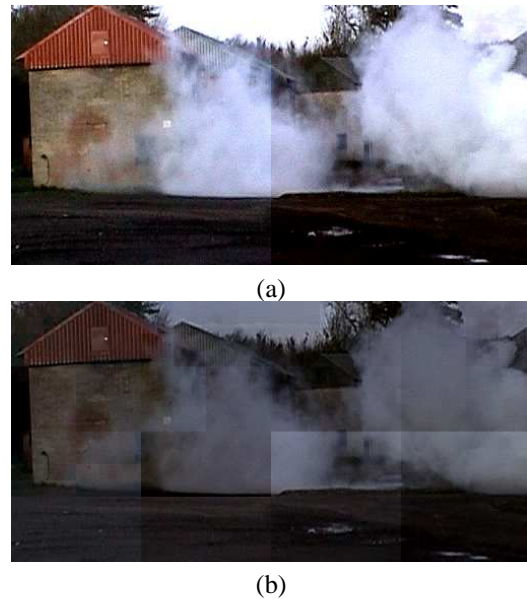


Fig. 6: Intensity and contrast adjustment via coefficient modification.

dressed in the past. We believe the wavelet-domain fusion should be modified to better adapt to the problem at hand. We expect that as the classification and region comparison procedures become more sophisticated, it will become possible to not only fuse information across modalities, but it will become possible to present gesture and human motion to an end user in an effective manner to allow quick responses to observed events.

Acknowledgements

The authors would like to thank the Air Force Research Laboratory (AFRL) and the Dayton Area Graduate Studies Institute (DAGSI) for partial support of this project. The authors also would like to thank Dr. Vincent Schmidt at AFRL for insightful comments and guidance.

References

- [1] D. Culibrk, M. Mirkovic, V. Zlokolica, M. Pokric, Vladimir S. Crnojevic, and D. Kukolj. Salient motion features for video quality assessment. *IEEE Transactions on Image Processing*, 20(4):948–958, 2011.
- [2] G. Galdi, A. Prati, and R. Cucchiara. Covariance descriptors on moving regions for human detection in very complex outdoor scenes. *2009 Third ACM/IEEE International Conference on Distributed Smart Cameras ICDSC*, (102):1–8, 2009.
- [3] H-X Jia and Y-J Zhang. Fast human detection by boosting histograms of oriented gradients. In *Proceedings of the Fourth International Conference on Image and Graphics, ICIG '07*, pages 683–688, Washington, DC, USA, 2007. IEEE Computer Society.
- [4] J J Lewis, R J O Callaghan, S G Nikolov, and D R Bull. Region-based image fusion using complex wavelets. *Image Rochester NY*, 8(2):119–130, 2007.
- [5] C-J Pai, H-R Tyan, Y-M Liang, H-Y Mark Liao, and S-W Chen. Pedestrian detection and tracking at crossroads. *Pattern Recognition*, 37(5):1025–1034, 2004.

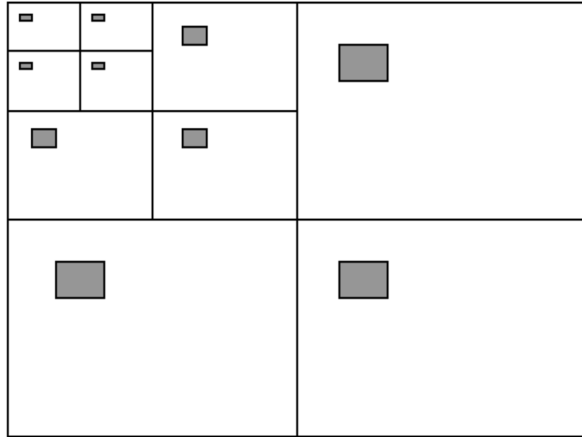


Fig. 8: An example of corresponding regions in the wavelet space.

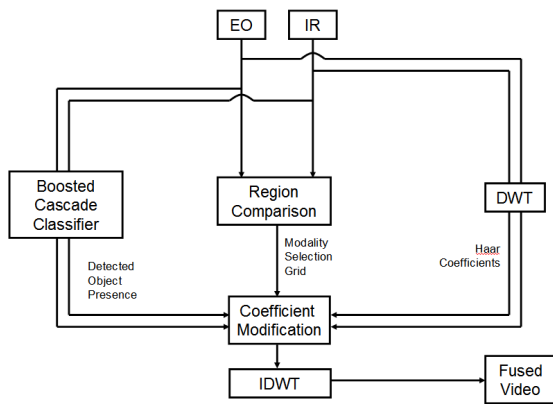


Fig. 9: The proposed motion detection and image fusion system.



(a)



(b)



(c)

Fig. 7: The results of a DWT-based fusion.

- [6] N. D. Rasmussen, B. S. Morse, M. A. Goodrich, and D. Eggett. Fused visible and infrared video for use in wilderness search and rescue. In *WACV*, pages 1–8. IEEE Computer Society, 2009.
- [7] B. S. Reddy and B. N. Chatterji. An FFT-based technique for translation, rotation, and scale-invariant image registration. *IEEE Transactions on Image Processing*, 5(8):1266–1271, August 1996.
- [8] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, Hawaii, 2001.
- [9] Z. Zhang and R. S. Blum. Region-Based Image Fusion Scheme For Concealed Weapon Detection. In *Proceedings of the 31st Annual Conference on Information Sciences and Systems*, pages 168–173, 2002.

A Contagion Threat on a Social Network: A Graphical Approach

F. Ciardiello¹, J. Binner², V. Schmidt³

¹ Management school, University of Sheffield, Sheffield, England, UK

² Management school, University of Sheffield, Sheffield, England, UK

³ Air Force Research Laboratory, Wright Patterson Air Force Base, Dayton, OH, US

Abstract— We propose an algorithm for computing main α -stable sets introduced by (Ciardiello and Di Liddo, 2009) on coalitional games modeled through a directed pseudograph. We give a visualization of computed solutions in complex game theoretical situations. The algorithm is based upon a graph traversing method exploring extended paths minimal in coalitions. Computation of stability concepts in coalitional games in effectiveness form is of key importance in the detection of fake players in social media.¹

Keywords: Algorithmic Game Theory, Data Representation, Stable Sets, Graph Theory.

1. Computational Cooperative Game Theory: A Graph Approach

Coalitional game theory focuses on what groups of players can achieve rather than on what individual players can do. They were introduced in seminal papers [2], [3], [4]. A key question in coalitional game theory is that of coalitional stability, usually captured by the notion of the *core*— the set of outcomes such that no subgroup of players has an incentive to deviate. Here we present a theoretical stability solution concept associated with farsighted behavior of players. Most classical solutions for cooperative games prescribe myopic choices. Many applications of farsighted stability solutions have been discussed in routing theory [5], competitive markets [6], international agreements [7], voting theory [8].

In many social situations, we observe the formation of independent and sometimes competing groups, teams, clubs, cooperatives (coalitions for short) each of them persecuting the same goal whose benefit (or cost) is exogenously divided between members of the group. The interdisciplinary aspects between coalitional game theory and computing theory is highly motivated by several scholars (see [9], [10], [11], [12]). To compute a stability concept in the framework of a cooperative game has an interesting appeal in perturbed social frameworks. Anomalous players can have different maximization problems with respect the rest of the community; they can have different behavioral assumptions when they decide to join a coalition. Therefore, to study the stability solution concept can have an impact on a more deep definition of resilience index of a structured

graph social situation. Apart from well known computational problems, approaching cooperative games from a visualized graph approach (endowed with underlying spatial networks) makes these problems interesting in terms of Representation of Data and their Geo Location. Loosely speaking we refer to concrete situations where players are connected in terms of a communication graph but they are topologically closed with respect to spatial distances.

We discuss our computation for some *input data* or *games* whose complexity is particularly high in terms of interactions between outcomes or coalition structures. In Section 4 this complexity is represented by

- multiple social loop situations
- same dynamics among coalition structures caused by different coalitions' moves.

We think that our approach is quite original because not much attention has been devoted to this class of cooperative games in terms of computation of farsighted solution concepts. At the same time our basic way to face this class of problems open the way to study new algorithmic choices to define theoretical solution concepts incorporating realistic assumption. For instance tractability of the computation of a stability concept implies bounded rationality of players which is quite often disregarded in theoretical formulation.

Against a plethora of algorithms traversing graphs (iteration of adjoint matrix of a graph), our independent algorithmic choices to compute our stability solution is mainly associated with situational detail in games in effectiveness form: the existence of coalitions labeling edges of graphs. Most solution concepts in games in effectiveness form have as *input data* a labelled graph they return as *output data*: a set of nodes or a subset of sets of nodes as shown in [13], [14] or paths as showed in [15]. In our case visualized output data is a subset of sets of nodes.

1.1 Coalitional games in Effectiveness Form

We illustrate our coalitional game-theoretic environment pointing out various concepts of farsighted dominance between coalition structures and the related stability notion. Mathematically speaking, a coalitional game in effectiveness form can be represented in such a way

$$\mathfrak{D} = (\mathcal{N}, \mathcal{Z}, \{\prec_i\}_{i \in \mathcal{N}}, \{\rightarrow_S\}_{S \subseteq \mathcal{N}, S \neq \emptyset}, \alpha)$$

where

¹This Research Report Paper has been inspired by [1].

- \mathcal{N} is the set of players; a nonempty subset $S \subseteq \mathcal{N}$ is called *coalition*.
- \mathcal{Z} is a subset of all the partitions of \mathcal{N} whose elements are disjoint coalitions. Any element of \mathcal{Z} is called *outcome* or *coalition structure*;
- \prec_i is a strong preference relation defined on \mathcal{Z} associated to player i ;
- \rightarrow_S is a reflexive relation defined on \mathcal{Z} associated to $S \subseteq \mathcal{N}$.
- $\alpha \in \mathbb{R}^{\mathcal{N}}$ is a vector measuring the safety/risk level of any player.

The family of preference relations $\prec := \{\prec_i\}_{i \in \mathcal{N}}$ can be replaced by a value function $\mathcal{V} : \mathcal{Z} \rightarrow \mathbb{R}^{\mathcal{N}}$ whose component $\mathcal{V}_i(a)$ denotes the payoff obtained by player i if the coalition structure a is formed. In addition, the family $\{\rightarrow_S\}_{S \subseteq \mathcal{N}}$ is called an *effectiveness relation* of \mathcal{D} . The game is played in the following manner: when the game begins, there is a status quo outcome, say a . If the members of a coalition S decide to change the status quo to an outcome b , then the status quo becomes b . This means $a \rightarrow_S b$. From this new status quo b , other coalitions might move to c (i.e. $b \rightarrow_T c$) through T and so forth. If the game reaches an outcome from which no coalition moves, the game ends and this outcome has to be necessarily considered stable. Step by step, any player i could prefer an outcome to another one by using his preference relation \prec_i or his value function \mathcal{V}_i . All actions are public. If $a \prec_i b$ for all $i \in S$, we write $a \prec_S b$. In Figure 1 an example of a coalitional game in effectiveness form is given where $a = \{\{1, 2\}, \{3\}\}$, $b = \{\{1\}, \{2, 3\}\}$, $c = \{\{1\}, \{2\}, \{3\}\}$, $d = \{\{1, 2, 3\}\}$ are coalition structures. Formation of the coalition $\{\{1, 2\}, \{3\}\}$ means that players 1, 2 are in the same coalition while player 3 forms a singleton.

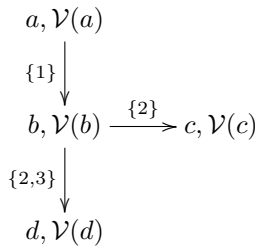


Fig. 1: An example of game in effectiveness form with 3 players and 4 outcomes.

Chwe introduces a new dominance relation in a seminal paper [16]. In such way, an outcome b is said to *dominate indirectly* another outcome a if b can replace a in a sequence of moves such that at each move the active coalition S_i prefers b to the alternative outcome a_i it faces at that stage. The indirect dominance captures the idea that coalitions can anticipate other coalitions' actions.

Definition 1.1 (Chwe): Let a, b two outcomes in \mathcal{Z} . We say that a is indirectly dominated by b , or $a \ll b$, if there exists a sequence of outcomes $a = a_0, a_1, \dots, a_m = b$ and a sequence of coalitions S_0, S_1, \dots, S_{m-1} such that $a_i \rightarrow_{S_i} a_{i+1}$ and $\mathcal{V}_j(a_i) < \mathcal{V}_j(b)$ for $j \in S_i$, $i = 0, 1, 2, \dots, m-1$.

Some necessary definitions from [17] are now presented about stability solution concepts on coalitional games in effectiveness form.

Definition 1.2 (path-believable farsighted dominance): Given $a, b \in \mathcal{Z}$, we say that a is believable-path dominated by b if and only if there exists a chain of coalitions $S_0, S_1 \dots S_{m-1}$ and a chain of outcomes $a_0, a_1 \dots a_m$ such that $a = a_0 \rightarrow_{S_0} a_1 \rightarrow_{S_1} a_2 \dots a_{m-1} \rightarrow_{S_{m-1}} a_m = b$

$$B^h \subseteq P_{h+1} \quad h = 0 \dots m-1 \quad (1)$$

where

$$B^0 = \{c \in \mathcal{Z} \mid b \ll c\} \neq \emptyset$$

$$\text{if } \{c \in \mathcal{Z} \mid b \ll c\} = \emptyset, B^k = \{b\} \quad k = 0 \dots m-1. \quad (2)$$

$$B^h = \{c \in \mathcal{Z} \mid k \ll c \quad \forall k \in B^{h-1}\} \neq \emptyset \quad h = 1 \dots m-1$$

orlet h the smallest integer such that

$$\{c \in \mathcal{Z} \mid k \ll c \quad \forall k \in B^{h-1}\} = \emptyset, \quad (3)$$

$$B^k = B^{h-1} \quad k = h \dots m-1. \quad (4)$$

$$P_h = \{c \in \mathcal{Z} \mid \mathcal{V}_i(c) > \mathcal{V}_i(a_{m-h}) \quad \forall i \in S_{m-h}\} \quad h = 1 \dots m.$$

We denote it by $a \ll_{\text{pbf}} b$.

Definition 1.3 (pbf α -Stable Set): Let $z \in \mathcal{Z}$, we denote by $D(z) = \{x \in \mathcal{Z} : z \ll_{\text{pbf}} x\}$. A subset $Y \subseteq \mathcal{Z}$ is path-believable farsightedly α -stable if for any $a \in Y \forall S, d$ such that $a \rightarrow_S d$, we have that

- if $d \notin Y$ and $D(d) \cap Y = \emptyset$;
- if $d \in Y$ and $D(d) \cap Y = \emptyset$ and $\exists i \in S : \mathcal{V}_i(a) \geq \mathcal{V}_i(d)$;
- if $D(d) \cap Y \neq \emptyset$. For each $e \in D(d) \cap Y$:

$$\exists i \in S \quad \mathcal{V}_i(a) \geq (1-\alpha_i) \min_{b \in F_{d,e}} \mathcal{V}_i(b) + \alpha_i \max_{b \in F_{d,e}} \mathcal{V}_i(b);$$

We recall a particular class of stable sets.

Definition 1.4 (Main pbf α -Stable Sets): A subset $Y \subseteq \mathcal{Z}$ is a main α -stable set if and only if

- Y is a α -stable set;
- each $Y' \subseteq \mathcal{Z}$, $Y \subset Y'$ is not a α -stable set.

Therefore, in this paper we provide an high overview of an algorithm for computing main α -stable sets based upon a graph traversing method and we provide some visualization of games. Our merit is to give an algorithm for the computation of these solutions in coalitional game in effectiveness form.

In Section 2 games in effectiveness form are modeled through a non-empty directed pseudograph, together with the representation of farsighted dominancies and the associated

² B^h are sometimes denoted by $B_{a,b}^h$ throughout the paper because they depend on ordered pair (a, b) .

stability concept: our solution approach embeds Chwe's algorithm for computing the largest consistent set. In Section 3, the algorithm is illustrated, followed by a discussion on some computational issues. Section 4 illustrates the visualization of data and related practical results obtained by the algorithm when applied to some examples of effectiveness coalitional games. In these games directed pseudographs are meaningful with respect to peculiar graph-traversing issues, in order to outline some positive aspects of our algorithm. In Section 5, some conclusions and further research trends are discussed.

2. Representation of Data

In our work a game is modeled through a non-empty directed pseudograph³ $\mathcal{D} = (\mathcal{Z}, \mathcal{E})$ [18] (simply named *digraph*) where $\mathcal{E} = \bigcup_{S \subseteq \mathcal{N}} \rightarrow_S$. Say $e \in \mathcal{E}$ if a coalition S exists such that $e \in \rightarrow_S$: S is the *labelling coalition* of e . Digraph's nodes are $z \in \mathcal{Z}$ and edges are $e \in \mathcal{E}$. Let $e = (a, b)$; denote with $\eta(e)$ its head a and $\tau(e)$ its tail b . We give some useful definitions for clarity.

Definition 2.1 (Walk, Trail, Path, k -Cycle, Loop, Cycle): A walk of length $k \geq 2$ is a non-empty sequence of edges $e_0 e_1 \dots e_{k-1}$ where $\tau(e_j) = \eta(e_{j+1})$ $j = 0..k-2$. A trail $e_0 e_1 \dots e_{k-1}$ is a walk such that $0 \leq i, j < k$, $i \neq j \Rightarrow e_i \neq e_j$. A path $e_0 e_1 \dots e_{k-1}$ is a trail such that $0 \leq i, j < k$, $i \neq j \Rightarrow \eta(e_i) \neq \tau(e_j)$. A k -cycle is a walk $e_0 e_1 \dots e_{k-2} e_{k-1}$ such that $e_0 e_1 \dots e_{k-2}$ is a path, $\eta(e_0) = \tau(e_{k-1})$ and whose length is $k \geq 2$. A loop is an edge e where $\tau(e) = \eta(e)$. A cycle is a k -cycle or a loop; it is of length k if a k -cycle, or 1 if a loop.

Roughly speaking, in a trail no edge can be repeated. Note that $0 \leq i, j < k$, $i \neq j$ implies $\eta(e_i) = \tau(e_j)$ can be true in a trail.

Definition 2.2 (e-path): An e-path (extended path) $e_0 e_1 \dots e_{k-1}$ is a trail such that $\eta(e_i) = \tau(e_j)$ $0 \leq i < j \leq k-1$ implies $j = k-1$.

Note that a path or cycle is an e-path, and an e-path can include a cycle.

Definition 2.3 (Deviation, Sink): Let $z_i \in \mathcal{Z}$. We denote $\Delta_i = \{e \in \mathcal{E} \mid \eta(e) = z_i\}$. Its elements are called *deviations* from z_i . A sink is a node z_i such that $\Delta_i = \emptyset$.

Definition 2.4 (em-path): An e-path $p = e_0 e_1 \dots e_{k-1}$ with corresponding labelling coalitions $S_0 S_1 \dots S_{k-1}$ is an em-path (extended path minimal in coalitions) if another e-path $\bar{p} = \bar{e}_0 \bar{e}_1 \dots \bar{e}_{k-1}$ with corresponding labelling coalitions $\bar{S}_0 \bar{S}_1 \dots \bar{S}_{k-1}$ does not exist such that $\eta(e_i) = \eta(\bar{e}_i)$, $\tau(e_i) = \tau(\bar{e}_i)$ and $\bar{S}_i \subset S_i$ for some $i = 0 \dots k-1$.

Let $z_i, z_j \in \mathcal{Z}$. Note that there may be multiple em-paths joining z_i and z_j . Given an e-path p between z_i and z_j , a unique em-path p' joining z_i and z_j exists such that each labelling coalition of p' is included in the corresponding

labelling coalition of p : p' is the minimal em-path associated with p . Discovering em-paths instead of e-paths improves the efficiency of our algorithm because this does not alter dominancies and consequently the main α -stable sets are the same in both approaches. In fact, if there is a *pbf* dominance between z_i and z_j along an em-path p then the same dominance exists along any e-path q such that p is the minimal em-path associated with q .

2.1 High Level Data Structures

Let

- P_i the set of all em-paths with head z_i
- $P_{i,j}$ the set of all em-paths with head z_i and tail z_j
- $P = [P_{i,j}]$ the connectivity matrix
- n the number of players of the game
- \mathcal{N} the set of players
- m the number of outcomes = graph order $||D||$

Note that the connectivity matrix P is an extension of the classical adjacency list representation $A = [a_{ij}]$ of a digraph, in which a_{ij} is the number of edges from z_i to z_j . and the following parameters, derived from above:

- $M_P = \sum_{k=1}^m k \binom{n}{\lfloor \frac{n}{2} \rfloor}^k$: the upper bound of $|P_i|$ over z_i
- $M_L = m$: the number of effectiveness relation's elements (edges) in the longest em-path
- $M_W = M_L \cdot m \cdot \binom{n}{\lfloor \frac{n}{2} \rfloor}$: the "deepness" of the working stack (see Appendix)
- $M_S = \binom{m}{\lfloor \frac{m}{2} \rfloor}$: the upper bound on the total number of main α -stable sets found.

The data structures representing graph components are subsequently described. These data structures are modeled after the aforementioned constants and parameters. IDM is a matrix whose generic element $IDM_{i,j}$ is true if $z_i \ll z_j$, false otherwise. $BDM[b]$ is a matrix whose generic element $BDM[b]_{i,j}$ is true if $z_i \ll_b z_j$ $b \in \{pbf, wpbf, w^*pbf, lpbf, lwpbf, lw^*pbf\}$.

3. The Algorithm

The algorithm is based upon the following input data:

- \mathcal{Z} : the set of the m outcomes (graph nodes) of the game;
- $\mathcal{V} \in \mathbb{R}^{m \times n}$: the m real-valued payoffs' vectors, one for each outcome;
- $\alpha \in \mathbb{R}^n$: the vector of safety/risk values, one for each player;
- $\mathcal{E} \subseteq \mathcal{Z} \times \mathcal{Z}$: the set of all graph edges.

In our algorithm the hard part is to discover all e-paths linking outcome pairs: this is done with a constructive approach, exploring existing deviations from nodes until a sink or a cycle is found. In general, a directed pseudograph representing an effectiveness coalitional game is not a tree, so it has been necessary to make use of a slightly modified

³In which nodes can be linked through multiple edges and loops are admitted.

stack data structure for “discovering” any bifurcation starting from a node. A better approach consists of exploring the only deviations labelled by coalitions which are “minimal”, in the sense that they are included in coalitions of some other parallel (i.e. with same head and tail) edge, if it exists. The algorithm proceeds starting from the “construction” of the connectivity matrix P , then the dominance matrices are built; finally, all main α -stable sets S_α^* are generated. A high-level overview of the algorithm is thus:

- I. Build the connectivity matrix P
- II. Build the indirect dominance relation matrix IDM
- III. Build the farsighted dominance matrices BDM
- IV. Compute the main α -stable sets S_α^*

The algorithm is written according to different dominance modalities $b \in \{pbf, wpbf, w^*pbf, lpbf, lwpbf, lw^*pbf\}$ which generate different $pbf, wpbf, w^*pbf, lpbf, lwpbf, lw^*pbf$ α -stable sets. See [1] for a complete description. In Section 1.1 we introduce only pbf dominance and the related pbf α -stable sets and main pbf α -stable sets. Here we decide to maintain the structure of the original algorithm and we invite the reader to read this section just in case $b = pbf$. We think that this detail does not affect the scientific aim of this work; at the same time it shows a broad area of applications of our algorithm. The detailed articulation of the algorithm is subsequently described.

- I. Build the connectivity matrix P
 1. **for each** node (outcome) $z_i \in \mathcal{Z}$
 2. find all em-paths with initial node z_i
 3. find and store in $P_{i,j}$ all em-paths between z_i and $z_j, j = 1 \dots m$
 4. **next** z_i
- II. Build the indirect dominance matrix IDM
 1. **for each** $i, j = 1 \dots m$
 2. **if** $z_i \ll z_j$ **then** $IDM_{i,j} := true$ **else** $IDM_{i,j} := false$ **endif**
 3. **next** j, i
- III. Build the farsighted dominance matrices $BDM[b]$
 1. **for each** $b \in \{pbf, wpbf, w^*pbf, lpbf, lwpbf, lw^*pbf\}, i, j = 1 \dots m$
 2. build the initial backward induction set $B_0 = \{z_k : z_j \ll z_k\}$
 3. $BDM[b]_{i,j} := false$
 4. **for each** $p = e_0 e_1 \dots e_h \in P_{i,j}$
 5. $\Theta_{i,j,p}[b] := B_0, BDM_{i,j,p}[b] := true$ (hypothesize $z_i \ll_b z_j$ along p)
 6. **for each** r from h down to 0
 7. **if** $b \in \{wpbf, w^*pbf, lwpbf, lw^*pbf\}$ **then**
 8. $BDM_{i,j,p}[b] := false$ (reset hypothesis)
 9. **if** $b \in \{w^*pbf, lw^*pbf\}$ **then** $\Theta_{i,j,p}^*[b] := \emptyset$ **endif**
 10. **endif**
 11. Set $BDM[b]_{i,j,p}$ and $\Theta_{i,j,p}^*$ after the

appropriate b dominance criterion

12. **if** $BDM_{i,j,p}[b] = false$ **then** exit loop r **endif**
13. **if** $r > 1$ **then** rebuild $\Theta_{i,j,p}[b]$ **endif**
14. **next** r
15. **if** $BDM[b]_{i,j,p} = true$ **then** $BDM[b]_{i,j} := true$ **endif**
16. **next** p
17. **next** j, i, b

- IV. Compute the main α -stable sets $S_\alpha^*[b]$
 1. **for each** $b \in \{pbf, wpbf, w^*pbf, lpbf, lwpbf, lw^*pbf\}$
 2. Let $X \supseteq F_{[b]}(X, Y) \rightarrow 2^{\mathcal{Z}}, X, Y \in 2^{\mathcal{Z}}$ the stability test function
 3. **if** $F_{[b]}(\mathcal{Z}, \mathcal{Z}) = \mathcal{Z}$ **then** $\text{exit}(S_\alpha^*[b] := \mathcal{Z})$ **endif**
 4. $Q = \mathcal{Z} \setminus F_{[b]}(\mathcal{Z}, \mathcal{Z})$ (isolate bad outcomes, i.e. outcomes not in the initial stable set)
 5. $\bar{Q} = \emptyset$ (\bar{Q} will be the set of all $z \in Q$ for which $\mathcal{Z} \setminus Q \cup \{z\}$ is $F_{[b]}$ -stable)
 6. **for each** $z \in Q$
 7. **if** $F_{[b]}(\{z\}, \mathcal{Z} \setminus Q \cup \{z\}) = \{z\}$ **then** $\bar{Q} := \bar{Q} \cup \{z\}$ **endif**
 8. **next** z
 9. **if** $\bar{Q} = \emptyset$ **then** $\text{exit}(S_\alpha^*[b] := \mathcal{Z} \setminus Q)$ **endif**
 10. **for each** $A \in 2^Q$ such that A is maximal (search for all next possible stable sets)
 11. **if** $F_{[b]}(A, \mathcal{Z} \setminus Q \cup A) = A$ **then** $\text{return}(\mathcal{Z} \setminus Q \cup A)$ **endif**
 12. **next** A
 13. **next** b
 14. **exit**

We now illustrate the main computational issues related to the algorithm.

Step I.

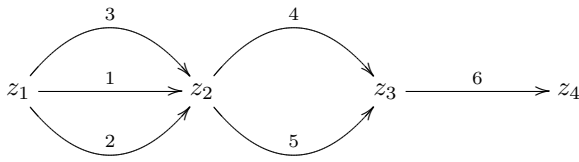
In line 1 all em-paths starting from current node z_i are explored and stored in P_i . If current node z_i is a sink there is no further processing to do for this step, otherwise \mathcal{E} is scanned until all edges with head z_i have been examined. Let $\bar{\Delta}_i$ a subset of Δ_i such that the labeling coalitions of its edges are minimal (see subsection 2). All deviations in $\bar{\Delta}_i$ are thus found and stored for subsequent processing in LIFO (Last In First Out) mode [19].

Let $e \in \mathcal{E}$; $\tau(e)$ denotes the tail of edge e and $\eta(e)$ its head. Let $\delta_{i,j_0} \in \bar{\Delta}_i$ and

$$\delta_{i,j_0;j_1 \dots j_{r-1} j_r} := \left(\begin{array}{c} \bar{\Delta} \\ \tau \left(\begin{array}{c} \dots \Delta \\ \tau \left(\bar{\Delta}_\tau(\delta_{i,j_0}) \right) \\ \dots \\ \tau \left(\bar{\Delta}_\tau(\delta_{i,j_{r-1}}) \right) \\ \dots \\ \tau \left(\bar{\Delta}_\tau(\delta_{i,j_r}) \right) \end{array} \right) \end{array} \right)_{j_r}$$

$$\begin{array}{l}
 1 : e_{1,2}^{\{1\}} \quad 4 : e_{2,3}^{\{2\}} \\
 2 : e_{1,2}^{\{2\}} \quad 5 : e_{2,3}^{\{1\}} \\
 3 : e_{1,2}^{\{2,3\}} \quad 6 : e_{3,4}^{\{3\}}
 \end{array}
 \mathcal{V} = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}$$

where $e_{i,j}^S$ denotes an edge with labelling coalition S , head z_i and tail z_j . This game has a graph visualization with three bifurcations from a node z_1 to a node z_2 . From the last, two other edges start labeled by two different coalitions. The graph ends with a sink z_4 .



Our algorithm selects all em-paths⁶ between z_1 and z_4 :

$$P_{1,4} = \{1 \triangleright 4 \triangleright 6, 1 \triangleright 5 \triangleright 6, 2 \triangleright 4 \triangleright 6, 2 \triangleright 5 \triangleright 6\}$$

Note that there are two other e-paths $3 \triangleright 4 \triangleright 6$ and $3 \triangleright 5 \triangleright 6$ which are discarded because they are not minimal in coalitions (due to the labelling coalition of edge 3). So, our algorithm goes on computing $BDM[pbf]$ as shown in Table 4.1.⁷

$i \setminus j$	z_1	z_2	z_3	z_4
z_1		true	true	
z_2			true	
z_3				
z_4				

Table 1: $BDM[pbf]$

The unique main pbf $(.5, .5, .5)$ -stable set is $\{z_3, z_4\}$. This example shows how our algorithm works well on computing all em-paths starting from node z_1 and ending with a sink.

4.2 An example with no sink

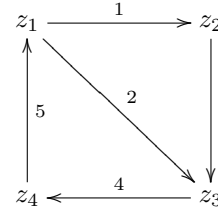
Game's input data are $n = 2, m = 4, \alpha = (0, 0)$ with effectiveness relation and payoff matrix:

$$\begin{array}{l}
 1 : e_{1,2}^{\{1\}} \quad 4 : e_{3,4}^{\{1\}} \\
 2 : e_{1,3}^{\{1,2\}} \quad 5 : e_{4,1}^{\{2\}} \\
 3 : e_{2,3}^{\{2\}}
 \end{array}
 \mathcal{V} = \begin{pmatrix} 1 & 2 \\ 2 & 3 \\ 0 & 4 \\ 4 & 1 \end{pmatrix}$$

⁶Which are strictly paths.

⁷ $BDM[pbf]_{i,j}$ is true if z_j dominates z_i on at least one path.

This game has the following graph visualization



Dominancies are presented in Table 2. The initial pbf α -stable set is $\{z_2\}$. Main pbf $(0, 0)$ -stable sets are $\{z_2, z_4\}$ and $\{z_2, z_3\}$.

$i \setminus j$	z_1	z_2	z_3	z_4
z_1				
z_2				
z_3				true
z_4	true			

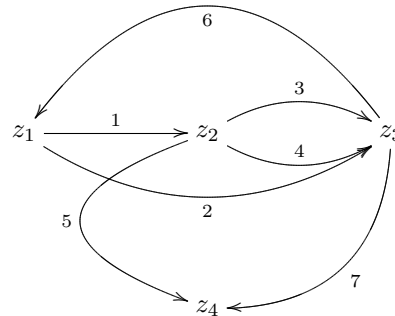
Table 2: $BDM[pbf]$

4.3 An example with multiple cycles and a sink

Game's input data are $n = 3, m = 4, \alpha = (.5, .5, .5)$, with effectiveness relation and payoff matrix:

$$\begin{array}{l}
 1 : e_{1,2}^{\{1,3\}} \quad 5 : e_{2,3}^{\{2,3\}} \\
 2 : e_{1,3}^{\{1,2\}} \quad 6 : e_{3,1}^{\{1,2\}} \\
 3 : e_{1,2}^{\{1,2\}} \quad 7 : e_{3,4}^{\{1,3\}} \\
 4 : e_{2,3}^{\{1,3\}}
 \end{array}
 \mathcal{V} = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}$$

This game has a graph visualization which has multiple 3-cycles, 2-cycles and no loop with a sink.



Our algorithm captures all the em-paths which are or contain cycles as follows: $2 \triangleright 6, 1 \triangleright 4 \triangleright 6, 1 \triangleright 3 \triangleright 6, 4 \triangleright 6 \triangleright 2, 4 \triangleright 6 \triangleright 1, 3 \triangleright 6 \triangleright 2, 3 \triangleright 6 \triangleright 1, 6 \triangleright 2, 6 \triangleright 1 \triangleright 4, 6 \triangleright 1 \triangleright 3$ and paths ending in the unique sink z_4 : $2 \triangleright 7, 1 \triangleright 5, 1 \triangleright 4 \triangleright 7, 1 \triangleright 3 \triangleright 7, 5, 4 \triangleright 7, 3 \triangleright 7, 7, 6 \triangleright 2, 6 \triangleright 1 \triangleright 5$. It enables to find out $P_{i,j}$ represented at the (i, j) entry of Table 3 as a subset of em-paths. All pbf main $(.5, .5, .5)$ -stable sets coincide with $\{z_1, z_2, z_4\}$.

	z_1	z_2	z_3	z_4
z_1	2 ▷ 6 1 ▷ 4 ▷ 6 1 ▷ 3 ▷ 6	1	2 1 ▷ 4 1 ▷ 3	2 ▷ 7 1 ▷ 5 1 ▷ 4 ▷ 7 1 ▷ 3 ▷ 7
z_2	4 ▷ 6 3 ▷ 6	4 ▷ 6 ▷ 1 3 ▷ 6 ▷ 1	4 4 ▷ 6 ▷ 2 3 3 ▷ 6 ▷ 2	5 4 ▷ 7 3 ▷ 7
z_3	6	6 ▷ 1	6 ▷ 2 6 ▷ 1 ▷ 4 6 ▷ 1 ▷ 3	7 6 ▷ 1 ▷ 5
z_4				

Table 3: Connectivity matrix P

5. Conclusions

We propose an algorithm for computing *main α -stable sets* introduced by Ciardiello and Di Liddo (2009) on coalitional games in effectiveness form modeled through a directed pseudograph. Our algorithm can be used, with slight adaptations, to compute other concepts of stability in effectiveness coalitional games, such as standard behaviour, the credible largest consistent set, the largest cautious consistent set, the stability set and the uncovered set. We finally observe that our algorithm may be further improved in the areas of e-path exploration and the construction of the connectivity matrix. These areas will be the subject of future research.

References

- [1] F. Ciardiello and C. Gallo, "A graph-traversing algorithm for computing some stable sets in effectiveness coalitional games," Tech. Rep., 2009.
- [2] R. Aumann and J. Drèze, "Cooperative games with coalition structures," *International Journal of Game Theory*, vol. 3, pp. 217–237, 1974.
- [3] R. Myerson, "Graphs and cooperation in games," *Mathematics of Operation Research*, vol. 2, pp. 225–229, 1977.
- [4] P. P. Shenoy, "On a coalition formation: a game theoretical approach," *International Journal of Game Theory*, vol. 8, pp. 133–164, 1979.
- [5] S. He, "Aspects of the bridge between optimization and game theory," Ph.D. dissertation, THE CHINESE UNIVERSITY OF HONG KONG (HONG KONG), 2011.
- [6] M. Nagarajan and G. Sosis, "Stable farsighted coalitions in competitive markets," *Management Science*, vol. 53, pp. 1–29, 2007.
- [7] A. De Zeeuw, "Dynamic effects on the stability of international environmental agreements," *Journal of Environmental Economics and Management*, vol. 55, pp. 163–174, 2008.
- [8] E. Penn, "A model of farsighted voting," *American Journal of Political Science*, vol. 53, pp. 36–54, 2009.
- [9] U. Faigle and B. Peis, "Zentrum für angewandte informatik," Universität zu Köln Weyertal 80 D-50931 Köln, Germany, Tech. Rep., 2006.
- [10] M. Wooldridge and P. E. Dunne, "On the computational complexity of qualitative coalitional games," *Artificial Intelligence*, vol. 158, pp. 27–73, 2004.
- [11] P. Herings, G. van der Laan, and D. Talman, "The average tree solution for cycle free games," *Games and Economic Behavior*, vol. 62, pp. 77–92, 2008.

- [12] F. Grafe, E. Inarra, and A. Mauleon, "An algorithm for computing the stable coalition structures in tree-graph communication games," *Sociedad de Estadística e Investigación Operativa Top*, vol. 7, pp. 71–80, 1999.
- [13] A. Mauleon and V. Vannetelbosch, "Farsightedness and cautiousness in coalition formation games with positive spillovers," *Theory and Decision*, vol. 56, no. 3, pp. 291–324, 2004.
- [14] P. Herings, A. Mauleon, and V. Vannetelbosch, "Rationalizability for social environments," *Games and Economic Behavior*, vol. 49, no. 1, pp. 135–156, 2004.
- [15] L. Xue, "Coalitional stability under perfect foresight," *Economic Theory*, vol. 11, no. 3, pp. 603–627, 1998.
- [16] M. S. Chwe, "Farsighted coalitional stability," *Journal of Economic Theory*, vol. 63, pp. 299–325, 1994.
- [17] F. Ciardiello and A. Di Liddo, "Farsighted stable sets," Dept. of Economics, Mathematics and Statistics - University of Foggia, Italy, Largo Papa Giovanni Paolo II, 1 - 71121 Foggia, Italy, Tech. Rep. 3, Feb 2009.
- [18] J. Bang-Jensen and G. Gutin, *Digraphs Theory, Algorithms and Applications*. Springer-Verlag, New York, 2008.
- [19] D. Knuth, *The Art Of Computer Programming - Fundamental Algorithms*, 3rd ed. Addison-Wesley, 1997, vol. 1.

SESSION

FORMAL METHODS, SPECIFICATION, MODELING, AND APPLICATIONS

Chair(s)

TBA

Z Formal Framework for Syntax-Based Module Level Software Metrics

Raouf Alomainy and Wei Li
 Computer Science Department, University of Alabama in Huntsville,
 Huntsville, AL 35899
 {ralomain, wli}@cs.uah.edu

Abstract

This paper introduces a framework to formalize module-level structural metrics that quantify inter-module dependencies in object-oriented software systems. We used a formal framework based on the Big Bang Graph (BBG) modelling and the formal Z specification language to formalize a modularization-based software metric as an example to demonstrate how the framework works. We have developed the Design State Space toolkit (DSS) to extract the state space represented in the formalized object-oriented model.

Keywords: Module-level structural metrics, Z formal language; design state space; software metrics tools;

Introduction

Modularization, as a feature of object-oriented programming paradigm, addresses the need for decoupled modules, each with a well-defined and advertised functionality, in order to achieve code reuse [5], separation of concerns [4], and minimize overlapping and unnecessary coupling among different modules [3]. Well-structured and modular programs are less costly to maintain than unstructured monolithic ones [2].

This paper focuses on structural metrics: quantitative measures of programs that are based on syntactic structures. Such metrics have been used to measure object-oriented programs to predict system properties such as class error probability and maintainability. An example of class-level structural metric is the Number of Children (NOC) metric [4] to calculate the number of immediate subclasses subordinated to a class in the class hierarchy in order to measure how many subclasses are going to inherit the methods of the parent class. An example of module-level structural metric is the Module Size Uniformity Index (MSUI) to assess if all modules are roughly equal in size [10]. The deviation from the module-size uniformity would generally be an indicative of a poor modularization.

Module-Level Structural Metrics

In building large systems, there is a consensus on the need for well-defined and structured architecture that is based on

modularized software components, in particular for large-scale software system [13]. However, a significant body of previous research work has often considered a module and a class to be synonymous concepts. Thus, most of the research is focused on class-level metrics and there is a lack of empirical studies on module-level metrics and their influence on the disorganization that exist today in many commercial systems [10].

In this paper, we refer to a module as a collection of classes that are grouped together to serve a purpose in a large object-oriented system.

Several common software practices contribute to the undermining of the original purpose of modularization. Some examples of such errors: a class extending another class from a different module, a class in one module is integrating, through instantiation, with a class in another module, either as an attribute or as a formal parameter in a method definition, the messaging between methods that belong to classes of different modules for local purpose functionality, etc [10]. Module-level metrics are meant to gauge well or poorly the modularity has been constructed.

In this paper, we propose a formal framework, based on the Z formal language [12], that links structural module metrics to design features in order to aid the precise definition and better understanding of module-level metrics. Z has the ability to prove properties in its specification to formally validate the metric [7]. .

Sakar and colleagues proposed module metrics that characterize large object-oriented software systems with regards to the inter-module dependencies created by associations, inheritance, and method invocations [11]. In the

remainder of this paper, we present the formalization of one of these metrics using our proposed approach.

Formal Modelling of Object-Oriented Design State Space

The formal framework is based on a new graph modelling technique, Big Bang Graph (BBG), for OO programs and the formalization of the graph model using the Z formal language. The formalized BBG model is the *design state space* of the program.

The distinguished feature of BBG is its simplicity: it uses only one graph to represent the entire design state space that we are interested in modelling structural metrics.

BBG is a colored, directed and connected graph modelled as $BBG = (E, V)$ where V is the set of colored vertices and E is the set of colored and directed edges. Vertices in BBG represent design entities such as class, object, object reference and method. Edges represent the primitive associations of the entities. BBG aims to model a program with the primitive (atomic) design features. A design feature is *primitive* or *atomic* if it cannot be further divided into other design features. For example, *object read* is a primitive design feature whereas *object access* is not because it consists of two primitive design features: *object read* and *object write*. For more details on the BBG modelling, please refer to [8].

An Example

We use a simple module-based Java program (Listing 1) to illustrate how the formal framework works. In Java, we equate a module to a package. The sample program consists of two Java packages: A and B, each with three Java classes. Package A consists of classes A1.java, A2.java, and A3.java. Package B consists of classes B1.java, B2.java, and B3.java.

```
package A;
import java.util.*;
import B.*;

public class A1 {
    private Vector APublicMethods ;
    private Vector AProtectedMethods ;
    private Vector APrivateMethods ;

    public A1 () {}

    public void MA11() { B1 obj_b1 = new B1 (); obj_b1.MB11(); }
    public void MA12 () {}
    public void MA13 () { B2 obj_b2 = new B2 (); obj_b2.MB21 (); }
    public Vector GetAPublicMethods() { return APublicMethods ; }
    public void SetAPublicMethods ( Vector vVal ) { APublicMethods = vVal ; }
    protected Vector GetAProtectedMethods () { return AProtectedMethods ; }
    protected void SetAProtectedMethods ( Vector vVal ) { AProtectedMethods =
```

```
    vVal ; }
    private Vector GetAPrivateMethods () { return APrivateMethods ; }
    private void SetAPrivateMethods ( Vector vVal ) { APrivateMethods = vVal ; }
} //end of class A1

package A;
import B.*;

public class A2 extends A1 {
    public A2 () {}

    public void MA21 () {}
    public void MA22 () {}
    public void MA23 () { B2 obj_b2 = new B2 (); obj_b2.MB21 (); }
    public void MA24 ( String para1 , Integer para2, Float para3){}
} //end of class A2

package A;
import B.*;

public class A3 extends A2 {
    public A3 () {}
    public void MA31 () {}
    public void MA32 () { B3 obj_b3 = new B3(); obj_b3.MB31 (); }
    public void MA33 () {}
    public void MA34 ( String para1 , Integer para2 , Float para3){}
} //end of class A3

package B;
import java.util.*;
import java.io.*;
import A.*;

public class B1 {

    private Vector BPublicMethods;
    private Vector BProtectedMethods;
    private Vector BPrivateMethods;

    public B1 () {}
    public void MB11 () { A1 obj_a1 = new A1 (); obj_a1.MA12 (); }
    public void MB12 () {}
    public void MB13 () { A3 obj_a3 = new A3 (); obj_a3.MA31 (); }
    public Vector GetBPublicMethods () { return BPublicMethods ; }
    public void SetBPublicMethods ( Vector vVal ) { BPublicMethods = vVal ; }
    protected Vector GetBProtectedMethods () { return BProtectedMethods ; }
    protected void SetBProtectedMethods ( Vector vVal ) { BProtectedMethods =
        vVal ; }
    private Vector GetBPrivateMethods () { return BPrivateMethods ; }
    private void SetBPrivateMethods ( Vector vVal ) { BPrivateMethods = vVal ; }
} // end of class B1

package B;
import A.*;
public class B2 extends B1 {
    public B2 () {}

    public void MB21 () {}
    public void MB22 () { A3 obj_a3 = new A3(); obj_a3.MA33(); }
    public void MB23 () {}
    public void MB24 ( String para1 , Integer para2, Double para3){}
} //end of class B2

package B;
import java.io.*;
import A.*;

public class B3 extends B2 {
    public B3 () {}

    public void MB31 () {}
    public void MB32 () {}
    public void MB33 () { A2 obj_a2 = new A2(); obj_a2.MA21(); }
    public void MB34 ( String para1 , Integer para2, Float para3){}
} //end of class B3
```

Listing 1: Java source code for classes in packages A and B

Table 1 shows the generated BBG relation sets that will be used as the base types to support the

Z formalization of the module-level metrics. For more details on the complete list of BBG relations sets, please refer to [8].

From	To	Relation semantics	BBG Relation Set pairs
Class	Class	Inheritance	{(A2,A1), (A3,A2),(B2,B1), (B3,B2)}
Class	Method	Private instance method definition	{(A1.GetAPrivateMethods), (A1.SetAPrivateMethods), (B1.GetBPrivateMethods), (B1.SetBPrivateMethods)}
Class	Method	Protected instance method definition	{(A1.GetAProtectedMethods), (A1.SetAProtectedMethods), (B1.GetBProtectedMethods), (B1.SetBProtectedMethods)}
Class	Method	Public instance method definition	{(A1.GetAPublicMethods), (A1.SetAPublicMethods), (A1,MA11), (A1,MA12), (A1,MA13), (A2,MA21), (A2,MA22), (A2,MA23), (A2,MA24), (A3,MA31), (A3,MA32), (A3,MA33), (A3,MA34), (B1,GetBPublicMethods), (B1,SetBPublicMethods), (B1,MB11), (B1,MB12), (B1,MB13), (B2,MB21), (B2,MB22), (B2,MB23), (B2,MB24), (B3,MB31), (B3,MB32), (B3,MB33), (B3,MB34)}
Class	Method	Private class method definition	{}
Class	Method	Protected class method definition	{}
Class	Method	Public class method definition	{}
Method	Object Reference	Formal parameters	{}
Class X Method	Class X Method	Method to method message	{((A1,MA11),(B1,MB11)), ((A1,MA13),(B2,MB21)), ((A2,MA23),(B2,MB21)), ((A3,MA32),(B3,MB31)), ((B1,MB11),(A1,MA12)), ((B1,MB13),(A3,MA31)), ((B2,MB22),(A3,MA33)), ((B3,MB33),(A2,MA21))}
Module	Class	Module-class association	{(A,A1), (A,A2), (A,A3), (B,B1), (B,B2), (B,B3)}

Table 1: Generated BBG relation sets

Once the BBG relation sets are created from the Java example to represent the design space, the design can be analyzed formally. Syntax-based software metrics (structural metrics) can be defined and extracted from these sets, as shown in the next section using the Z formal language.

Formalization of module metrics using Z and BBG

The design state space makes it possible to link a metric definition explicitly to the design feature (or features) in BBG. These design features are easy for programmers and analysts to understand and they underline the metric's measurement objective. Linking a metric definition to the design features in the formalized metrics eliminates potential misunderstanding and misinterpretations of the metrics.

We use the Module Interaction Index Metric (MII) metric as an example to show how to formalize module-level metrics. The MII metric calculates how frequently the methods listed in a module's APIs, both Service API (S-

APIs)¹ and Extension API (E-APIs)², are used by the other modules in the system. MII is defined as follows [11]:

$$MII(p) = \frac{\bigcup_{i \in I(p)} ExtCallRel(i)}{|ExtCallRel(p)|},$$

$$MII(S) = \frac{1}{p} \sum MII(p)$$

$I(p)$ denotes all of the APIs for a module p . For an API $i \in I(p)$, let $ExtCallRel(i)$ be the set of calls made to any of the methods of i from methods outside of the module p .

$$ExtCallRel(i) = \{ \langle m_f, m_t \rangle \mid Call(m_f, m_t) \wedge m_t \in M(i) \wedge m_f \notin M(p) \}$$

$ExtCallRel(p)$ denotes the set of all external calls made to all of the methods of all classes in module p .

$$ExtCallRel(p) = \{ \langle m_f, m_t \rangle \mid Call(m_f, m_t) \wedge m_t \in M(p) \wedge m_f \notin M(p) \}$$

In an ideal state, all of the external calls to module p should take place only through its officially designated API methods. The $MII(S)$ value ranges from 0 to 1. A max $MII(S)$ value of 1 indicates an ideal system where all intermodule interactions are only through the officially designated S-API methods. A min $MII(S)$ value of 0 is indicative of a system with very bad intermodule interaction.

A structural metric is denoted by a *characteristic set*. The cardinality of the characteristic set equals to the metric value. A characteristic set contains syntactical elements that share certain characteristics. For example, the set of red buses is a characteristic set because all the elements in the set are red buses. In other words, the elements in a characteristic set are not arbitrarily chosen. Table 2 introduces the basic Z notations that are used to formalize the module-level metric we used to validate our formalized framework.

¹ A Service API (S-API) declares the services that the module provides to the rest of the software system [12][11].

² Extension API: An Extension API (E-API) is a declaration of what functionality needs to be provided by an external plugin for the module.

Z function	Meaning	Z Notation	Example
Domain	If R is a relation of type $X \leftrightarrow Y$, then the domain of R is the set of elements in X related to something in Y (first element in each pair in the set)	dom	$drives = \{ (helen, beetle), (indra, alfa), (jim, beetle), (kate, cortina) \}$ $dom\ drives = \{ helen, indra, jim, kate \}$
Range	The range of R is the set of elements of Y to which some element of X is related	ran	$ran\ drives = \{ alfa, beetle, cortina \}$
domain restriction	If R is a relation of type $X \leftrightarrow Y$, and A is any subset of X, then $A \triangleleft R$ denotes the domain restriction of R to A	$A \triangleleft R$	$\{jim, kate\} \triangleleft drives = \{beetle, cortina\}$
range restriction	If R is a relation of type $X \leftrightarrow Y$, and B is any subset of Y, then $R \triangleright B$ denotes the range restriction of R to B	$R \triangleright B$	$drives \triangleright \{beetle, cortina\} = \{jim, kate\}$
function	If each object of one set is related to at most one object of another, then the relation between the two sets is said to be a function.	\rightarrow	$X = \{A, B\}, Y = \{C, D\}$ $X \rightarrow Y = \{(A, C), (B, D)\}$
relation	If X and Y are sets, then $X \leftrightarrow Y$ denotes the set of all relations between X and Y. Any element of $X \leftrightarrow Y$ is a set of ordered pairs in which the first element is drawn from X, and the second from Y	\leftrightarrow	$X = \{A, B\}, Y = \{C, D\}$ $X \leftrightarrow Y = \{(A, C), (B, C), (A, D), (B, D)\}$
power set	If a is a set, then the set of all subsets of a is called the power set of a, and written \mathbb{P} .	\mathbb{P}	if a is the set $\{x, y\}$ then $\mathbb{P}a = \{ \emptyset, \{x\}, \{y\}, \{x, y\} \}$

Table 2: Basic Z specification language notations

We define the characteristic sets of the MII metric using the Z notations as follows:

$[CLASS, METHOD]$

CLASS is the basic set of all classes, *METHOD* is the basic set of all methods in a program, and *MODULE* is the basic set of all modules in a program.

$ClassAPIMethod : CLASS \leftrightarrow \mathbb{P} METHOD$
$ClassAPIMethod = ClassPublicInstanceMethod \cup ClassPublicClassMethod$
MEA
$mea_ : MODULE \leftrightarrow \mathbb{P}(CLASS \leftrightarrow \mathbb{P} METHOD)$
$\forall modl : MODULE; c : CLASS; m, cm : (CLASS \leftrightarrow \mathbb{P} METHOD);$ $meaCM : \mathbb{P}(CLASS \leftrightarrow \mathbb{P} METHOD) \mid$ $c \in ran(\{modl\} \triangleleft ModuleClass) \bullet cm \in ClassAPIMethod \triangleleft c \bullet$ $meaCM \in mea\ modl \Leftrightarrow meaCM \in ran(MethodToMethodMessage \triangleleft cm)$

The *ModuleClass* is a basic relation set in BBG. Table 1 shows the extracted *ModuleClass* set for the Java example.

MHEA
$mhea_ : (CLASS \leftrightarrow METHOD) \leftrightarrow \mathbb{P}(CLASS \leftrightarrow \mathbb{P} METHOD)$
$\forall cm : (CLASS \leftrightarrow METHOD); mheaCM : \mathbb{P}(CLASS \leftrightarrow \mathbb{P} METHOD) \mid$ $mheaCM \in dom(MethodToMethodMessage \triangleleft cm)$

Z formalization

We will use the Java example (Listing 1) to illustrate the Z formalization of the MII metric based on the BBG relation sets.

- First, we get all modules in the design, using the BBG relation set *ModuleClass*, which is the set of all module-class relations defined in the design. We assign the extracted set to the variable *MoC*. The *MoC* set for the example returns: $\{(A, A1), (A, A2), (A, A3), (B, B1), (B, B2), (B, B3)\}$
- Then, we collect the API methods, defined as public in the module. Formally defined as *ClassAPIMethod* =

$(ClassPublicInstanceMethod \cup ClassPublicClassMethod)$. The two sets involved in the union are basic BBG relations. For our example, $ClassAPIMethod = \{(A1, GetAPublicMethods), (A1, SetAPublicMethods), (A1, MA11), (A1, MA12), (A1, MA13), (A2, MA21), (A2, MA22), (A2, MA23), (A2, MA24), (A3, MA31), (A3, MA32), (A3, MA33), (A3, MA34), (B1, GetbPublicMethods), (B1, SetBPublicMethods), (B1, MB11), (B1, MB12), (B1, MB13), (B2, MB21), (B2, MB22), (B2, MB23), (B2, MB24), (B3, MB31), (B3, MB32), (B3, MB33), (B3, MB34)\}$.

- Next, we collect the method-to-method messaging in each class. This is represented by the BBG *MethodToMethodMessage* relation set. In the example, $MethodToMethodMessage = \{((A1, MA11), (B1, MB11)), ((A1, MA13), (B2, MB21)), ((A2, MA23), (B2, MB21)), ((A3, MA32), (B3, MB31)), ((B1, MB11), (A1, MA12)), ((B1, MB13), (A3, MA31)), ((B2, MB22), (A3, MA33)), ((B3, MB33), (A2, MA21))\}$.
- To calculate the external calls into the module, we use these steps in the Z schema definition:

- " $ran(\{module\} \triangleleft MoC)$ " domain restricts the set of modules in the design by a specific module, then get the range set. For our example, applying this on module A would result in:
 - $\{A\} \triangleleft MoC = \{(A, A1), (A, A2), (A, A3)\}$.
 - Then, $ran(A \triangleleft MoC) = \{A1, A2, A3\}$.

The result assigned to variable MC.

- Get the methods in all classes, represented by the set *ClassAPIMethod* calculated before for our example.

- c) Domain restrict *ClassAPIMethod* by *MC* – in Z syntax this is formally written as $ClassAPIMethod \triangleleft MC$ - to get the class methods pairs for the module(s) that of interest to this example. $(ClassAPIMethod \triangleleft \{A1, A2, A3\}) = \{(A1, GetAPublicMethods), (A1, SetAPublicMethods), (A1, MA11), (A1, MA12), (A1, MA13), (A2, MA21), (A2, MA22), (A2, MA23), (A2, MA24), (A3, MA31), (A3, MA32), (A3, MA33), (A3, MA34)\}$. The result set is assigned to variable *MIC*.
- d) Next, we get all of the external calls by the methods in the set *MIC*. By domain restricting the BBG relation set *MethodToMethodMessage* by *MIC* calculated in (c) above. $(MethodToMethodMessage \triangleleft MIC) = \{(A1, MA11), (B1, MB11), ((A1, MA1), (B2, MB21)), ((A2, MA23), (B2, MB21)), ((A3, MA32), (B3, MB31))\}$. Result is assigned to variable *MethodsExtCalls*.
- e) Then, we get only the range from the set $ran(MethodsExtCalls) = \{(B1, MB11), (B2, MB21), (B3, MB31)\}$ with cardinality value 3.
- f) This represents $ExtCallRef(p)$ in the original MII definition, where p is the module A under measurement in the design.
5. To calculate the external calls to a single method in a module, we use these steps in another formal Z schema definition:
- a) The input will be the class-method pair, which we want to calculate the external calls for. For example, we want to know the external calls for (A1, MA11).
- b) We get all external calls to the class-method pair. By domain restricting *MethodToMethodMessage* by the class-method pair passed. $MethodToMethodMessage \triangleleft \{(A1, MA11)\} = \{(A1, MA11), (B1, MB11)\}$. The result is assigned to variable *MethodExtCall* (note: this is different from the variable *MethodsExtCall* used in 4.d before).
- c) Then, we get only the range from the set $ran(MethodExtCalls) = \{(B1, MB11)\}$ with cardinality value 1.
- d) This represents $ExtCallRef(i)$, where *i* is an API method in a module p to represent the external calls to this

method only, and not all of the methods in the module. For the previous steps, *i* represents the pair (A1, MA11).

The same steps 5.a to 5.d will be repeated for the remaining class-method pairs in module A. These are represent in this set $\{(A1, GetAPublicMethods), (A1, SetAPublicMethods), (A1, MA12), (A1, MA13), (A2, MA21), (A2, MA22), (A2, MA23), (A2, MA24), (A3, MA31), (A3, MA32), (A3, MA33), (A3, MA34)\}$. And the result would be $\{(B1, MB11), (B2, MB21), (B3, MB31)\}$ with cardinality 3.

Therefore, the set $\{(B1, MB11), (B2, MB21), (B3, MB31)\}$ represents the characteristics set for the metric Module External Access (MEA), which in our example was applied to module A. The result represents the frequency that methods in a particular module are being accessed by methods in other modules.

And the set $\{(B1, MB11), (B2, MB21), (B3, MB31)\}$ represents the characteristics set for the metric Method External Access (MHEA), when applied to a particular method in a module, to represent the frequency that this method is being accessed by methods in other modules. This completes the Z formalization of the MII metric.

The Design State Space Toolkit

We implemented a new toolkit called the Design State Space (DSS) toolkit to validate that the proposed framework is practical and useful.

The DSS Parser utilizes the *JavaCC* technology [9] and the *SableCC* compiler compiler [6] to support the parsing and extraction of the design state space. The DSS Analyser serves two main purposes. First, it provides the mapping of the parsed tree sets from the source code into the BBG relation sets as the intermediate representation. Second, it extracts definitions, such as structural metrics, from the BBG sets. These relations sets provide convenience and flexibility for all kinds of analyses and future plug-in modules to the tool. For example, we can use these sets to extract structural metrics, if the structural metrics are formalized in the framework. If a new metric definition is provided through the plug-in interface, the tool can extract that metric without changing anything else in the software design.

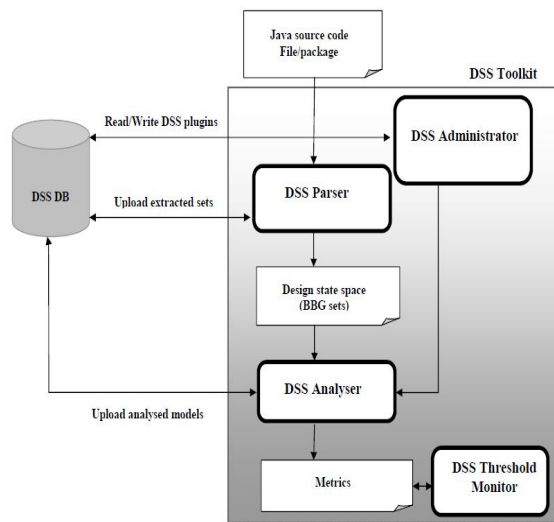


Figure 1 : The Design State Space architecture

Figure 1 shows a high-level architecture of the DSS toolkit. For a more detailed discussion of the DSS toolkit and the different modules, please refer to [1].

Conclusions and future research

In this paper, we presented the Z formalization of an object-oriented module-level metric. This formal framework links metrics definitions to the design structures in a graph-based design representation (BBG) that is formalized by the Z specification language. Using the proposed approach, we formalized the module metric: Module Interaction Index (MII). In the future, we plan to formalize more module-level metrics of different complexity to further validate the proposed framework.

References

- [1] R. Alomainy, W Li, and M. Currie., "DSS: A Software Tool to Extract Object-Oriented Design State Space," 24th International Conference on Computers and Their Applications in Industry and Engineering (CAINE-2011), November 2011.
- [2] M.H. Ammann, R.D. Cameron, "Measuring program structure with inter-module metrics," Eighteenth Annual International on Computer Software and Applications Conference, vol., no., pp.139-144, November 1994.
- [3] E. Arisholm, L.C. Briand, A. Foyen, "Dynamic coupling measurement for object-oriented software," IEEE Transactions on SE, vol.30, no.8, pp. 491- 506, August 2004.
- [4] S.R. Chidamber and C.F. Kemerer, "A Metrics Suite for Object Oriented Design," IEEE Transactions on Software Engineering, vol. 20, no.186, pp. 476-493, 1994.
- [5] W. Frakes, K Kang, "Software Reuse Research: Status and Future", IEEE Transactions on Software Engineering, pp. 529-536, July 2005.
- [6] É. Gagnon, "SABLECC: An Object-Oriented Compiler Framework." MSC thesis School of Computer Science McGill University, Montreal March 1998.
- [7] W. Li and R. Alomainy, "An Experimental Approach to Prove Metrics Properties," Software Engineering Research and Practice, pp. 408-413, 2009.
- [8] W. Li, "The Big Bang Graph – A colored-graph representation of software design," Computer Science Research Trends. Editor: Casey B. Yarnall, pp. 377-388, June 2008.
- [9] Oracle Corporation, "Java Compiler Compiler (JavaCC)," Version 5.0, <http://java.net/projects/javacc>.
- [10] S. Sarkar, S.; Rama, G.M.; Kak, A.C.; "API-Based and Information-Theoretic Metrics for Measuring the Quality of Software Modularization," IEEE Transactions on Software Engineering, vol.33, no.1, pp.14-32, January 2007.
- [11] S. Sarkar, A.C. Kak, G.M. Rama, "Metrics for Measuring the Quality of Modularization of Large-Scale Object-Oriented Software," IEEE Transactions on SE, , vol.34, no.5, pp.700-720, October 2008.
- [12] M. Spivey, "The Z Notation: A Reference Manual," Prentice Hall International Computer Press: Upper Saddle River, NJ, 1992.
- [13] W.E. Wong, J.R. Horgan, M. Syring, W. Zage, D. Zage, "Applying design metrics to a large-scale software system," The 9th International Symposium Proceedings on SE Reliability, pp.273-282, November 1998.

eSQUARE: A Formal Methods Enhanced SQUARE Tool

Hadil Abukwaik¹, Cui Zhang

Computer Science Department, California State University Sacramento, CA 95819
hadeel.abukwaik@informatik.uni-kl.de, zhangc@ecs.csus.edu

Abstract - *This paper presents a formal methods enhanced tool named eSQUARE that supports the SQUARE security requirements engineering methodology developed by CMU SEI [7]. Development of security requirements has been neglected for a long time in the software development industry [9]. This has caused many errors and failures in the delivered software products and increased the cost on defect correction and product maintenance. Therefore it is important to have precise and concise specification of security requirements and have early integration of security requirements in software development process. The tool presented is aiming at helping practitioners to integrate the specification of security requirements using formal language Z [3, 6, 11] with the security requirement engineering methodology SQUARE.*

Keywords: Security Requirements, Requirements Engineering, Formal Methods

1 Introduction

There is no doubt that software products are affecting almost every aspect of our daily lives. Software is found in electronic medical systems, online calls, bank ATMs, controllers of spaceships and rockets, etc. In such products, security requirements, including safety, gets a special importance as failures in satisfying the requirements may lead to a serious harm that can affect lives and money. However, if we look at the implemented security in software projects, we find only general mechanisms describing password protection, firewalls, virus detection tools, etc. [8]. This indicates the improper elicitation and inadequate development of the system-specific security requirements.

Requirements engineering plays a vital role in developing quality in software applications and in reducing the cost for correcting the defects in the released products. Generally, in order to meet software requirements successfully, all requirements need to be specified at the very beginning of the software development process. Unfortunately, this is not the case for security requirements. According to Mead et al. [7], "Studies show that upfront attention to security saves the economy billions of dollars.

Industry is thus in need of a model to examine security and quality requirements in the development stages of the production lifecycle." Therefore, a number of methods and techniques have been developed to ensure meeting the security requirements. One of these methods is the Security Quality Requirements Engineering (SQUARE) [7] which aims at integrating the security requirements engineering with the requirements engineering phase.

In addition, the more complex the software systems become the more precise and complete requirements specification is needed. As pointed out by Clarck et al. [2], "One way of achieving this goal is by using formal methods, which are mathematics-based languages, techniques, and tools for specifying and verifying such systems." Using formal methods will support the consistence and the correctness of software systems. In other words, it solves the ambiguity problem that can be faced when informal natural languages are used by many requirements engineers in eliciting and documenting requirements from the users and stakeholders. Not only this, but formally specified requirements are also absolutely better testable than informal ones. Although, using formal methods will add a reasonable advantage in expressing and testing the requirements it is not a guarantee of correctness [9, 12].

This paper presents a web-based tool named eSQUARE that fully supports the nine steps of SQUARE and uniquely enhanced using the formal methods based language Z [3, 6, 11] for modeling and checking elicited security requirements. The rest of the paper is structured as follows. Section 2 is a review on SQUARE methodology, formal methods based languages, and the related work. Section 3 illustrates the tool eSQUARE. Section 4 concludes the paper and presents future work.

2 Background and related work

Security Quality Requirements Engineering (SQUARE) is a methodology developed at Software Engineering Institute (SEI) of Carnegie Mellon University (CMU) [7]. This methodology helps in engaging security requirements in the early stages of the software development process

¹ Currently in Computer Science Department, University of Kaiserslautern, Germany 67663.

where requirements engineering activities take place. It has been proved to be useful for documenting and analyzing the security aspects of already developed systems and can direct the enhancements and the changes that can be applied to these systems in the future [9]. SQUARE process is formed by nine discrete steps [7]: (1) agree on definitions, (2) identify security goals, (3) develop artifacts, (4) perform risk assessment, (5) select elicitation techniques, (6) elicit security requirements, (7) categorize requirements, (8) prioritize requirements and (9) requirements inspection.

There are many formal methods based languages that can be used to specify the requirements of software systems. Representative formal languages include Z, VDM, Larch, and LOTOS [12]. As pointed out by Mead et al. [9], "Some useful techniques are formal specification approaches to security requirements, such as REVEAL and Software Cost Reduction (SCR), and the higher levels of the Common Criteria." Automated tools that combine both a security requirements engineering methodology and a formal methods based language will be of great help in the development of secure software systems. We believe that developing a tool that can support the usage of a formal methods based language in the SQUARE methodology can provide a high confidence in the security of the developed software. This is the motivation of the development of the eSQUARE tool.

Although there are many methodologies and tools that support the security requirements engineering, the interest of this paper is particularly on supporting the SQUARE methodology and enhancing it with the formal methods. There are two existing tools serving the same focus.

mySQUARE was the first tool developed to support some steps of the SQUARE methodology to ease the management and administration of the process [14]. It enables tracking the progress of the work for each of the nine steps of the methodology, generating reports for users with the least time and effort, and using the XML technology to provide users with the portability of their files from one place to another. In spite of the great benefits mySQUARE offers, it has the following limitations: it is a stand-alone application that cannot be accessed from anywhere; it does not provide a built-in support for common techniques related to requirements elicitation, categorization and prioritization; and it does not provide import/export features for the project documents which many users may need [14].

In 2008, in conjunction with CyLab, SEI CMU developed and released the SQUARE prototype tool, workshop, tutorial, and educational materials which are useful in understanding the methodology [7, 10]. A number of papers present a light version of SQUARE called SQUARE-Lite and case studies about projects used it [4].

SQUARE-Lite is a five-step process taken from SQUARE. The technical approach of introducing SQUARE as part of the standard software life-cycle models is described in [10].

The team in CyLab built the SQUARE tool that supports the nine steps of SQUARE with a managerial interest. It takes care of managing the contributions among multiple requirement engineers and stakeholders. The SQUARE tool can be used in large companies working on large projects with a team for requirements engineering, however, it does not provide any formal methods based support for the specification of security requirements. Until the research stage for eSQUARE in early 2012, there are no official documentations or published papers for this type of tools.

3 The eSQUARE tool

3.1 The eSQUARE functionality

One of the vital factors of successful software projects is the effective cooperation between the stakeholders and the developers in general and the requirements engineers in particular. However, these two groups can be based in miles apart or even in different continents. In addition, the larger the project requirements the harder the management becomes and the more effort it needs to take. Many reports and documents are produced in following the SQUARE methodology which means more time, space and management are needed. In response, eSQUARE has been developed to support the nine steps of the SQUARE methodology to overcome these challenges with a unique enhancement that introduces the usage of the formal language Z [3, 6, 11] for the specification of elicited security requirements. Starting a project in eSQUARE, users benefit from interfaces that enable them to:

- Go through the nine steps of the SQUARE with the preferred order along with the online help which can be accessed at any time.
- Add, edit and delete the following units: project terms, business goals, security goals, risks, elicitation techniques, requirements categories and inspection techniques. These units are the base of the SQUARE methodology. As the units can be repeated in different projects, eSQUARE enables reusing them by users who created them to save their time and effort.
- Set up the project scale components' value, each risk likelihood, consequences, source and impact. These entries will be used by eSQUARE to calculate the risk value automatically and present it to users.
- Upload project security artifacts like security use cases and Z files and enable users to download or preview them as needed.
- Select the preferred formal methods based language from the four options offered: the standard Z, the Object Z, Circus or Z Rules.

- Select the preferred markup from the four options offered: Latex, UTF8, UTF16 and XML.
- Parse, check and export Z files that formally specify their security requirements. By this feature, eSQUARE offer a high confidence in the consistency and correctness of secure software systems.
- Un-check the type checking option if they do not want it.
- Get the structure of the Z sections that is written in the parsed Z files.
- Rank requirements based on categories and risk assessment results that have been computed by eSQUARE.
- Produce a summary for users' projects.

In addition to all the mentioned functionalities, eSQUARE improves users' experience in practicing the SQUARE methodology as they do not need to be professional requirements engineers to follow the nine steps. Users have the online help at each step where they can learn how to make progress in the process. This can reduce the project cost dedicated to security requirements engineering experts.

eSQUARE is a web-based tool designed to be flexible and user friendly so that users are capable of performing the steps in the order they prefer without restrictions. Another major advantage of the eSQUARE is the ability to upload the project's artifacts in a central database. In this way, users are able to view their uploaded artifacts from any computer connected to the internet.

3.2 The eSQUARE architecture

The web-based eSQUARE tool is a thin-client program that receives users' requests, performs all the processing on the server side and sends results to users as HTML. As shown in Figure 1, the multi-tier architecture pattern is chosen for eSQUARE as it provides good quality attributes to the system that will be described in the rest of the section. As seen in Figure 1 there are three basic components for the system:

- **The client machine:** This is where the web browser is hosted to enable users to start using the web-based tool.
- **The application server:** This server hosts the entire application files including HTML files, JSP files, Java files and the Community Z Tools (CZT) components' files [3, 6].
- **The database server:** This can be the same application server or any other server where the MySQL database of the application is hosted.

One of the advantages in a multi-tiered application is that users are able to work on the application data without knowing where the data is stored at the build time. Another

advantage is the modularity of the application's components with loose coupling and high cohesion characteristics. In other words, this architecture provides better modifiability and extensibility for the application in the future. Also, the code is easier to read, understand, and re-use. In addition, this architecture has advantages in the performance of the system. That is, hosting components of the layers on different machines decreases the work load and increase the speed of response compared to applications having all the components hosted on one machine dealing with requests for all components. This is noticed when the application usage grows up and its traffic increases. Furthermore, the multi-tier architecture results in a robust application. From one point, whenever a change needs to be made in a tier it does not affect the other tiers and is independent. This again helps in re-using the components of the software application.

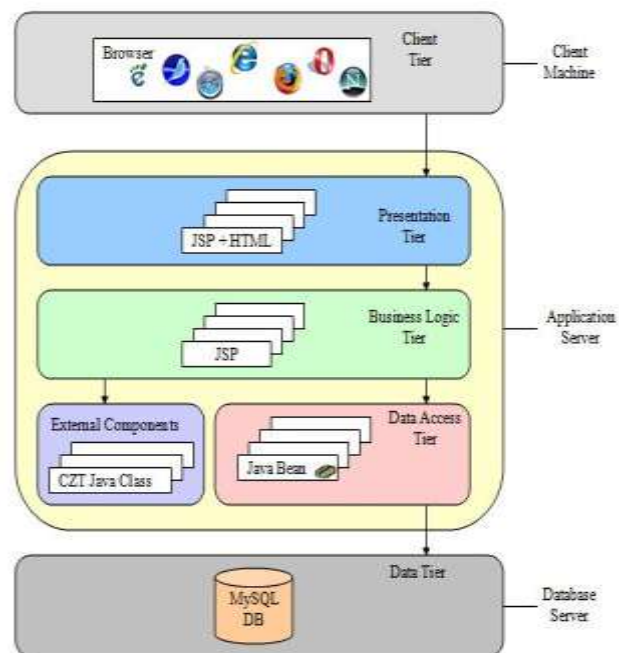


Figure 1. eSQUARE software architecture

3.3 The eSQUARE implementation

eSQUARE is implemented using the iterative life cycle model. In each iteration, a part of the requirements was selected based on its importance to the project, then implemented and tested. This approach has helped in monitoring the progress and planning in a simple and reliable way. eSQUARE is built using JSP (Java Server Pages) technology as it is portable (can be hosted in any operating system), easy to write and modular.

Like all technologies, JSP has its disadvantages too. The one identified during the implementation of eSQUARE is that there is a noticeable delay when we access the JSP page

for the first time. In fact, JSP files are compiled on the server when they are accessed for the first time. This compilation causes a delay.

It's important to mention that the open source CZT package [3] consists of many classes that build the Z tool which users can launch in the security requirement elicitation step provided by the eSQUARE tool. Despite of the number of tools available for the Z specification language, most of them do not support the ISO standard for this language [6]. Andrew Martin found that the many tools developed by school and academic were not complete and were not robust enough [6]. Based on that in 2001, he proposed the idea of the Community Z Tools to be a useful open source that supports the ISO specification of Z language. Integrating part of the CZT in the eSQUARE tool is of great advantage that allows the usage of existing tools to enhance the SQUARE methodology.

3.4 Example

In this section, the eSQUARE web-based tool is used to examine its efficiency in supporting the SQUARE methodology and in enhancing it with the formal methods based language Z in specifying the security requirements. All the data used in this example are taken from an in-depth case study performed by graduate students in Carnegie Mellon University under the supervision of Nancy Mead where the SQUARE methodology was applied on a product called Asset Management System (ASM) [1, 5]. Data are entered into eSQUARE to show how the tool can be used for managing the nine step process and how to use the integrated Z tool for writing the security requirements specification.

First of all a new project "ASM" is created to be used in our example. Terms that have been used in the ASM project were entered into eSQUARE under the first step of the process. These terms are available for users to use in other projects as they are associated to their accounts not to the project. In the second step, the business goal of ASM and its related security goals are entered to be in the system database and displayed to users. Many artifacts like use cases, misuse cases, architectural diagrams and figures are uploaded to eSQUARE each in a file that the user can download and review from any computer with internet access and a browser. With the designed risk assessment methodology in eSQUARE, only the risk impact level is displayed near each risk of the ASM project as it does not provide weights for the required factors of the risk assessment method. The agreed technique for eliciting the security requirements in ASM was the Interactive approach. With this approach, names and descriptions are entered into the system and can be viewed and edited as users need.

In the second phase of the SQUARE methodology case

study, the security requirements were refined and summarized in nine security requirements [5]. In eSQUARE these requirements are entered and saved successfully in the system. To examine the CZT component integrated in eSQUARE, a file of the Z specification in latex format is written for each of the nine security requirements of ASM. Figure 2 shows the Z specification for one security requirement of ASM called R_07, where its English description is [5]: It is a requirement that both process-centric and logical means be in place to prevent the installation of any software or device without prior authorization.



```

\begin{zed}

{Installer, Installation, ProcessCentricAuthorization,
LogicalAuthorization}

\end{zed}

\begin{zed}
Response : := Not_Authorized | Installed_Successfully
\end{zed}

\begin {schema} {ASM}
installers: \power Installer\
installations: \power Installation\
pcas: \power ProcessCentricAuthorization\
las: \power LogicalAuthorization\
install: Installation \pfun Installer\
hasPca: ProcessCentricAuthorization \fun Installer\
hasLa: LogicalAuthorization \fun Installer\
\where
installations = \dom install\
\ran install = installers\
\end{schema}

\begin {schema}{AuthorizedInstallation}
\Delta ASM\
i1? : Installer\
i2? : Installation\
a1? : ProcessCentricAuthorization\
a2? : LogicalAuthorization\
r! : Response\
\where
i2? \notin installations\
i1? = hasPca(a1?)\
i1? = hasLa(a2?)\
installations' = installations \cup \{i2?\}\
installers' = installers \cup \{i1?\}\
install' = install \cup \{i2? \mapsto i1?\}\
r! = Installed_Successfully\
\end{schema}

```

Figure 2. Z Specification of ASM security requirement R_07

After launching the CZT component, the written Z specification file is opened for parsing and type checking. In this example Standard Z language and Latex markup are selected and the files are both parsed and checked as seen in Figure 3.

After determining the option the component starts reading the file and gives feedback on the correctness of the specification. The Z file is examined twice; the first with its original correct specification and the next with an intended syntactical error. eSQUARE has passed this examination and responds correctly. Figure 4 shows the positive feedback given by eSQUARE for the correct version of the Z file along with its structure. As the file has been parsed and checked successfully, users are able to export it to different formats. A test is made and an export is performed on asm.tex to the UTF16 format and the result is a perfect Z file in the specified format. After examining the formal methods based features of eSQUARE, we go back to step seven of the SQUARE methodology case study. There were eight defined categories used in categorizing the security requirements of ASM. These categories have been defined in eSQUARE and are ready to be used in categorizing the nine security requirements of ASM.

eSQUARE uses integer numbers to express the priority of projects' security requirements. However in the ASM case study the ninth security requirements were prioritized under three levels (Essential, Conditional and Optional) [5]. As a result, each of the three level is assigned a number and is used in eSQUARE (Essential: 1, Conditional: 2 and optional: 3).

Finally, inspection technique selected for the ASM project was the Peer Review technique. This technique has been inserted and selected for the ASM project efficiently. A Project Summary Form is generated once by clicking the last link in the Nine Steps Form.

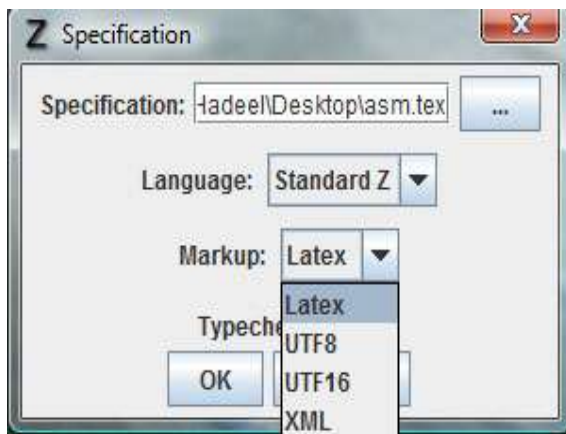


Figure 3. Selecting a language and a markup

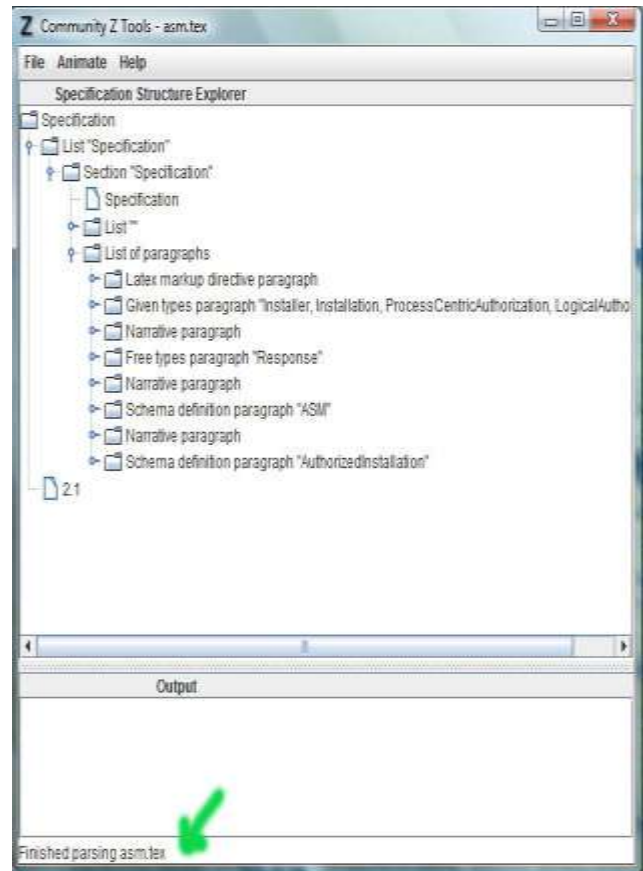


Figure 4. Feedback on Correct Z Specification

4 Conclusion

The eSQUARE tool is developed with a strong belief that tool support should be sufficiently provided for efficient security requirements engineering methodologies like SQUARE. eSQUARE has a promising future for its unique integration of the formal methods based language Z with the SQUARE process. It can grow to be a robust and a powerful tool that security requirement engineers can rely on to increase their confidence in the security requirements specifications. As shown by the example, eSQUARE is simple and easy to use where users are capable of practicing the nine steps of the SQUARE methodology in an effective way.

Table 1 presents a summary of similarities and differences among eSQUARE, mySQAURE and CyLab SQUARE tools. The three tools share multiple features like supporting the administration of SQUARE projects, providing users with help option at any time of the work flow and generating automatic summaries for projects. Though, eSQUARE and CyLab SQUARE are web-based and support the risk assessment step, but mySQAURE does

not. A noticeable difference is that eSQUARE provides the support for the formal methods based language Z while mySQUARE and CyLab SQUARE do not.

Table 1. SQAURE Tools Comparison

Tool \ Feature	eSQUARE	mySQUARE	CyLab SQUARE
Web Access	✓	×	✓
SQUARE Administrative Support	✓	✓	✓
Risk Assessment Support	✓	×	✓
Z language Support	✓	×	×
Automatic Project Summary Generation	✓	✓	×
Automatic Reports Generation	×	×	✓
Tool Accessible Documentation	✓	✓	×
User Help	✓	✓	✓
Team Work Support	×	×	✓

Although the presented example in the previous section is built upon real data taken from a real project called ASM [1, 5], it doesn't provide the empirical evaluation of the tool. The usage of the ASM data helped in getting the feeling on how to use eSQUARE forms and in its sequence, but there is still a need for an experiment or case study where eSQUARE get used along the security requirements engineering activities. Also, metrics for eSQUARE advantages should be defined and used in evaluating the tool.

5 References

- [1] P. Chen, M. Dean, D. Ojoko-Adams, H. Osman, L. Lopez, N. Xie, N. R. Mead. "System Quality Requirements Engineering (SQUARE) Methodology: Case Study on Asset Management System;" Software Engineering Institute, Special Report CMU/SEI-2004-SR-015 (December 2004). [Online]. Available: <http://www.sei.cmu.edu/library/abstracts/reports/04sr015.cfm>
- [2] E. Clarck, J. Wing, E. Al. Formal Methods: State of the Art and Future Directions. ACM Computing Surveys, vol.28, pp. 626 - 643 . (1996, December). [Online]. Available: <http://portal.acm.org/citation.cfm?id=242257>
- [3] Community Z Tools [Online] Available: <http://czt.sourceforge.net/manual.html>
- [4] A. Gayash, V. Viswanathan, D. Padmanabhan, N. R. Mead. "SQUARE-Lite: Case Study on VADSoft Project;" Software Engineering Institute, Technical Report CMU/SEI-2008-SR-017 (June 2008) [Online]. Available: <http://www.sei.cmu.edu/library/abstracts/reports/08sr017.cfm>
- [5] D. Gordon, T. Stehney, N. Wattas, E. Yu, N. Mead. "System Quality Requirements Engineering (SQUARE): Case Study on Asset Management System, Phase II;" Software Engineering Institute, Special Report CMU/SEI-2005-SR-005 (May 2005). [Online]. Available: <http://www.sei.cmu.edu/library/abstracts/reports/05sr005.cfm>
- [6] P. Malik, M. Utting. "CZT: A Framework for Z Tools;" Proceedings of 4th International Conference of B and Z Users, pp. 65-84. (2005, April). [Online]. Available: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.74.7673&rep=rep1&type=pdf>
- [7] N. R. Mead, E. D. Hough, Theodore R. Stehney. "Security Quality Requirements Engineering (SQUARE) Methodology;" Software Engineering Institute, Technical Report CMU/SEI-2005-TR-009 (November 2005). [Online]. Available: <http://www.sei.cmu.edu/reports/05tr009.pdf>
- [8] N. R. Mead. "How To Compare the Security Quality Requirements Engineering (SQUARE) Method with Other Methods;" Software Engineering Institute, Technical Report CMU/SEI-2007-TN-021 (August 2007). [Online]. Available: <http://www.sei.cmu.edu/reports/08tn006.pdf>
- [9] N. R. Mead, E.D Hough. "Security Requirements Engineering for Software Systems: Case Studies in Support of Software Engineering Education;" Presented at 19th Conference on Software Engineering Education & Training, pp.149-158. (April 2006). [Online]. Available: http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?tp=&arnumber=1617341&isnumber=33901
- [10] N. R. Mead, V. Viswanathan, D. Padmanabhan, A. Raveendran. "Incorporating Security Quality Requirements Engineering (SQUARE) into Standard Life-Cycle Models;" Software Engineering Institute, Technical Note CMU/SEI-2008-TN-006 (May 2008). [Online]. Available:

- <http://www.sei.cmu.edu/library/abstracts/reports/08tn006.cfm>
- [11] J. M. Spivey. "The Z Notation: A Reference Manual;" Prentice Hall, 1992. [Online]. Available: <http://spivey.oriel.ox.ac.uk/~mike/zrm/>
- [12] G. Vinu, R. Vaughn. "Application of Lightweight Formal Methods in Requirement Engineering1;" CrossTalk Journal (January 2003). [Online]. Available: <http://www.stsc.hill.af.mil/CrossTalk/2003/01/George.html>
- [13] G. H. Walton, T. A. Longstaff, R. C. Linger. "Computational Evaluation of Software Security Attributes;" Software Engineering Institute, Technical Report CMU/SEI-2006-TR-021 (December 2006). [Online]. Available: <http://www.sei.cmu.edu/reports/06tr021.pdf>
- [14] G. Yip, C. Zhang. "SQUARE AND mySQAURE: SECURITY MADE EASY;" Proceedings of the International Conference on Software Engineering and Applications. (November 2009).

Formal Modeling and Analysis of Autonomous Robotics System Using Z

Yujian Fu¹, Nithin Yama¹, and Zhijiang Dong²

¹Department of Computer Science, Alabama A&M University, Normal, AL, USA

²Department of Computer Science, Middle Tennessee State University, Murfreesboro, TN, USA

Abstract—*In recent years, the mobile robotics systems have been developed and attracted more and more attentions because of its applications in various disciplines. The software development of mobile robotics is a key and complex process and it is required to be reliable and correct when placed in the unknown working environment. To illustrate how to design reliable and correct robotics systems, software design is the key concern in robotics systems. Based on the set theory and first order logic, Z notation is a formal approach widely used for the modeling of distributed and embedded system design. In this research, we presented a formal framework based on the Z notation to describe generic behaviors of robotics systems. A case study of field guard robot that uses the LEGO NXT toolkit was implemented in Java to validate the framework. In the mobile robot navigation system, a robot travels from a start state to a specified final state. To navigate the position of robot and objects, coordinate systems are used to specify and analyze the paths from start state to the final state. Using Z based framework, it is able to investigate and analyze the entire formal specification of the autonomous mobile system.*

Keywords: Mobile robotics systems, software design, formal framework, Z notation

1. Introduction

Many applications of autonomous mobile robots are developed and widely used in various disciplines, such as, manufacturing, construction, waste management, undersea task, space exploration, medical surgery, serving and assistance for the disabled people. Autonomy and navigation are two major concerns in a broad domain covering a large spectrum of applications and technologies. They have drawn more and more researchers attentions due to multiple disciplines as well as the advanced techniques involved. Therefore, how to design a reliable and correct mobile robot system is a challenge issue now. Making progress toward autonomous robots is of major practical interest in a wide variety of applications.

It is a fundamental requirement for any autonomous robot that is able to navigate from one location to another. Avoiding the dangerous situation on the path such as collision with obstacles, falling in a cave to stay in the safe operational environment is the first class requirement during navigation.

Navigation is an ability to identify robot's current position, calculate the new path facing the obstacles while remember and travel towards the destination. To build reliable and robust autonomous robotics systems and achieve the quality of navigation, different approaches have been proposed. In this research, we use Z notation to model mobile robot including navigation. The problem scenario we examine is an autonomous robot that travel through an area that limited by color line. The robot needs to find all obstacles, collect them and removes to certain area and goes back to the place that obstacle was found. The mobile robot needs to be able to navigate within the area, identify current, collect locations and calculate the path between.

Formal methods are mathematical based notations that can precisely describe the system in an abstract level by reducing ambiguity and inconsistency. The Z notation is based on the set theory and first order logic with denotational semantics. However, due to the complexity of the symbols and knowledge, formal methods are not widely accepted in industry. The basic unit in Z notation is a schema, each data and operation can be specified by a schema or set of operations of schemas. This research work modularizes the Z notation into several blocks of robotics systems and expresses in the set of schemas. Therefore, it will facilitate further research in the automatic validation and verification tool.

The remainder of this paper is organized as follows. Section 2 gives a brief description of Z notations. In Section 3, we discuss the related works of formal modeling of robotics systems. After that, a Z model of robotics systems is presented in Section 4. A case study using LEGO NXT toolkit is presented in Section 5. Conclusion and future investigations are discussed in the Section 6.

2. Overview of Z Notations

Z notation is a formal specification notation that was first created by J. R. Abrial and further developed by the Programming Research Group at Oxford. It is based on the well-known mathematical concepts of set theory and predicate logic.

The primary construct in Z notation is called schema, which represents a block, an operation, a set of data or a subsystem. A schema is defined by two parts – data

declaration and system constraints. The data declaration consists of the definition of necessary data of the schema with data types in the form of $v : T$, where v is the variable and T represents the type of data (data set). The constraints of the system can be specified by a set of predicates usually denoted by first order logic. In the operation schema, the constraints of the system includes the precondition and postcondition that define the relationships between the declared variables. A schema has following form:

<i>Schema</i>
<i>//datadeclaration</i>
<i>//preconditions</i>
<i>//postconditions</i>

Data types in Z notation can be defined as basic or composite. The basic data types are usually denoted by capitalized letters. Composite data types can be the cartesian product and schema types. Any data defined must have a data type.

The operations are functions that maps a data set to another. The mapping function can be total or partial relation.

Any schemas in Z notation can be either state schemas or operation schemas. State schemas capture the static aspect of a system by defining the database, constraints and initial data. Operation schemas capture the dynamic aspects and define input-output relations. In other words, operation schema defines an operation in terms of the relationship between the state before the operation executes and the state after it has completed execution. The declaration part contains variables representing the before and after states, input and outputs. The constraints specify the precondition that can cause state change and postcondition after state change. For each system model, it is important to identify the key data and operations. Following notations are conventionally used in the content:

- Unprimed variable (e.g. v) – value of a variable before execution of an operation.
- Primed variable (e.g. v') – value of a variable after execution of an operation.
- Variable suffix with ? – an input variable to an operation.
- Variable suffix with ! – an output variable to an operation.
- ΔS – denotes change for the data in set (or database of) S .
- $\exists S$ – denotes no change for the data in set (or database of) S .

3. Related Work

There are a lot of works have been done on the modeling and analysis of embedded systems using Z notation, but only very few works that has been done in the modeling robotics systems using Z notation in the literature. In this section, we

will overview the works that applied formal methods to the robotics systems.

A model to guide and keep track of a robot such that it is able to complete several tasks is described in [9]. Petri nets and Petri net extension methodologies have been used to model systems for the control and coordination in the unstructured environment. Murata et al. [7] presented an algorithm to construct predicate/transition models of robotic operations. Basically, robot actions were described as enabled and firing transitions and the model was used for the planning of concurrent activities of multiple robots. The model shows the ability of the Petri net models to capture interactions between the agents that are not evident in the design process. In a similar way, Xu et al. [12] proposed a methodology based on predicate/transition nets for multiple agents under static planning of activities. In addition, they proposed a validation algorithm for plans with parallel activities. The work of Leitao et al. [5], proposed a Petri net model approach to formal specification of holonic control systems for manufacturing. They developed a Petri net submodel for each of the four types of holons (agents) suggested in the ADACOR (Adaptive Holonic Control Architecture for Distributed Manufacturing Systems) architecture. There was no attempt to study the structural properties of the Petri net model in order to assess some sort of dependability in the proposed architecture. Now the existing navigation of single robot has extended to multiple robots system. The multiple robots addressing the issues of cooperation and formation control is discussed in several works ([3], [2]) where reactive behavior based approach to formation control is described. Some other works use Petri net plan can be found in [6-13].

Finite automata and graph theory [4] have proved to be useful mathematical models for robot navigation through a discrete environment. In [8], finite automata are used to control the navigation of mobile robot along the possible paths. They analyzed the movement and controlled the population of a mobile robot by applying the algorithms of graph theory. A number of modeling techniques for mobile robot have been developed by researchers such as partially observable Markov and behavior based navigation but the existing approaches have lack of the formalization. Melo, Isabel and Lima in [6] has examined the problem of multi-robot navigation. They have analyzed the problem of driving a robot population moving in a discrete environment from some initial to a target configuration. In this paper, we build a modeling framework using Z notation to formal describe autonomous mobile robot systems.

4. A Modeling Framework of Robotics Using Z Notation

Formal specification language (FSL) is a mathematical based notation on top of algebraic, logics and/or discrete mathematics. Therefore, FSL can provide a precise analysis

and consistent reasoning on the system properties from design model. However, to establish a formal model is not a straightforward job because it is challenging and tricky to connect the mathematical notations to the dynamic and flexible real world systems.

One of the commonly used formal specification languages is Zed language or Z notation, which is based on the set theory and predicate logic. The Z notation specifies the system by input-output relation and describes the operations by the constraints on the states. Two typical components in the Z notation are state and operations that represent the data and events of the systems. The fundamental block in Z notation is schema that is composed of states and operations. In this case, Z notation provides a precise and simple style for the system model. The fundamental unit of the Z notation is a schema, which includes two parts – data and constraints. There are two types of schemas – state schema and operation schemas. In the state schema, the data of the system and initial conditions of the data are specified. In the operational schemas, the data that is affected and operated on, the pre-conditions and post conditions are described as set notation, predefined syntax and first order logic. For specific syntax of the Z notation, readers can refer to Spivey's book [10].

The modeling framework of autonomous robot includes three major components – environment, controller, sensor. The environment schema aims at description of the scope. Two types of approaches can be used – coordinate systems or directed graph (DG). In this work, we present a model based on the coordinate system. Assume a two dimensional system, a coordinate system is described by a triple $Co \triangleq \langle O, X, Y \rangle$, where O is the origin, and X and Y are horizontal and vertical numbers respectively.

$$\begin{array}{l} \textit{Environment} \\ \textit{CS} : \textit{ORIGIN} \times \textit{HX} \times \textit{VY} \times \textit{RC} \times \textit{OC} \\ \forall c \in \textit{CS}.c[1] == \langle 0, 0 \rangle \wedge c[4] \neq \lambda \end{array}$$

In the above schema of *Environment*, any data is specified by five fields – origin, horizontal number, vertical number, robot position and obstacle position. It requires that the origin starts at $\langle 0,0 \rangle$, and robot is within the scope and should be known $c[4] \neq \lambda$. The coordinate system provides the direction for the robot movement. Similarly, we can define the schema of Robot as follows:

$$\begin{array}{l} \textit{Robot} \\ \textit{Robot} : \textit{ORIGIN} \times \textit{HX} \times \textit{VY} \times \textit{STATUS} \times \textit{OC} \\ \forall r \in \textit{Robot}, e \in \textit{Environment}.r[2] == \langle 0, 0 \rangle \wedge c[4] \neq \lambda \end{array}$$

The *Robot* state schema is defined by a data with five fields. The first field specifies the original coordinates of the robot, second and third place specify the current location,

while the last field specifies the robot status, which includes *forward*, *backward*, *moving*, *spinning*, *turnleft*, *turnright*, and *Uturn*.

Now we can define sensor component based on the above two schemas. The sensor component includes several schemas of the various sensors. The schema of sensor highly depends on the sensing mechanism. In this paper, we define ultrasonic sensor and light sensor based on the LEGO NXT Mindstorm toolkit as follows.

$$\begin{array}{l} \textit{UltrasonicSensor} \\ \textit{UltraSensor} : \textit{FREQUENCY} \times \textit{DIST} \\ \forall u \in \textit{UltraSensor}.u[2] \leq 1.5 \wedge u[1] \neq \lambda \end{array}$$

In the above schema of *UltrasonicSensor*, if there is an obstacle within 1.5m, then there is frequency returned and the obstacle can be detected.

$$\begin{array}{l} \textit{LightSensor} \\ \textit{LightSensor} : \textit{FREQUENCY} \times \textit{COLOR} \\ \forall l_1, l_2 \in \textit{LightSensor}.l_1[1] \neq l_2[1] \wedge l_1[2] \neq l_2[2] \end{array}$$

In the schema of *LightSensor*, it is expected that for all different colors the light sensor can return different frequencies. The schema of *Sensor* is a composite schema and can be defined as

$$\textit{Sensor} \triangleq \textit{UltraSensor} \oplus \textit{LightSensor}$$

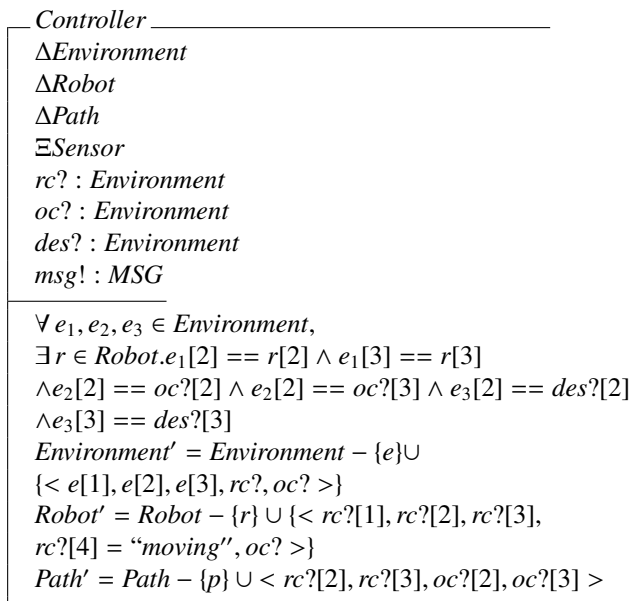
A path of a robot can be defined as the distance from source to destination. A path needs to be updated every time interval due to the environment changes. The robot needs to remember the recent path. The calculation of path is the line between two points – current position and the destination. Considering one obstacle between the current position and the destination, a path is two coordinates for robot and destination.

$$\begin{array}{l} \textit{Path} \\ \textit{Path} : \textit{HX} \times \textit{VY} \times \textit{HX} \times \textit{VY} \\ \forall p \in \textit{Path}, e_1, e_2 \in \textit{Environment}. \\ p[1] == e_1[2] \wedge p[2] == e_1[3] \wedge \\ p[3] == e_2[2] \wedge p[4] == e_2[3] \end{array}$$

The controller schema can be complicated and highly depends on the requirements and tasks of the robot. The controller takes input from sensor, perform some tasks and output the commands to actuators to drive the robot. The task can be calculation of the shortest path, uploading some devices to help movement (e.g. a pad to the foot so that the robot can walk cross the river.), and/or starting other motors (rescue people). In this framework, we only present the shortest path calculation so that the robot can avoid the collision with obstacle and move to the destination.

The system needs current robot position ($rc?$) and obstacle position ($oc? = \langle x, y \rangle$). In addition, the frequency of the ultrasonic sensor ($freq?$) is required for the sensor to return the knowledge of detected object and ($des?$) denotes the destination position that is retrieved from memory. The precondition for the robot to calculate the correct moving path is the detection of obstacle is precisely evaluated. The path includes two parts – $path1$ represents the distance from the robot to the obstacle, while $path2$ describe the distance from obstacle to the destination ($des?$).

The preconditions for the controller are: i) the robot, obstacle and the destination are all identified in the coordinate systems (*Environment*); ii) the value returns from the sensor matches the value for the obstacle. After the controller calculates the path, the new paths should be generated, commands are sent to motors to turn in a certain angle and move the robot.



The block diagram that describes the architecture of the framework is shown in the Figure 1. Fig. 1 show a generic view and summarizes the relations of the above well defined schemas.

5. Case Study Using LEGO Toolkit

The Mindstorms NXT brick uses a 32-bit ARM processor as its main processor, with 256 kilobytes of flash memory available for program storage and 64 kilobytes of RAM for data storage during program execution. To acquire data from the input sensors, another processor is included that has 4 kilobytes of flash memory and 512 bytes of RAM. Two motors can be synchronized as a drive unit. To give the robot the ability to “see,” the ultrasonic sensor, which is accurate to 3 centimeters and can measure up to 255 centimeters, and the light sensor, which can distinguish between light and

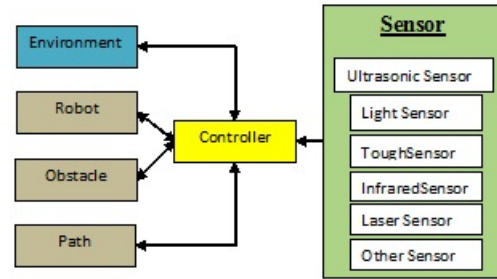


Fig. 1: The Z Notation Framework for Autonomous Mobile Systems

dark, can be attached to the brick. A sound sensor that can be adjusted to the sensitivity of the human ear can be used to give the robot the ability to hear and react, if programmed, to noises. Finally, the two touch sensors give the ability for a robot to determine if it has been pressed, released, or bumped, and react accordingly [1].

As a replacement for the standard Lego firmware, the LeJOS project has support for threading, arrays, recursion, synchronization, exceptions, non-generic data structures, standard data types, and input and output [11]. The LeJOS virtual machine supports much of the java.util package, but the data structures require that data be stored as type Object and then cast to a type that inherits Object. For input and output, both streams and sockets are available for use. For control purposes, the LeJOS platform supports the direct connection of Bluetooth-enabled GPS units for spatial location information and keyboards for the navigational control of a robot [11].

5.1 Field Guard Robot (FGR)

LEGO NXT Mindstorm is a highly integrated toolkit for the robot educators and learners in different levels. The main features are i) is easy to assembly and ii) supports multiple interfaces and platforms for various levels of users. Java is supported by LEGO NXT toolkit by a plug-in package named LeJOS, a source forge project created to develop a technological infrastructure for LEGO Mindstorms products. LEGO NXT has three output ports to drive motors, and four input ports for data acquiring from sensors. To validate and apply the above formal framework, a LEGO NXT field guard robot (FGR) was assembled to search the objects in a certain field by patrolling around. The robot is going to make a buzz for six times and starts spinning in the center of the guarded area continuously looking for an object within the specified area around it; If it is found any object within the area, it will take and push the object to a certain area outside the guarded place. After that it comes back to the center of the area and continue to spin.

5.2 Modeling of FGR In Z

To ensure the system works correctly with expected behaviors, we apply the above Z notation framework on this LEGO NXT Mindstorm field guard robot (FGR). The main schemas include followings – Environment, Robot, UltrasonicSensor, TouchSensor, Pickup and Drop. Since we use the coordinate system for the robot navigation, the schema of *Environment*, *Robot* and sensors (*UltrasonicSensor* and *LightSensor*) are same as those defined in section 4. In this section a new sensor *TouchSensor* and some operation schemas will be discussed in this section. The touch sensor is used to react to the obstacle on the moving path. It responds only to the surface interaction with the object, and can be defined as following:

<i>TouchSensor</i>
$\Delta TouchSensor : VALUE \times STATUS$
$\forall t \in TouchSensor.t_1[1] == \lambda \wedge t[2] == false$

The data field for the touch sensor are the value read from touch sensor and the status of object existence. The status is a boolean value, where true indicates that the object is detected, otherwise, no object. The constraint of touch sensor is that if there is no touch with object, the status returned from touch sensor is false, which indicates no object in front of the sensor.

Two basic operations of the FGR can be pick_up and drop the object on the path. Besides, we need to define the moving operation to instruct the robot move correctly. For the space limitation, we only show the pick and drop operation schemas, which are defined as follows:

$Pickup \triangleq Robot \times Controller \times Sensor \times Environment \rightarrow Environment$, and

$Drop \triangleq Robot \times Controller \times Sensor \times Environment \rightarrow Environment$

The initialized Robot schema can be:

<i>Robot_Init</i>
$\Delta Robot$
$\exists Environment$
$\langle \langle 5, 5 \rangle, 5, 5, start, \lambda \rangle$

Considering the success and failure of each operation, the pickup and drop operation can be defined as composite schemas, where

$Pickup \triangleq Pickup_Success \vee Pickup_Failure$, and

$Drop \triangleq Drop_Success \vee Drop_Failure$ where

There are two preconditions specified in the schema of *Pickup_Succ*: i) the robot, object and destination are all in the environment (identified by the coordinate system) and ii) the touch sensor needs to interact with the object when pick up. After pick up the object, the path needs to be updated immediately. Similarly, if this operation is failed, it can be

defined as following schema:

In the drop operation, to simplify the case, let assume there is no obstacle on the moving path. Thus, we do not need to update the path from the obstacle to destination. Otherwise, another schema for moving needs to be defined. The drop schema can be defined as following.

<i>Pickup_Success</i>
$\Delta Robot$
$\Delta Environment$
$\Delta Path$
$\exists TouchSensor$
$rc? : ROBOT$
$oc? : ENVIRONMENT$
$des? : ENVIRONMENT$
$value? : VALUE$
$msg! : MSG$
$\forall e_1, e_2, e_3 \in Environment, \exists r \in Robot.$ $e_1[2] == rc?[2] \wedge e_1[3] == rc?[3]$ $\wedge e_2[2] == oc?[2] \wedge e_2[3] == oc?[3] \wedge$ $e_3[2] == des?[2] \wedge e_3[3] == des?[3] \wedge$ $\forall s \in TouchSensor.s[1] == value?$ $Robot' = Robot - \{r\} \cup \{r[1], r[2], r[3],$ $r[4] = pickup, r[5] = oc? \}$ $Environment' = Environment' - \{e\} \cup \{e[1], e[2],$ $e[3], e[4] = rc?, e[5] = oc? \}$ $Path' = Path - \{p\} \cup \{r[2], r[3], oc?[2], oc?[3] \},$ $\langle oc?[2], oc?[3], des?[2], des?[3] \rangle$ $msg! = "Objectpickedup!"$

<i>Pickup_Failure</i>
$\exists Robot$
$\exists Environment$
$\exists Path$
$\exists TouchSensor$
$rc? : ROBOT$
$oc? : ENVIRONMENT$
$des? : ENVIRONMENT$
$value? : VALUE$
$msg! : MSG$
$\forall e_1, e_2, e_3 \in Environment, \exists r \in Robot.$ $(e_1[2] \neq rc?[2] \wedge e_1[3] \neq rc?[3])$ $\vee (e_2[2] \neq oc?[2] \wedge e_2[3] \neq oc?[3]) \vee$ $(e_3[2] == des?[2] \wedge e_3[3] == des?[3]) \wedge$ $\forall s \in TouchSensor.s[1] \neq value?$ $msg! = "NoObject"$

The reason that the *Path* data was not updated because the *Drop* schema only takes care from the found obstacle to the destination where the path had been defined in the NXT controller. In this design, LEGO robot uses light sensor to identify the destination place by placing a certain color of

line in a place. Next, we define the failure case of drop operation.

Drop_Success

$\Delta Robot$

$\Delta Environment$

$\exists Path$

$\exists LightSensor$

$rc? : ROBOT$

$des? : Environment$

$status? : STATUS$

$value? : VALUE$

$msg! : MSG$

$\forall e_1, e_2 \in Environment, \exists r \in Robot.$

$(e_1[2] == rc?[2] \wedge e_1[3] == rc?[3]) \wedge$

$e_2[2] == des?[2] \wedge e_2[3] == des?[3] \wedge$

$\forall s \in LightSensor.s[1] == value?$

$\forall r \in Robot.r[5] = status?$

$Robot' = Robot - \{r\} \cup \{< r[1], r[2], r[3],$

$r[4] = "drop", r[5] = des? >\}$

$Environment' = Environment - \{e_1\} - \{e_2\} \cup \{< e[1],$

$e[2], e[3], e[4] = des?, e[5] = \lambda >\}$

$msg! = "Objectisdropped!"$

Drop_Failure

$\Delta Robot$

$\exists Environment$

$\exists LightSensor$

$rc? : ROBOT$

$des? : Environment$

$status? : DOUBLE$

$value? : VALUE$

$msg! : MSG$

$\forall e_1, e_2 \in Environment, \exists r \in Robot.$

$(e_1[2] \neq rc?[2] \wedge e_1[3] \neq rc?[3]) \vee$

$e_2[2] == des?[2] \wedge e_2[3] == des?[3] \vee$

$\forall s \in LightSensor.s[1] \neq value?$

$\forall r \in Robot.r[4] \neq status?$

$msg! = "Noobjecttodrop!"$

There are several conditions that drop operation can be failed. For example, the robot is not in the coordinate system (not in the environment defined); the destination is not in the coordinate system, the sensor cannot detect the destination (by color), or the robot does not move (status is not properly set up to the correct value). There is one case that the robot can fail and not defined in the above schema – object is not hold in the claws. It is mostly caused by hardware imprecision not software controller based on our observation.

5.3 Implementation and Discussion of LEGO FGR

The LEGO FGR was implemented in Java. The set up includes the Eclipse IDE, LeJOS plug in for the APIs defined for LEGO robot accessories and conversion of from Java code to NXT brick. The java implementation of the LEGO FGR can realize all defined functionalities and finish expected requirements.

From the implementation of LEGO NXT robot in Java, we found that:

- 1) The model of robot is very important for the description and precisely implement the functions. There are some cases was missed in the code but defined in the model during implementation.
- 2) On the other side, it is noticed that the above model is a formal framework for the robot design. Due to limitation, the coordinate system and robot may need more detail description.
- 3) LEGO is a highly integrated tool kit. Some data fields have been specified properly without definition. A more sophisticated setting is needed for the more research study.

6. Conclusions and Future Works

We have presented a frame of Z notation for modeling the dominant design paradigms used in autonomous mobile systems. We have done this by taking advantage of formal specification languages to allow for navigation system, network connectivity and proactive process migration. We have also used the set theory and first order logics our models. In addition, the modular approach is used for the framework so that the robotics system with navigation can be modeled incrementally, exploiting commonalities among the design paradigms and reusing the defined state schemas and improving the system maintenance. Modularity can similarly be used to extend our models to represent their realization with specific technologies, or to capture their use in specific applications.

The formal basis of Z notation allows us to use the models and their potential extensions to reason about properties of mobile systems. This process can be aided by taking advantage of the substantial reuse of classes within our framework, and the particular form to which our specifications conform. Future work will look at how these aspects of the models can be exploited to simplify reasoning and, hence, the development of suitable reasoning support tools.

Acknowledgment

The authors would like to thank all reviewers for the kindly comments and suggestions on this work.

References

- [1] B. Bagnall. *Maximum Lego NXT: Building Robots with Java Brains*. Variant Press, 2007.
- [2] T. Balch and R. Arkin. Behavior-based formation control for multirobot teams. *Robotics and Automation, IEEE Transactions on*, 14(6):926 – 939, dec 1998.
- [3] T. Balch and M. Hybinette. Social potentials for scalable multi-robot formations. In *Proceedings of the IEEE International Conference on Robotics and Automation, ICRA '00*, volume 1, pages 73–80, 2000.
- [4] W. P. Jonathan L. Gross, Jay Yellen. *Handbook of graph theory*. CRC Press, December 2003.
- [5] P. Leita, A. Colombo, and F. Restivo. An approach to the formal specification of holonic control systems. In V. Marik, D. McFarlane, and P. Valckenaers, editors, *Holonic and Multi-Agent Systems for Manufacturing*, volume 2744 of *Lecture Notes in Computer Science*, pages 1090–1090. Springer Berlin / Heidelberg, 2003.
- [6] F. A. Melo, P. Lima, and M. I. Ribeiro. Event-driven modelling and control of a mobile robot population. In *Proceedings of the 8th Conference on Intelligent Autonomous Systems*, pages 237–244, 2004.
- [7] T. Murata, P. C. Nelson, and J. Yim. A predicate-transition net model for multiple agent planning. *Information Sciences*, 57–58:361–384, 1991.
- [8] N. S. Nazir Ahmad Zafar and A. Ali. Construction of intersection of nondeterministic finite automata using z notation. In *Proceedings of World Academy of Science, Engineering and Technology*, volume 30, pages 96–101, 2008.
- [9] R. Simmons and S. Koenig. Probabilistic robot navigation in partially observable environments. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI '95)*, pages 1080 – 1087, July 1995.
- [10] J. M. Spivey. *The Z Notation: A Reference Manual*. <http://www.rose-hulman.edu/class/csse/cs415/zrm.pdf>, 2e edition.
- [11] L. Team. Nxj technology. <http://lejos.sourceforge.net/nxj.php>.
- [12] D. Xu, R. A. Volz, T. R. Ioerger, and J. Yen. Modeling and verifying multi-agent behaviors using predicate transition nets. In *Proc. of the 14th International Conference on Software Engineering and Knowledge Engineering*, pages 193–200. ACM Press, 2002.

Implementing a Sound Mapping from Normative Requirements to Event-B Design Blueprints Using MDA

Iman Poernomo¹, Timur Umarov²

¹Department of Computer Science, King's College London, Strand, London, UK, WC2R2LS
(iman.poernomo@kcl.ac.uk),

²Department of Computer Engineering, Kazakh-British Technical University,
59 Tole bi, Almaty 050000 Kazakhstan
(t.umarov@kbtu.kz)

Abstract—*This paper addresses the problem of implementing a mathematically sound MDA transformation from requirements to design. The question of validation is particularly important as preserving the semantics of the systems requirements across the transformation is a problem that has been addressed by the MDA community for long. The soundness check is performed over the transformation from normative requirements specification into Event-B models. The business requirement specification as a computation independent model of MDA is described by an ontology model and an associated set of normative rules that define the ways in which business processes can interact. MDA transformation uses normative requirements to generate Event-B models, a platform independent model, which is very close to a typical loosely coupled component-based implementation of a business system workflow. This paper is based on the previous works of the authors and specifically addresses the soundness problem of the transformation.*

Keywords: MDA, model transformation, normative requirements, formal methods, business processes

1. Introduction

Business process management (BPM) is an increasingly challenging aspect of the enterprise. Middleware support for BPM, as provided by, for example, Oracle, Biztalk and the recent Windows Workflow Foundation (WWF), has met some challenges with respect to performance and maintenance of workflow.

A business process implementation within a BPM middleware requires detailed treatment of both information flow and information content. The abstraction gap is identified by Hepp and Roman in [4]: an abstract workflow that ignores information content provides an abstract view of business processes that does not fully define the key aspects necessary for BPM implementation. We argue in [3] that this abstraction gap can be addressed by developing event-driven data models in the Event-B language from an initial business process requirements specification. We have employed a Model Driven Architecture approach. The complete description of the formal transformation function

ϕ is provided in [2]. In [3], for CIM we use the ontologies and normative language of the MEASUR method [1]. Our approach in [3] addresses the semantic gap by defining a formal transformation of MEASUR models to Event-B machines, permitting: (i) a full B-based formal semantics for vocabularies and data manipulation that is carried out within the modeled workflow, which is validated for consistency; and (ii) an initial, abstract B model can potentially be refined by B-method to a final optimal executable system in an object-oriented workflow middleware.

In this paper, we are ensuring *semantic compatibility* between the normative MEASUR models and the Event-B blueprints, i.e. semantics we have given for norms is *preserved* in some form by the semantics given by the MDA transformation. This relates the research program proposed by Poernomo in [5], where MDA transformations always involve some form of check to ensure that what is meant is preserved across the transformation. A notion of semantic compatibility holds over the transformed models, so that any property derived over the normative-ontological view of the system will hold over potential processes that arise from the Event-B machine. We address this problem in Theorem 4.1 below. This semantic preservation check can potentially be applied for our previous MDA transformations described in [3], [6], [7].

The paper proceeds as follows:

- In section 2, we shortly describe our CIM-to-PIM transformation.
- Section 3 provides a brief introduction to the soundness of refinement.
- Section 4 provides a semantic preservation theorem and its proof.
- Section 5 shortly discusses conclusions, related work and future work.

2. Background on the MDA Transformation

In this section, we provide an informal description of the mapping strategy of our normative rules to the Event-B machines, which is described in [3]. The CIM-to-PIM

transformation are the ones that draw much attention of the MDA community. Automation of this leg of transformation is not always possible due to high level of abstraction of the models involved. There were several successful attempts to automate CIM-to-PIM transformation described in [8], [11], [12], [13]. However, implementation of the semantically sound and automatic CIM-to-PIM transformation still represents a scientific interest among software engineers.

In MDA terms, ontologies and norms are a computation independent model (CIM) and Event-B machines represent platform independent model (PIM).

The mapping of affordances is straightforward. The general framework is shown in Fig. 1. All the elements of the source and target models are marked with different patterns to be able to differentiate separate mappings of the subclasses of affordances to Event-B constructs: agents are mapped to machines, business entities and relations are mapped to Event-B sets, relations and state variables, and communication acts are mapped to events. The transformation of normative constraints is more difficult. Conceptually, norms of the form

$$(\text{Trigger} \wedge \text{pre-condition}) \rightarrow E_{\text{agent}} \text{Ob/Pe/Im post-condition.} \quad (1)$$

appear similar in form to a machine event:

- A *trigger* and *pre-condition* correspond to a *guard*. The former defines the situation that must hold before an agent can act. The latter defines the state that must hold before a machine can perform an action.
- The responsibility modality E_a corresponds to the location of the event within the machine corresponding to agent a .
- The deontic modality *Ob/Pe/Im post-condition* identifies whether the action corresponding to *post-condition* should be necessarily performed, or whether execution of another (skip action) is possible instead. The *Im* deontic modality means the negation of the post-condition holds.

Because the normative constraints are essentially abstract business rules, while the conditions of the B machine define further implementation-specific detail, the mapping will depend on how we interpret relations and functions of the ontology. For this purpose our transformation must be based on a given semantic mapping of individual relations and functions to B relations and functions. We assume this is defined by a domain expert with the purpose of wide reusability for the ontology's domain.

Models specified using the Event-B language are comprised of two main building structures: *machine* and *context*. The relationship between these constructs is that machine *sees* the context, i.e. machine can use the content by having a read-only access to it without an ability to modify it. Machines have a dynamic content, which means that its

Table 1: Types of generalised substitutions, where A_1 and A_2 are arbitrary generalised substitutions.

Type of Substitution	Realisation in Event-B
Deterministic	$x := f$
Sequential	$A_1; A_2$
Empty	<i>skip</i>

elements such as *events* may modify the state variables. Contexts only contain static information which is used by machines and never changes.

Fig. 2 depicts the structure of the Event-B model consisting of *Context* and *Machine*. It is shown that context contains carrier sets, constants, axioms and theorems. Carrier sets mainly describe typing information for state and local variables. One can also define a carrier set and assign to it a limited number of constants in which case we obtain an enumeration set. Axioms in this case are used to specify these constants as distinct values.

Because the language is essentially first order logic, the intended meaning of the context/theory can be understood by anyone with a basic knowledge of first order logic and set theory. This usability is one of the motivating arguments behind the simplicity of the Event-B language.

An Event-B machine is comprised of four elements:

- an (optional) context, whose sets, constants and axioms may be used to define properties over the machine;
- variables that effectively define the state of the machine;
- invariants, logical formulae that define what must *always* hold over the execution of the machine; and
- events (or as otherwise referred in [19], a collection of transitions) that define how the machine is allowed to behave in the face of particular guard conditions being met over its state.

The events consist of three parts, as shown in Fig. 3.

- name E ,
- guard G , logical statements over machine variables,
- *action* S , also known as a *generalised substitution* over machine variables.

Syntactically, an event E is written

$$E \hat{=} \text{WHEN } G \text{ THEN } S \text{ END.}$$

Generalised substitution consists simply of assignment of state variables to new values, resulting in changing the state of the whole machine. When used within an event, a guard defines a condition when substitution should occur. These generalised substitutions play a role of modifying the state of machines by updating state variables. In Event-B, there are three types of generalised substitutions: deterministic, sequential, and empty. Table 1 depicts how all three types of generalised substitutions are realised in Event-B. The informal meaning of assignment should be clear. Empty substitution is specified as *skip* and it does nothing by skipping the execution.

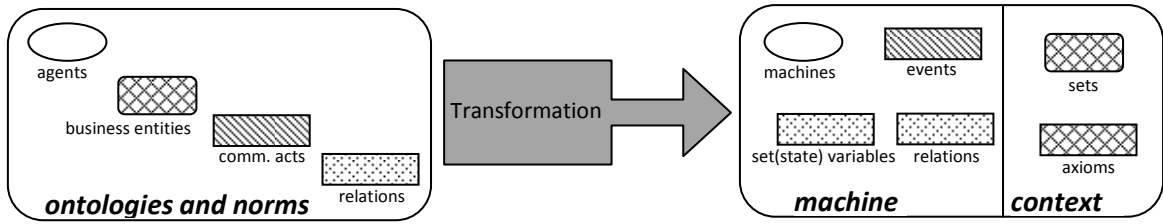


Fig. 1: The general framework of the transformation

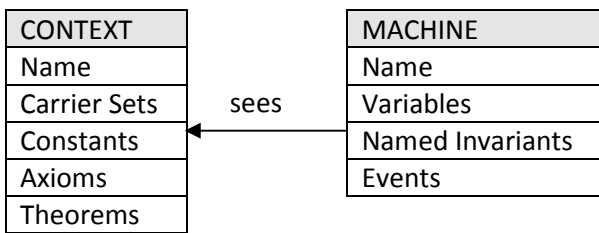


Fig. 2: Context and machine relationship in Event-B

EVENT
Name
Named Guards
Generalised Substitution

Fig. 3: Structure of events in Event-B

Machines have a formal operational semantics that models system execution as a sequence of events and changes to state. If an event's guard holds over the machine's state, its action may be executed. This will change machine's state, which may cause another event's guard to hold, and an action to be executed. The sequence continues until the system has halted (deadlocked). Note that execution is potentially nondeterministic: when a number of event guards are true, then *one* of the corresponding event actions is chosen at random. A deterministic machine is one in which each event guard mutually excludes the possibility of the other guards.

3. Soundness of Transformation

The set of *models* of a normative requirements specification denote possible kinds of *implementations* of these requirements: some closer to a real, executing system than others. For example, one possible model would take the interpreting sets for affordances to denote actual computational entities, such as Java data types for unit values and Java class objects for agents that contain possible communication acts

as potential methods for interaction. Another possible model might treat communication acts as mathematical functions and agents as abstract values in and of themselves, that simply save relationships with values and communication acts, divorced from a component, modular or agent-based view.

If there is no model for a set of requirements, then this means that there is no possible implementation. Similarly, the set of models for an Event-B design specification constitutes a set of possible implementations. Again, some models will be closer to an executable system than others. For example, one model might take an understanding of states and execution traces to be precisely the states and execution traces contained within a .NET WWF implementation of the Event-B machines, taking agents as Web Services. A more abstract model might be not committed to considering states as exactly .NET datatype values.

We do not consider an immediate implementation but, following the MDA demarcation, are interested in how ϕ might be used to move from one level of abstraction (our normative CIM) to a level somewhat closer to implementation (an Event-B PIM). A more formal way of putting this question is: Given a set of normative *requirements* as a CIM, automatically transformed into an Event-B *design* is PIM, if the PIM has an implementation, will this implementation serve as an implementation of the CIM? In what follows, we will attempt to prove the corresponding semantic preservation theorem.

4. Semantic Preservation Theorem

In this section, we are providing a theorem for semantic preservation and a proof. Let us first define a particular Kripke model from the possible execution traces of the B specification generated from a normative specification.

Definition 4.1 (Kripke model) Assume *REQ* is a deterministic set of behavioural norm/definition pairs, each of the form

$$REQ = \{(G_i \rightarrow E_{b_i} \text{ Ob } A_i, \forall x_1 : T_1, \dots, x_n : T_n. A_i \rightarrow DEF_i \mid i = 1, \dots, n)\}$$

Let S be the set of machines generated by applying ϕ over each norm in REQ that is

$$S = \phi(REQ)$$

Consider any norm/definition tableaux

$$N = (G \rightarrow E_b \text{ Ob } A, \forall x_1 : T_1, \dots, x_n : T_n A \leftrightarrow DEF) \in REQ$$

Take any model \mathcal{M}_{PIM} that satisfies S . Assume

$$\langle UNIT, ENTITY, AGENT, CN, COMMACT, RN, REL, DETERM \rangle$$

be the relational ontology over which REQ is defined. We first define the associated normative model is tuple

$$\mathcal{M}_{CIM} = \langle F, V, h, h_{AN}, h_{AV}, h_{ID}, h_{EAPP}, h_{ACT}, h_{REL}, =^M \rangle$$

- V is the union of sets of interpreting values of units, identifiers, agents, agent names, entities and entity names, defined as

$$\bigcup \{V_{UNIT}, V_{ID}, V_{AN}, V_{AGENT}, V_{EN}, V_{ENTITY}\}$$

where we define

$$V_{AGENT} = LIST(V_{UNIT})$$

and

$$V_{ENTITY} = V_{ID} \times LIST(V_{UNIT})$$

and

$$V_{UNIT} = \{b \in B \mid b = \iota(\text{UNITtoB}(t)) \text{ for any interpretation } \iota \text{ over } \mathcal{M}_{PIM}, \text{ any } t \in \text{Term}_{UNIT}\}$$

the interpretation of $\iota(\text{UNITtoB}(t))$ should always be the same, the closure of h_{PIM} , because any element of Term_{UNIT} will be mapped only to functions and constant symbols, without variables. We let V_{ID} be exactly the set of entity identifiers, respectively, taken from the normative ontology, so that $h_{ID}(i) = i$. We let V_{AN} be exactly the set of Event-B machine names.

- F is a normative Kripke frame, defined

$$F = \langle W, R \rangle$$

where

- W is a non-empty set, consisting of all non-intermediate states of the Event-B specification
- R is a trinary relationship between two elements of W and V_{AGENT} instances $R \subset W \times W \times V_{AN}$, and defined so that

$$R(\sigma, \sigma', a)$$

if, and only if,

$$\langle S, \sigma \rangle \xrightarrow{*} \langle S, \sigma_1 \rangle \xrightarrow{*} \dots \xrightarrow{*} \langle S, \sigma_n \rangle \xrightarrow{M_{a,e}} \langle S, \sigma' \rangle$$

for some event e (where a is a name of an Event-B machine, the same name as the agent identifier used in the CIM model). For this to be a model, we are required to show that R is serial, so that, for any $\sigma \in W$ and $a \in V_{AN}$ there exists at least one $\sigma' \in W$ and one $a \in V_{AN}$ such that $R(i, j, a)$ holds. This is clearly the case, because we have only taken states where such a relationship holds by definition.

- h_{ACT} is a function from a communication act name from CN and worlds of F to possible interpreting values of the agents and entities that the act might predicate over:

$$h_{ACT} : (CN \times W) \rightarrow (V_{AGENT} \times V_{AGENT} \times V_{ENTITY})$$

defined

$$h_{ACT}(CN, \sigma) \ni (a_1, a_2, e)$$

for any a_1, a_2 and e satisfying:

- Let $DEF(x_1, x_2, x_3)$ be the definition associated with the communication act CN , defined in a norm $N \in REQ$ as

$$CN(x_1, x_2, x_3) \leftrightarrow DEF(x_1, x_2, x_3)$$

for agent variables $x_1 : A_1\{i_1 : T_1, \dots, i_m : T_m\}$, $x_2 : A_2\{j_1 : T_1, \dots, j_n : T_n\}$, and entity variable $x_3 : E\{k_1 : T_1, \dots, k_o : T_o\}$ so that

$$COMMACT(CN) =$$

$$(A_1\{i_1 : T_1, \dots, i_m : T_m\}, A_2\{j_1 : T_1, \dots, j_n : T_n\}, E\{k_1 : T_1, \dots, k_o : T_o\})$$

- It must be the case that

$$\sigma(x_1) = a_1 = \{v_1^i, \dots, v_m^i\}$$

$$\sigma(x_2) = a_2 = \{v_1^j, \dots, v_n^j\}$$

$$\sigma(x_3) = e = (i, \{v_1^k, \dots, v_o^k\})$$

with

$$\mathcal{M}_{PIM}, \sigma \models \text{toB}(DEF(x_1, x_2, x_3))$$

- For agent variable $a : A\{i_1 : T_1, \dots, i_m : T_m\}$

$$h(a, \sigma) = \{v_1^i, \dots, v_m^i\}$$

if, and only if,

$$\sigma(i_1) = v_1^i, \dots, \sigma(i_m) = v_m^i$$

- For entity variable $e : E\{i_1 : T_1, \dots, i_m : T_m\}$

$$h(e, \sigma) = (id, \{v_1^i, \dots, v_m^i\})$$

if, and only if,

$$\sigma(i_1)id = v_1^i, \dots, \sigma(i_m)id = v_m^i$$

- h_{REL} is a function from relationship names RN and worlds of F to possible interpreting values of the agents and/or entities that the act might predicate over:

$$h_{REL} : (RN \times W) \rightarrow (V_{UNIT} \cup V_{AGENT} \cup V_{ENTITY} \times V_{UNIT} \cup V_{AGENT} \cup V_{ENTITY})$$

defined

$$h_{REL}(R, \sigma) \ni (b_1, b_2)$$

if, and only if, whenever

– If

$$REL(R) \equiv (A_1 \{i_1 : T_1, \dots, i_m : T_m\}, A_2 \{j_1 : T_1, \dots, j_n : T_n\})$$

where A_1, A_2 are agent type names,

$$b_1 = h(x_1, \sigma) = \{v_1^i, \dots, v_m^i\}$$

$$b_2 = h(x_2, \sigma) = \{v_1^j, \dots, v_n^j\}$$

and

$$\mathcal{M}_{PIM}, \sigma \models \text{toB}(R(x_1, x_2))$$

– If

$$REL(R) \equiv (E_1 \{i_1 : T_1, \dots, i_m : T_m\}, E_2 \{j_1 : T_1, \dots, j_n : T_n\})$$

where E_1, E_2 are entity type names,

$$b_1 = h(x_1, \sigma) = (id_1, \{v_1^i, \dots, v_m^i\})$$

$$b_2 = h(x_2, \sigma) = (id_2, \{v_1^j, \dots, v_n^j\})$$

and

$$\mathcal{M}_{PIM}, \sigma \models \text{toB}(R(x_1, x_2))$$

- If $REL(R) = (T_1, T_2)$ for R a relationship defined between unit types and

$$b_1 = h(x_1, \sigma)$$

$$b_2 = h(x_2, \sigma)$$

where

$$\mathcal{M}_{PIM}, \sigma \models \text{toB}(R(x_1, x_2))$$

- each element $t \in \text{Term}_{UNIT}$ is mapped to an element of $h(t, w) \in V_{UNIT}$:

$$h(t, w) = \iota(\text{UNITtoB}(t)) \in V_{UNIT}$$

for any interpretation into \mathcal{M}_{PIM} – noting again that all interpretations will give the same value, the closure of h_{PIM} .

- each entity variable is mapped to an element of V_{EN}

$$h(e, w) = e.$$

Let us now prove the semantic preservation theorem.

Theorem 4.1 (Semantic preservation) Assume REQ is a deterministic set of behavioural norm/definition pairs. Let S be the set of machines generated by applying ϕ over each norm in REQ that is $S = \phi(REQ)$.

Consider any norm/definition tableaux

$$N = (G \rightarrow E_b \mathbf{M} A, A \leftrightarrow DEF) \in REQ$$

for modality \mathbf{M} set to Ob or Pe (“obligatory” or “permissible”, respectively).

Take any model \mathcal{M}_{PIM} that satisfies S . The execution traces of \mathcal{M}_{PIM} for S form the Kripke model \mathcal{M}_{CIM} , as defined above. It is then the case that

$$\mathcal{M}_{CIM} \models G \rightarrow E_b \mathbf{M} A$$

Proof.

We first consider the case where the modality \mathbf{M} is obligation Ob . We need to show that given $N = (G \rightarrow E_b Ob A, \forall x_1 : T_1, \dots, x_n : T_n A \rightarrow DEF) \in REQ$ it is the case that $\mathcal{M}_{CIM} \models G \rightarrow E_b Ob A$.

First, assume that, for any σ , and agent a

$$\mathcal{M}_{CIM} \models_{\sigma}^a G \quad (2)$$

By the definition of our model, we know that σ is non-intermediate: that is

$$\mathcal{M}, \sigma \models R_N = \perp \quad (3)$$

for any flag variable R_N , any norm $N \in R$. We are required to show that, for any σ' (such that $R(\sigma, \sigma', b)$) $\mathcal{M}_{CIM} \models_{\sigma}^b A$. That is, we must show that, if state σ' satisfies

$$\langle S, \sigma \rangle \xrightarrow{*} \langle S, \sigma_1 \rangle \xrightarrow{*} \dots \xrightarrow{*} \langle S, \sigma_n \rangle \xrightarrow{M_b^e} \langle S, \sigma' \rangle \quad (4)$$

then it is the case that $\mathcal{M}_{CIM} \models_{\sigma}^b A$.

We proceed by induction over the possible form of G .

- Assume $G \equiv P(e_1, e_2)$ between two entity variables $e_1 : E_1 \{v_1^1 : T_1^1, \dots, v_m^1 : T_m^1\}$ and $e_2 : E_2 \{v_1^2 : T_1^2, \dots, v_m^2 : T_m^2\}$. The assumption (2) entails that

$$h_{REL}(P, \sigma) \ni (h(e_1, \sigma), h(e_2, \sigma)) \quad (5)$$

for $h(e_1, \sigma) = (n_1, \{u_1^1, \dots, u_m^1\})$ and $h(e_2, \sigma) = (n_2, \{u_1^2, \dots, u_m^2\})$. By the definition of h_{REL} , (5) entails that

$$\mathcal{M}_{PIM}, \sigma \models \text{toB}(P(e_1, e_2)) \quad (6)$$

- If $G \equiv P(e, c)$, where $e : E\{\dots\}$ and $c : C\{\dots\}$ are entity and agent variables, respectively, so that $E\{\dots\} \in ENTITY$ and $C\{\dots\} \in AGENT$, then the assumption (2) entails that

$$h_{REL}(\sigma, P) \ni (\sigma(e), \sigma(c)) \quad (7)$$

By the definition of h_{REL} , (7) entails that

$$\mathcal{M}_{PIM}, \sigma \models e \in P \quad (8)$$

- If $G \equiv P(e.f_i, v)$, where $e : E\{f_1 : T_1 \dots f_n : T_n\}$ and $v : T \in \text{Term}_{UNIT}$ are an entity variable and a closed

UNIT term, respectively, then the assumption (2) entails that

$$h_{REL}(P, \sigma) \ni (h(e.f_i, \sigma), \text{UNITtoB}(v)) \quad (9)$$

where $\sigma(e) = (id, \{v_1, \dots, v_i, \dots, v_n\})$ and $h(e.f_i, \sigma) = v_i$. By the definition of h_{REL} , (9) entails that

$$\mathcal{M}_{PIM}, \sigma \models R(f_i(e), \text{UNITtoB}(v)) \quad (10)$$

- The other atomic cases are similar. (Note that G is the guard of a normative tableaux: consequently, it does not contain an instance of a communication act). The cases involving connectives follow trivially by the induction.

In each case, we have that

$$\mathcal{M}_{PIM}, \sigma \models \text{toB}(G) \quad (11)$$

We need to use this condition to show that the guard condition of event $event_N$ in machine $M_B \in \phi(REQ)$ will *always* be invoked after a finite number of steps. To show this, we proceed by working over two possible forms of the action A :

- If A is of the form $R(p, b', b)$, where $p : P\{\dots\}$, $P\{\dots\} \in ENTITY$, $b : B\{\dots\}$, $B\{\dots\} \in AGENT$, $b' : B'\{\dots\}$, $B'\{\dots\} \in AGENT$ and $R \in COMMACT$ and where the two agent types are *different*, so $B\{\dots\} \neq B'\{\dots\}$, then we take the post-condition $E_b \text{ Ob } R(p, b', b)$. By the definition of ϕ , we know that there is an event $event_N$ in $M_B \in S$ of the form

$$\begin{array}{l} \lceil \text{Event } event_N \hat{=} \quad \rceil \\ \text{WHEN } GUARD_N(G)^* \quad \in b \\ \text{THEN } \phi_N(D); R_N := \perp \end{array} \quad (12)$$

where

$$\begin{aligned} GUARD_N(G)^* &\equiv GUARD_N(G)/G' \wedge R_N = \top \equiv \\ &\text{toB}(G)/G' \wedge R_N = \top \end{aligned} \quad (13)$$

and there is an event $comEvent_N$ in machine $M_{B'}$:

$$\begin{array}{l} \lceil \text{Event } comEvent_N \hat{=} \quad \rceil \\ \text{WHEN } G'_N \quad \in E_{M_{B'}}, \\ \text{THEN } R_N := \top \end{array} \quad (14)$$

where $G' \equiv GUARD_N(G) \upharpoonright_{ExtVAR(M_{B'})} \equiv \text{toB}(G) \upharpoonright_{ExtVAR(M_{B'})}$.

Now, by (11) we know that

$$\mathcal{M}_{PIM}, \sigma \models \text{toB}(G) \quad (15)$$

so it *cannot* be the case that $\mathcal{M}_{PIM}, \sigma \models GUARD_N(G)^*$ holds by (13), because $\mathcal{M}_{PIM}, \sigma \models R_N = \perp$ by (3). However, by (15) it must be the case that $\mathcal{M}_{PIM}, \sigma \models G'_N$ (because $G'_N \equiv GUARD_N(G) \upharpoonright_{ExtVAR(M_{B'})}$, and the restricted form of a conjunctive formula should hold if the original formula holds over a state). Consequently, we know that $\langle S, \sigma \rangle \xrightarrow{M_{B'}, comEvent_N} \langle S, \sigma_1 \rangle$, where $\sigma_1 = [R_N := \top]\sigma$. So

$$\mathcal{M}_{PIM}, \sigma_1 \models R_N = \top \quad (16)$$

Furthermore, because all other events are generated from norms that exclude each other's guards, σ_1 is the *only* such state that can follow from σ . By (15) and the fact that G (and so toB) do not contain any reference to R_N , we have that

$$\mathcal{M}_{PIM}, \sigma_1 \models \text{toB}(G) \quad (17)$$

But then it must be the case that

$$\mathcal{M}_{PIM}, \sigma_1 \models GUARD_N(G)^* \quad (18)$$

by (13), (17) and (16) because $GUARD_N(G)^* \equiv \text{toB}(G)/G' \wedge R_N = \top$. Finally, by definition of the executable semantics, we know that there is a state σ' such that $\langle S, \sigma_1 \rangle \xrightarrow{M_B, event_N} \langle S, \sigma' \rangle$, where $\sigma' = [\phi(D); R_N := \perp]$ and so $\mathcal{M}_{PIM}, \sigma' \models \text{toB}(DEF)$. Furthermore, because all other events are generated from norms that exclude each other's guards, σ' is the *only* such state that can follow from σ_1 , as required.

- The proof is very similar for the case where $R(p, b', b)$, where $p : P\{\dots\}$, $P\{\dots\} \in ENTITY$, $b : B\{\dots\}$, $B\{\dots\} \in AGENT$, $b' : B'\{\dots\}$, $B'\{\dots\} \in AGENT$ and $R \in COMMACT$ and where the two agent types are *the same*, so $B\{\dots\} = B'\{\dots\}$. The main difference is that the transitions are occurring within the same machine, rather than out of it – however this does not effect the argument over the transition semantics, which is essentially the same as the dual machine case above.

Now, in either case, we have that $\mathcal{M}_{PIM}, \sigma' \models \text{toB}(DEF)$ and so (by a straightforward induction over DEF using the definition of toB) it can be seen

$$\mathcal{M}_{CIM} \models_{\sigma'} DEF \quad (19)$$

Now, by the definition of \leftrightarrow and the assumptions regarding the form of h_{ACT} , we know that $\mathcal{M}_{CIM}, \sigma'' \models A \leftrightarrow DEF$ for any σ'' . So it must be the case that $\mathcal{M}_{CIM}, \sigma' \models A$. Because all other events are generated from norms that exclude each other's guards, σ' is the *only* such state that can follow from σ_1 , which in turn is the only state that can follow from σ , which means

$$\mathcal{M}_{CIM} \models G \rightarrow E_b \mathbf{M} A$$

as required. \square

5. Conclusion and Future Work

In this paper, we have provided a semantic preservation proof of our MDA transformation, which was described in [3]. The transformation uses CIM as a source model and PIM as a target model. The CIM is represented as normative ontologies and norms, i.e. MEASUR combined with the action logic and deontic logic in the light of the theory of normative positions. PIM is formal Event-B machines. The MDA transformation model addresses the semantic gap between these two abstract artefacts.

Norms are used for regulating and constraining behavioral patterns in certain organized environments. For example, in the area of artificial intelligence and multi-agent systems [9], agents need to organize their action patterns in a way to avoid conflicts, address complexity, reach agreements, and achieve a social order. These patterns are specified by norms which constrain what may, must and must not be done by an agent or a set of agents. The fulfillment of certain tasks by agents can be seen as a public good if the benefits that they bring can be enjoyed by the society. [9], [10]

There are several works in the scope of MDA devoted to CIM-to-PIM transformation with semantic preservation. For example, Rodriguez, et al. [11] define CIM-to-PIM transformation using QVT mapping rules. The model of the information system that they obtain as a PIM are represented as certain UML analysis-level classes and the whole idea is reflected in a case study related to payment for electrical energy consumption. This work was continued [12] by extending CIM definitions of BPMN to define security requirements and transforming them into UML use cases using QVT mapping rules. Another approach described by Wil van der Aalst, et al. [17] meets the difficulty of BPEL. While being a powerful language, BPEL is difficult for end-users to use. Its XML representation is very verbose and only readable for the trained eye [17]. It describes implementation of the transformation from Workflow-Nets to BPEL which is built on the rich theory of Petri nets and can also be applied for other languages.

The Event-B language was successfully applied in several serious projects where there was a need for rigorous and precise specification of the system. For example, Rezazadeh, et al. [18] discuss redevelopment of the central control function display and information system (CDIS). CDIS is a computer-based system for controlling important airport and flight data for London terminal control center. The system was originally developed by Praxis and was operational but yet had several problems related to the questions of formalization. Namely, the problems included difficulty of comprehending the specifications, lack of mechanical proof of the consistency and difficulties in distribution and refinement. These problems were addressed in redeveloping the system using the advantages of the Event-B language and Rodin platform.

Future work will investigate how our B-based PIMs can be further transformed into an actual platform specific solution utilizing industrial BPM solutions. We hope that our specifications involving data and operations making them semantically richer will map naturally onto the modular technologies employed in, for example, WWF. Another plan includes incorporation of the SOA concept into the final implementation. In particular, we will look into the possibility of using web services in specifying business processes thereby making them loosely coupled and adaptable to performing different tasks.

References

- [1] L. Kecheng, *Semiotics in Information Systems Engineering* Cambridge University Press, 2000.
- [2] T. Umarov, *Analytical Business Computing: Bridging the Semantic Gap between Requirements and Design Using Model Driven Architecture*, Ph.D. thesis King's College London, London, UK, 2009
- [3] I. Poernomo, T. Umarov "A Mapping from Normative Requirements to Event-B to Facilitate Verified Data-Centric Business Process Management," in *Advances in Software Engineering Techniques*, Eds. Tomasz Szmuc and Marcin Szyrka and Jaroslav Zendulka 2009, Springer, pp. 136-149.
- [4] M. Hepp and D. Roman, "An Ontology Framework for Semantic Business Process Management," in *Proceedings of the 8th International Conference Wirtschaftsinformatik*, 2007, Universitaetsverlag Karlsruhe.
- [5] I. Poernomo, "Proofs-as-Model-Transformations," in *Proc. Theory and Practice of Model Transformations, First International Conference, ICMT'08*, 2008, Springer, pp. 214-228.
- [6] I. Poernomo, T. Umarov, "Business Process Development in Semantically-Enriched Environment," in *Proc. The Fifth International Conference on Information Technology: New Generations ITNG'08*, 2008, IEEE Computer Society, pp. 57-62.
- [7] I. Poernomo, T. Umarov, "Normative Ontologies for Data-Centric Business Process Management," in *Proc. Enterprise of Distributed Object Computing Workshop EDOCW'08*, 2008, pp. 23-34.
- [8] S. Kherraf, É. Lefebvre, W. Suryn, "Transformation from CIM to PIM Using Patterns and Archetypes," in *Proc. of the 19th Australian Conference on Software Engineering*, 2008, IEEE Computer Society, pp. 338-346.
- [9] M. d'Inverno, M. Luck, *Understanding Agent Systems*, Springer Series on Agent Technology Springer, 2004.
- [10] C. Castelfranchi, R. Conte, Paolucci, "Normative reputation and the costs of compliance," in *Journal of Artificial Societies and Social Simulation*, 1998.
- [11] A. Rodríguez, E. Fernández-Medina, M. Piattini, "CIM to PIM Transformation: A Reality," in *Proc. International Conference on Research and Practical Issues of Enterprise Information Systems (2)*, 2007, pp. 1239-1249.
- [12] A. Rodríguez, E. Fernández-Medina, M. Piattini, "Towards CIM to PIM Transformation: From Secure Business Processes Defined in BPMN to Use-Cases," in *Business Process Management*, 2007, pp. 408-415.
- [13] W. Zhang, H. Mei, H. Zhao, J. Yang, "Transformation from CIM to PIM: A Feature-Oriented Component-Based Approach," in *Proc. Model Driven Engineering Languages and Systems*, 2005, Springer Berlin Heidelberg, pp. 248-263.
- [14] L. Baresi, K. Ehrig, R. Heckel, "Verification of model transformations: a case study with BPEL," in *Proc. of the 2nd international conference on Trustworthy global computing TGC'06*, 2007, Springer-Verlag, pp. 183-199.
- [15] P. Barbosa, F. Ramalho, J. de Figueiredo, A. Junior, "An Extended MDA Architecture for Ensuring Semantics-Preserving Transformations," in *Software Engineering Workshop, Annual IEEE/NASA Goddard*, 2008, IEEE Computer Society, pp. 33-42.
- [16] P. Barbosa, F. Ramalho, J. de Figueiredo, A. Junior, A. Costa, L. Gomes, "Checking Semantics Equivalence of MDA Transformations in Concurrent Systems," in *Proc. JUCS'09*, 2009, pp. 2196-2224.
- [17] W. van der Aalst, K. Lassen, "Translating Workflow Nets to BPEL," in *BETA Working Paper Series*, 2005, Eindhoven University of Technology, Eindhoven, series 145.
- [18] A. Rezazadeh, N. Evans, M. Butler, "Redevelopment of an Industrial Case Study Using Event-B and Rodin," in *Formal Aspects of Computing Science - Formal Methods In Industry*, 2007, The British Computer Society, pp. 1-8.
- [19] J.-R. Abrial, C. Métayer, L. Voisin, "Event-B Language," in *RODIN Deliverable 3.2*, 2005, version 1.1.

SESSION
WEB APPLICATIONS AND CASE STUDIES

Chair(s)

TBA

An approach for testing passively Web service compositions in Clouds

Sébastien Salva

LIMOS CNRS UMR 6158, PRES Clermont University,
Aubière, FRANCE

Abstract—*This paper proposes a formal passive testing approach for Web service compositions deployed in Clouds. It addresses an issue lifted by the PaaS layer of Clouds, which represents virtualised environments where compositions can be deployed. In these latter, sniffer-based modules, which are the heart of passive testing methods, cannot be installed for technical reasons. We propose a new approach based on the notion of transparent proxy. We define a new model, called proxy-tester, which is used to observe the functional behaviours of the composition under test but also to check if it is ioco-conforming to its specification. We also provide a solution and algorithms for testing several composition instances in parallel.*

Keywords: Web service composition; ioSTS; passive testing; proxy tester.

1. Introduction

Conformance testing is now a well-established activity, in model-based development, which aims at trying to find defects in systems or software by analysing the observed functional behaviours of an implementation and by comparing them with those of its specification. Two kind of approaches may be considered for testing. On the one hand, active methods can be applied to experiment an implementation with a set of predefined test cases, constructed from the specification, and to conclude whether a test relation is satisfied. For instance, *ioco* [1] is a standard conformance test relation which expresses the set of correct implementations by means of suspension traces (sequences of actions and quiescence).

On the other hand, passive testing, which is the topic of the paper, is another approach which does not actively experiment the implementation under test, but which passively observes its reactions over a long period of time without the need of a pervasive test environment. This approach offers definite advantages in comparison to active methods e.g., to publish more rapidly a system or to not inadvertently disturb it while testing. The passive tester is composed of a kind of sniffer-based module which is supposed to observe both the stimuli sent to the implementation and its reactions in the environment where it is running. Then, the resulting traces are used to check the satisfiability of a test relation or of properties called invariants [2]. In literature, all the proposed

works, dealing with passive testing, rely on a sniffer-based module to extract traces [2], [3], [4]. This module must be installed in the implementation environment and may require modifications of the latter. However, at the same time, some current trends in Computer Science, such as Cloud computing, propose to replace physical environments, such as servers, by virtualised environments composed of resources whose locations and details are not known. In these environments, a sniffer-based analyser cannot be installed, consequently the current passive methods cannot be applied.

This paper addresses this issue by focusing on the passive testing of Web service compositions deployed on Clouds, and more precisely on *PaaS* (Platform as a Service) which is the layer that supports service deployment [5]. In the context of *PaaS* environments, we propose a model-based passive testing method which relies on the notion of transparent proxy to observe traces. With this concept, no code modification is required, it is only necessary to configure Web services and clients to pass through a proxy. For Web service compositions described with ioSTSs (input/output Symbolic Transition Systems [6]), we firstly define another model called *proxy-tester*. Intuitively, this model describes the functioning of a dedicated kind of proxy of one specification: it represents an intermediary between the different partners taking part to the composition (Web services and clients). So, once executed with an algorithm provided in the paper, it helps to follow the functioning of one Web service composition instance and to forward any received message to the right partner. Thanks to this model, we also show that composition traces can be extracted while testing and that these traces can be also used to check whether the composition under test is ioco-conforming to its specification.

This paper focuses on another issue briefly mentioned previously. Web service compositions deployed in *PaaS* can be invoked by several clients at the same time. Consequently, several composition instances can be executed concurrently. We consider these composition instances in the paper and provide original algorithms to construct traces in parallel.

This paper is structured as follows: Section 2 defines the Web service composition modelling. Section 3 describes our passive testing method by defining the ioco proxy-tester of one specification. In Section 4, we details the passive tester functioning, which is suitable to test several instances of

the same composition in parallel. Finally, we conclude in Section 5.

2. Model Definition and notations

To express formally the functional behaviours of Web service compositions, we focus on models called input/output Symbolic Transition Systems (ioSTS). An ioSTS is a kind of automata model which is extended with a set of variables and with transition guards and assignments, giving the possibilities to model the system state and constraints on actions. With ioSTSs, action set is separated with inputs beginning by ? to express the actions expected by the system, and with outputs beginning by ! to express actions produced (observed) by the system.

Below, we give the definition of an ioSTS extension, called ioSTS suspension which also expresses quiescence i.e., the absence of observation from a location. Quiescence is modelled by a new symbol ! δ and an augmented ioSTS denoted $\Delta(\text{ioSTS})$. For an ioSTS \mathcal{S} , $\Delta(\mathcal{S})$ is obtained by adding a self-loop labelled by ! δ for each location where quiescence may be observed. The guard of this new transition must return true for each value which does not allow firing a transition labelled by an output.

Definition 1 (ioSTS suspension) An ioSTS suspension is a tuple $\langle L, l_0, V, V_0, I, \Lambda, \rightarrow \rangle$, where:

- L is the finite set of locations, with l_0 the initial one,
- V is the finite set of internal variables, while I is the finite set of interaction ones. We denote D_v the domain in which a variable v takes values. The internal variables are initialised with the assignment V_0 on V , which is assumed to be unique,
- Λ is the finite set of symbols, partitioned by $\Lambda = \Lambda^I \cup \Lambda^O \cup \{\delta\}$: Λ^I represents the set of input symbols, (Λ^O) the set of output symbols,
- \rightarrow is the (potentially non deterministic) finite transition set. A transition $(l_i, l_j, a(p), G, A)$, from the location $l_i \in L$ to $l_j \in L$, denoted $l_i \xrightarrow{a(p), G, A} l_j$ is labelled by an action $a(p) \in \Lambda \times \mathcal{P}(I)$, with $a \in \Lambda$ and $p \subseteq I$ a finite set of interaction variables $p = (p_1, \dots, p_k)$. G is a guard over $(p \cup V \cup T(p \cup V))$ which restricts the firing of the transition. $T(p \cup V)$ are boolean terms over $p \cup V$. Internal variables are updated with the assignment function A of the form $(x := A_x)_{x \in V}$. A_x is an expression over $V \cup p \cup T(p \cup V)$.

Web service compositions exhibit special properties relative to the service-oriented architecture (operations, partners, etc.). This is why we adapt (restrict) the ioSTS action modelling:

Messages model: to represent the communication behaviours of Web service compositions with ioSTSs, we firstly assume that an action $a(p)$ expresses a message i.e., the call of a Web service operation op ($a(p) = \text{opReq}(p)$), or

the receipt of an operation response ($a(p) = \text{opResp}(p)$), or quiescence. The set of parameters p must gather also some specific variables:

- the variable *from* is equal to the calling partner and the variable *to* is equal to the called partner,
- Web services may engage in several concurrent interactions by means of several stateful instances called sessions, each one having its own state. For delivering incoming messages to the correct running session when several sessions are running concurrently, the usual technical solution is to add, in messages, correlation values which match a part of the session state [7], [8]. So when a session calls another partner, the message must be composed of a set of values called *correlation set* which identifies the session. We model a correlation set in a message $a(p)$ with a parameter, denoted $\text{cor} \in p$.

The use of correlation sets with ioSTSs also implies to set the following hypotheses on actions:

Session identification: the specification is well-defined. When a message is received, it always correlates with at most one session.

Message correlation: except for the first operation call which starts a new composition instance, a message $\text{opReq}(p)$, expressing an operation call, must contain a correlation set $\text{cor} \subseteq p$ such that a subset $c \subseteq \text{cor}$ of the correlation set is composed of parameter values given in previous messages.

The first hypothesis results from the correlation sets functioning. The last one is given to coordinate the successive operation calls together so that we could follow the functioning of one composition instance without ambiguity by observing the messages and the correlation sets exchanged between sessions, while testing.

These notation are expressed in the straightforward example of Figures 1 and 2. This specification describes a composition of two Web services: *Shoppingservice* represents an interface which allows a customer to look for the availability of books with isbn (International Standard Book Number). This service calls *StockService* to check the availability and the current stock of one book.

For one composition instance, we have two sessions of services connected together with correlations sets. Each service session is identified with its own correlation set e.g., *Shoppingservice* with $c1 = \{\text{account} = \text{"custid"}\}$, *StockService* with $c2 = \{\text{account} = \text{"custid"}, \text{isbn} = \text{"2070541274"}\}$. As these two correlation sets respect the *Message correlation* assumption, we can correlate the call of *StockService* with one previous call of *Shoppingservice* even though several sessions are running in parallel.

An ioSTS is also associated to an ioLTS (Input/Output Labelled Transition System) to formulate its semantics. Intuitively, the ioLTS semantics corresponds to a valued automaton without symbolic variable, which is often infinite:

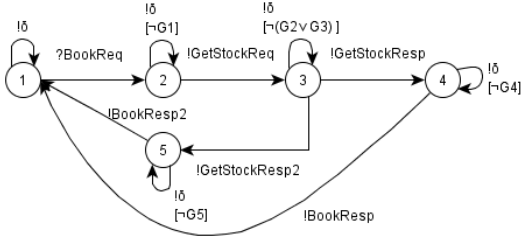


Fig. 1: A suspension ioSTS

Symbol	Message	Guard	Update
?BookReq	?BookReq(account, isbn, from, to, coor)	[from="Client" ∧ to="S" ∧ corr={account}]	{a:=account, c1=coor}
!GetStockReq	!GetStockReq(isbn, from, to, coor)	G1 = [from="S" ∧ to="Stock" ∧ coor={a, isbn}]	{i:=isbn, c2=coor}
!GetStockResp	!GetStockResp(stock, from, to, coor)	G2=[from="Stock" ∧ to="S" ∧ valid(i) ∧ coor=c2]	st:=stock
!GetStockResp2	!GetStockResp(resp, from, to, coor)	G3=[from="Stock" ∧ to="S" ∧ ¬valid(i) ∧ resp="invalid" ∧ coor=c2]	st:=null
!BookResp	!BookResp(resp, from, to, coor)	G4=[from="S" ∧ to="Client" ∧ resp="Book available" ∧ corr=c1]	
!BookResp2	!BookResp(resp, from, to, coor)	G5=[from="S" ∧ to="Client" ∧ resp="Book unavailable" ∧ coor=c1]	

Fig. 2: Symbol table

the ioLTS states are labelled by internal variable values while transitions are labelled by actions and interaction variable (parameter) values. The semantics of an ioSTS $\mathcal{S} = \langle L, l_0, V, V_0, I, \Lambda, \rightarrow \rangle$ is an ioLTS $\|\mathcal{S}\| = \langle Q, q_0, \Sigma, \rightarrow \rangle$ composed valued states in $Q = L \times D_V$, $q_0 = (l_0, V_0)$ is the initial one, Σ is the set of valued symbols and \rightarrow is the transition relation. The complete definition of ioLTS semantics can be found in [6].

Intuitively, for an ioSTS transition $l_1 \xrightarrow{a(p), G, A} l_2$, we obtain an ioLTS transition $(l_1, v) \xrightarrow{a(p), \theta} (l_2, v')$ with v a set of values over the internal variable set, if there exists a parameter value set θ such that the guard G evaluates to true with $v \cup \theta$. Once the transition is executed, the internal variables are assigned with v' derived from the assignment $A(v \cup \theta)$. Finally, runs and traces of ioSTS can be defined from their semantics:

Definition 2 (Runs and traces) For an ioSTS \mathcal{S} , interpreted by its ioLTS semantics $\|\mathcal{S}\| = \langle Q, q_0, \Sigma, \rightarrow \rangle$, a run $q_0 \alpha_0 \dots \alpha_{n-1} q_n$ is an alternate sequence of states and valued actions. $RUN_F(\mathcal{S}) = RUN_F(\|\mathcal{S}\|)$ is the set of runs of \mathcal{S} finished by a state in $F \times D_V \subseteq Q$ with F a location of \mathcal{S} .

It follows that a trace of a run r is defined as the projection $proj_{\Sigma}(r)$ on actions. $Traces_F(\mathcal{S}) = Traces_F(\|\mathcal{S}\|)$ is the set of traces of runs finished by states in $F \times D_V$.

3. Ioco passive testing with proxy-testers

This Section covers the theoretical aspects of ioco proxy-testing. Instead of using a classical proxy for observing messages, we formalise below the notion of proxy-tester of an ioSTS specification. This model will help to observe traces but will be also used to directly check whether a composition implementation is ioco-conforming to its specification.

3.1 Proxy-tester definition

A proxy-tester corresponds to a passive intermediary between the partners of one composition (Web services and clients) which must observe any behavioural action (messages or quiescence). To act as an intermediary between partners, a proxy-tester must exhibit different behaviours: to receive client requests or to forward received messages to the composition, it must behave as in the specification. It must also collect, by means of input actions, the observable messages (output actions) produced by any partner. This can be expressed with a mirror specification (inputs are replaced with outputs and vice-versa). While receiving actions, we propose that it also recognises the correct messages and the incorrect ones to detect failures. As a consequence, a proxy-tester must be a combination of the specification with a mirror specification augmented with the potential incorrect behaviours of the composition. This second part corresponds to a non-conformance observer of the specification, also called canonical tester [9].

The non-conformance observer of an ioSTS, denoted NCO_{observer} , gathers the specification transitions labelled by mirrored actions (inputs become outputs and vice versa) and transitions leading to a new location $Fail$, exhibiting the receipt of unspecified actions. Transitions to a $Fail$ location are guarded by the negation of the union of guards of the same output action in outgoing transitions. Due to its extent and generality, we do not provide here the definition of the NCO_{observer} of an ioSTS which can be found in [10]. Instead, we illustrate the NCO_{observer} of the previous specification in Figures 3 and 4. Inputs are replaced with outputs and vice-versa. Incorrect behaviours are also added with new transitions to $Fail$. For instance, if we consider the location 2, new transitions to $Fail$ are added to model the receipt of unspecified actions (messages or quiescence).

As stated earlier, a proxy-tester corresponds to a kind of fuse of an ioSTS with its NCO_{observer} over the transition set. To express this combination without ambiguity, we initially separate locations of \mathcal{S} and $NCO(\mathcal{S})$ with a renaming function ϕ . Locations are renamed by $\phi : L \rightarrow L'$, $\phi(l) \rightarrow l'$. For an ioSTS \mathcal{S} , we also denote $\phi(\mathcal{S}) = \langle \phi(L_{\mathcal{S}}), \phi(l_0_{\mathcal{S}}), V_{\mathcal{S}}, V_{0_{\mathcal{S}}}, I_{\mathcal{S}}, \Lambda_{\mathcal{S}}, \rightarrow_{\phi(\mathcal{S})} \rangle$.

Now, we are ready to define the proxy-tester of an ioSTS \mathcal{S} :

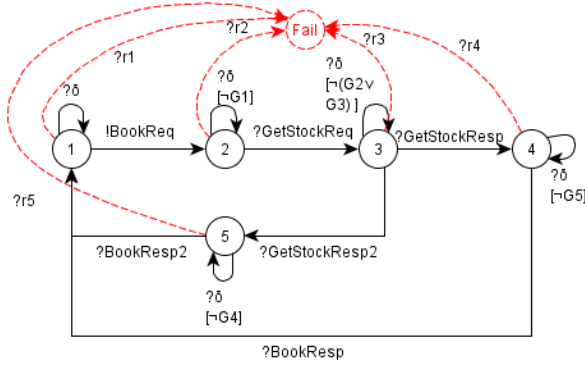


Fig. 3: An ioSTS NCObserver

Symbol	Message
?r1	?CartCreateReq ?CartCreateResp ?Connectresp
?r2	?CartCreateReq[G1] ?CartcreateResp ?ConnectResp ?δ[G1]
?r3	?CartCreateReq ?CartcreateResp [-G2 wedge -G3] ?ConnectResp ?δ [G2 ∨ G3]
?r4	?CartCreateReq ?CartcreateResp ?ConnectResp [-G5] ?δ[G5]
?r5	?CartCreateReq ?CartCreateResp ?ConnectResp[-G4] ?δ[G4]

Fig. 4: Symbol table

Definition 3 (Proxy-tester) The proxy-tester $\mathcal{P}(\mathcal{S})$ of the specification $\mathcal{S} = \langle L_S, l_{0_S}, V_S, V_{0_S}, I_S, \Lambda_S, \rightarrow_S \rangle$ is a combination of $\Delta(\mathcal{S})$ with its NCObserver $\phi(NCO(\mathcal{S}))$. $\mathcal{P}(\mathcal{S})$ is defined by an ioSTS $\langle L_P, l_{0_P}, V_S \cup \{side\}, V_{0_S} \cup \{side := ""\}, I_S, \Lambda_{\Delta(\mathcal{S})} \cup \Lambda_{NCO}, \rightarrow_P \rangle$ such that L_P, l_{0_P} and \rightarrow_P are constructed by the following inference rules.

(Env)	$l_1 \xrightarrow{?a(p),G,A} \Delta(\mathcal{S}) l_2, l_1' \xrightarrow{!a(p),G,A} NCO l_2',$
to	$l_1' = \phi(l_1), l_2' = \phi(l_2)$
IUT):	\vdash
	$(l_1 l_1') \xrightarrow{?a(p),G,A(\{p_t=p, side:=""\})} \mathcal{P} (l_2 l_2', ?a$
	$GA) \xrightarrow{!a(p),[p=p_t],A(\{side:="" NCO''\})} \mathcal{P} (l_2 l_2'),$
(IUT	$l_1 \xrightarrow{!a(p),G,A} \Delta(\mathcal{S}) l_2, l_1' \xrightarrow{?a(p),G,A} REF l_2',$
to	$l_1' = \phi(l_1), l_2' = \phi(l_2)$
Part	\vdash
ner)	$(l_1 l_1') \xrightarrow{?a(p),G,A(\{side:="" NCO'' .p_t=p\})} \mathcal{P} (l_1 l_2,$
	$?aGA) \xrightarrow{!a(p),[p=p_t],A(\{side:=""\})} \mathcal{P} (l_2 l_2'),$
(to	$l_1' \xrightarrow{b(p),G,A} NCO Fail, l_1 \in L_S, l_1' = \phi(l_1)$
Fail):	\vdash
	$(l_1 l_1') \xrightarrow{b(p),G,[A(\{side:="" NCO''\})]} \mathcal{P} Fail$

Intuitively, the first rule combines a specification transition

and a NCObserver one carrying the same mirrored actions and guards to express that if an action is received from the client environment then it is forwarded to the Web service composition. The two transitions are separated by a unique location $(l_2 l_1', ?aGA)$. The second rule (IUT to Partner) similarly combines a specification transition and an NCObserver one labelled by the same mirrored actions to express that if an action is received from the Web service composition then it is forwarded to right partner (Web service or client). Transitions labelled by δ , modelling quiescence, are also combined: so if quiescence is detected from the implementation, quiescence is also observed from the client environment. The last rule (to Fail) completes the resulting ioSTS with the transition leading to Fail of the NCObserver. In each rule, a new internal variable, denoted *side*, is also added to keep track of the transitions provided by the ioSTS NCObserver (with the assignment $side := "NCO"$). This distinction will be useful to define partial traces of proxy-testers.

Figure 5 depicts the resulting proxy-tester obtained from the previous specification (Figure 1) and its NCObserver (Figure 3). For sake of readability, the *side* variable is replaced with solid and dashed transitions: dashed transitions stand for labelled by the assignment $(side := NCO)$. Figure 5 clearly illustrates that the initial specification behaviours are kept and that the incorrect behaviours modelled in the NCObserver are present as well.

In the proxy-tester definition, transitions carrying actions provided by NCObservers are emphasised by means of the variable *side*. Specific properties on runs and traces of the proxy-tester can be deduced from this property. In particular, we can define partial runs and traces over the variable *side*.

Definition 4 (Partial runs and traces) Let $\mathcal{P}(\mathcal{S}) = \langle L_P, l_{0_P}, V_P, V_{0_P}, I_P, \Lambda_P, \rightarrow_P \rangle$ be a proxy-tester and $\|\mathcal{P}(\mathcal{S})\| = P = \langle Q_P, q_{0_P}, \sum_P, \rightarrow_P \rangle$ be its ioLTS semantics. We define $Side : Q_P \rightarrow D_{V_P}$ the mapping which returns the value of the *side* variable of a state in Q_P . $Side_E(Q_P) \subseteq Q_P$ is the set of states $q \in Q_P$ such that $Side(q) = E$.

Let $RUN(\mathcal{P}(\mathcal{S}))$ be the set of runs of $\mathcal{P}(\mathcal{S})$. We denote $RUN^E(\mathcal{P}(\mathcal{S}))$ the set of partial runs derived from the projection $proj_{(Q_P \sum_P Side_E(Q_P))}(RUN(\mathcal{P}(\mathcal{S})))$.

It follows that $Traces^E(\mathcal{P}(\mathcal{S}))$ is the set of partial traces of (partial) runs in $RUN^E(\mathcal{P}(\mathcal{S}))$.

For a proxy-tester $\mathcal{P}(\mathcal{S})$, we can now write $Traces_{Fail}^{NCO}(\mathcal{P}(\mathcal{S}))$ for representing the partial traces leading to Fail derived from the NCObserver part. With these notations, we can deduce an interesting trace property on proxy-testers. We can write that the incorrect behaviours expressed in the NCObserver with $Traces_{Fail}(NCO(\mathcal{S}))$ can be still captured in the proxy-tester with the trace set $Traces_{Fail}^{NCO}(\mathcal{P}(\mathcal{S}))$.

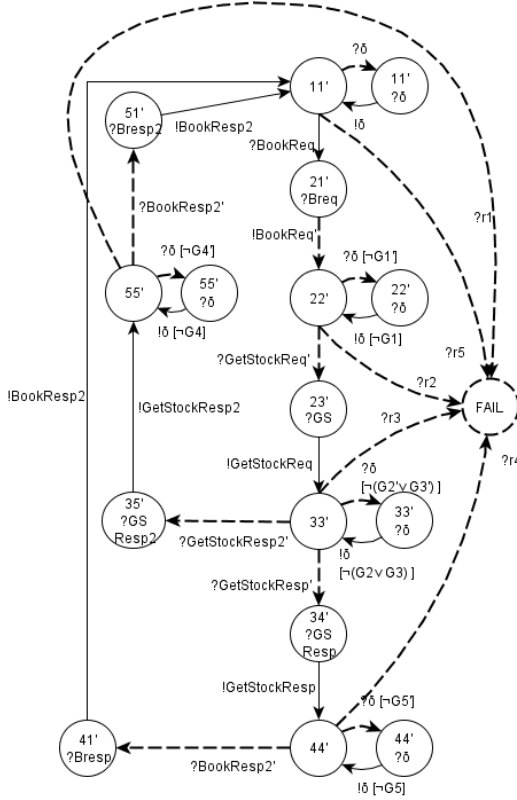


Fig. 5: A proxy-tester

Proposition 5 Let S be a specification and $NCO(S)$, $\mathcal{P}(S)$ be its NCoobserver and its proxy-tester respectively. We have $Traces_{Fail}^{NCO}(\mathcal{P}(S)) = Traces_{Fail}(NCO(S))$.

The proof is given in an extended version of the paper in [11].

3.2 Ioco testing with proxy-testers

Formal testing methods often define the confidence degree between an implementation I and its specification S by means of a test relation. To reason about conformance and to formalise a test relation, the implementation under test is assumed to behave like its specification and is modelled by an ioLTS I . $\Delta(I)$ represents its suspension ioLTS.

In the paper, we consider the *ioco* relation, which is defined as a partial inclusion of suspension traces of the implementation in those of the specification [6]. In [10], *ioco* has been also defined for ioSTS by making explicit the non-conformant trace set:

Definition 6 Let I be an implementation modelled by an ioLTS, and S be an ioSTS. I is *ioco*-conforming to S , denoted $I \text{ ioco } S$ iff $Traces(\Delta(I)) \cap NC_Traces(\Delta(S)) = \emptyset$ with $NC_Traces(\Delta(S)) = Traces(\Delta(S)) \cdot (\Sigma^O \cup \{\delta\} \setminus Traces(\Delta(S)))$.

To check if I is *ioco*-conforming to its specification, we can collect traces of I with proxy-testers and compare them with the specification one. However, since we have formalised proxy-testers, we can also rephrase *ioco* with proxy-tester traces. So, we reformulate *ioco* below.

Firstly, a NCoobserver exhibits the incorrect behaviours of the specification with traces leading to its Fail states. However, $NCO(S)$ is constructed by exchanging inputs and outputs symbols of its specification. If we define $refl : (\Sigma^*)^* \rightarrow (\Sigma^{!*})^*$ the function which constructs a mirrored trace set from an initial one (for each trace, input symbols are exchanged with output ones and vice-versa), we can write:

Proposition 7 Let S be an ioSTS. The non-conformant trace set of $\Delta(S)$, denoted $NC_Traces(\Delta(S))$, is equal to $refl(Traces_{Fail}(NCO(S)))$.

We have also asserted previously that $Traces_{Fail}(NCO(S)) = Traces_{Fail}^{NCO}(\mathcal{P}(S))$ (Proposition 5). Consequently, *ioco* can be formulated with Propositions 5 and 7 as:

$$I \text{ ioco } S \Leftrightarrow Traces(\Delta(I)) \cap refl(Traces_{Fail}^{NCO}(\mathcal{P}(S))) = \emptyset$$

So defined, *ioco* means that I is *ioco*-conforming to its specification when implementation traces do not belong to the set of partial proxy-tester traces leading to Fail, obtained from the NCoobserver part. However, we can go farther, in the *ioco* rephrasing, by taking into account the parallel execution of the client environment, the proxy-tester and the Web service composition implementation. This execution can be defined by a parallel composition:

Definition 8 (Passive test execution) Let $P = \langle Q_P, q0_P, \Sigma_P, \rightarrow_P \rangle$ be the ioLTS semantics of a proxy-tester $\mathcal{P}(S)$, and $I = \langle Q_I, q0_I, \Sigma_I \subseteq \Sigma_P, \rightarrow_I \rangle$ be the implementation model. We assume that the client environment can be modelled with an ioLTS $Env = \langle Q_{Env}, q0_{Env}, \Sigma_{Env} \subseteq \Sigma_P, \rightarrow_{Env} \rangle$.

The passive testing of I is defined by the parallel composition $\|((Env, P, I) = \langle Q_{Env} \times Q_P \times Q_I, q0_{Env} \times q0_P \times q0_I, \Sigma_{Env} \subseteq \Sigma_P, \rightarrow_{\|((Env, P, I))} \rangle$ where the transition relation $\rightarrow_{\|((Env, P, I))}$ is given by the following rules. For readability reason, we denote an ioLTS transition $q_1 \xrightarrow[?a]{?a} q_2$ if $Side(q_2) = E$ (the variable side is valued to E in q_2):

$$\frac{q_1 \xrightarrow[?a]{?a} q_2, q_2 \xrightarrow[?a]{?a} q_3, q_1 \xrightarrow[?a]{?a} q_2 \xrightarrow[?a]{?a} q_3}{q_1 q_1' q_2' \xrightarrow[?a]{?a} q_2 q_2' q_3' \xrightarrow[?a]{?a} q_2 q_2' q_3'}$$

$$\frac{q_2 \xrightarrow[?a]{?a} q_3, q_1 \xrightarrow[?a]{?a} q_2, q_1 \xrightarrow[?a]{?a} q_2 \xrightarrow[?a]{?a} q_3}{q_2 q_1' q_1'' \xrightarrow[?a]{?a} q_2 q_2' q_2'' \xrightarrow[?a]{?a} q_3 q_3' q_3''}$$

$$\frac{q_2 \xrightarrow[?a]{?a} q_3, q_1 \xrightarrow[?a]{?a} q_2, q_1 \xrightarrow[?a]{?a} q_2 \xrightarrow[?a]{?a} q_3}{q_2 q_1' q_1'' \xrightarrow[?a]{?a} q_2 q_2' q_2'' \xrightarrow[?a]{?a} Fail}$$

The immediate deduction of the $\rightarrow_{\parallel(Env, P, I)}$ definition is that $Traces(\Delta(I)) \cap refl(Traces_{Fail}^{NCO}(P)) = Traces(\Delta(I)) \cap refl(Traces_{Fail}^{NCO}(\mathcal{P}(\mathcal{S})))$ is equivalent to $refl(Traces_{Fail}(\parallel(Env, P, I)))$ (third rule). In other terms, the non-conformant traces of $\Delta(I)$ can be also found in $Traces_{Fail}(\parallel(Env, P, I))$. Therefore, *ioco* can be finally also reformulated as:

$$I \text{ ioco } \mathcal{S} \Leftrightarrow Traces(\Delta(I)) \cap refl(Traces_{Fail}^{NCO}(\mathcal{P}(\mathcal{S}))) = \emptyset \\ \Leftrightarrow Traces_{Fail}(\parallel(Env, P, I)) = \emptyset$$

The *refl* function can be removed in the last equivalence since if the set $refl(Traces_{Fail}(\parallel(Env, P, I)))$ is empty, then $Traces_{Fail}(\parallel(Env, P, I))$ is also empty (the function *refl* only yields mirrored trace sets).

4. Passive tester functioning

A straightforward consequence of *ioco* is that non-conformance ($I \not\text{ioco } \mathcal{S}$) is detected when a trace of the parallel composition $\parallel(Env, P, I)$ leads to one of its Fail states. From this assertion, we can also deduce the intuition of the proxy-tester functioning. It can be summarised by these steps: wait for an action (message or quiescence), cover some proxy-tester transitions when an action is received and construct traces, detect non-conformance when one of the proxy-tester Fail states is reached or continue.

Nevertheless, service compositions, deployed in PaaS, can be invoked concurrently by several client applications. This implies that a tester must also cope with several composition instances gathering several sessions in parallel. To extract traces from these composition instances, several proxy-tester instances, running in parallel, are also required. All of these will be managed by a unique entity that we call *passive tester*. Its architecture is given in Figure 6. The passive tester aims to cover, in parallel, the behaviours of several composition instances to collect traces. Incoming messages must be delivered to the correct proxy-tester instance by the passive tester. This step is performed by an *entry point* which routes messages by means of correlation sets.

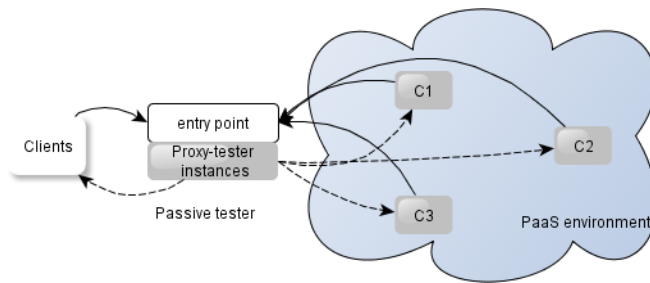


Fig. 6: The passive tester architecture

The entry point functioning is given in Algorithm 1. Each composition instance must be passively tested by a unique proxy-tester instance. Therefore, Algorithm 1 handles a set L of pairs (pi, PV) with pi a proxy-tester instance and PV the set of parameter values received with messages. When

a new message is received, this set is used to correlate it with an existing composition instance in reference to the *Message correlation* hypothesis. As stated in Section 2, the latter helps to correlate messages by assuming that a part of the correlation set of a message is composed of parameter values of messages received previously. Whenever a message $(e(p), \theta)$ is received, its correlation set c is extracted to check if a proxy-tester instance is running to accept it. This instance exists if L contains a pair (pi, PV) such that a subset $c' \subseteq c$ is composed of values of PV . In this case, the correlation set has been constructed from parameter values of messages received previously. If one instance is already running, the message is forwarded to it. Otherwise, (line 7), a new one is started. If a proxy-tester instance pi returns a trace set (line 11), then the latter is stored in $Traces(\mathcal{P}(\mathcal{S}))$ and the corresponding pair (pi, PV) is removed from L .

Algorithm 1: Proxy-tester entry point

```

input : Proxy-tester  $\mathcal{P}(\mathcal{S})$ 
output:  $Traces(\mathcal{P}(\mathcal{S}))$ 

1  $L = \emptyset$ ;
2 while message  $(e(p), \theta)$  do
3   extract the correlation set  $c$  in  $\theta$ ;
4   if  $\exists (pi, PV) \in L$  such that  $c' \subseteq c$  and  $c' \subseteq PV$  then
5     forward  $(e(p), \theta)$  to  $pi$ ;  $PV = PV \cup \theta$ ;
6   else
7     create a new proxy-tester instance  $pi$ ;
8      $L = L \cup (pi, \{\theta\})$ ; forward  $(e(p), \theta)$  to  $pi$ ;
9   if  $\exists (pi, PV) \in L$  such that  $pi$  has returned the trace set  $T$  then
10     $Traces(\mathcal{P}(\mathcal{S})) = Traces(\mathcal{P}(\mathcal{S})) \cup T$ ;
11     $L = L \setminus \{(pi, PV)\}$ ;

```

The proxy-tester algorithm, which aims to test passively one composition instance, is given in Algorithm 2. Both the client environment and the implementation are assumed to behave as ioLTS suspensions. The proxy-tester algorithm handles a set of runs denoted $RUNS$. A single run is not sufficient since both the proxy-tester and the implementation may be indeterministic and may cover different behaviours. The proxy-tester algorithm is based on a forward checking approach. It starts from its initial state i.e., $(I0_{\mathcal{P}(\mathcal{S})}, V0_{\mathcal{P}(\mathcal{S})})$. Upon a received action $(e(p), \theta)$ which is either an valued action or quiescence (line 2), it looks for the next transitions which can be fired for each run r in $RUNS$ (line 5). Each transition must have the same start location as the one found in the final state (l, v) of r , the same action as the received action $e(p)$ and its guard must evaluate to true over the current internal variable value set v and the parameter values θ . If this transition leads to a Fail state then the proxy-tester algorithm adds the resulting run r' to $RUNS$ (lines 8-11). Otherwise, the valued action $(e(p), \theta)$ is forwarded to the called partner with the next proxy-tester transition t_2 (lines 12 to 17). The new run r'' is composed of r' followed by

the sent action and the reached state $q_{next2} = (l_{next2}, v'')$. Once, each run of $RUNS$ is covered, the proxy-tester waits for the next action.

Algorithm 2: Proxy-tester algorithm

```

input : A proxy-tester  $\mathcal{P}(S)$ 
output: Trace set

1  $RUNS := \{(q_0 = (l_{0\mathcal{P}(S)}, V_{0\mathcal{P}(S)}))\}$ ;
2 while  $Action(e(p), \theta)$  do
3    $r' = \emptyset$ ;
4   foreach  $r = q_0 \alpha_0 \dots q_i \in RUNS$  with  $q_i = (l, v)$  do
5     foreach  $t = l \xrightarrow{e(p), G, A} l_{next} \in \rightarrow_{\mathcal{P}(S)}$  such that  $G$ 
        evaluates to true over  $\theta \cup v$  do
6        $q_{next} = (l_{next}, v' = A(v \cup \theta))$ ;
7        $r' = r.(e(p), \theta).q_{next}$ ;
8       if  $l_{next} == Fail$  then
9         Fail is detected;
10         $RUNS = (RUNS \setminus r) \cup r'$ ;
11      else
12        Execute( $t_2 = l_{next} \xrightarrow{!e(p), G_2, A_2} \rightarrow_{\mathcal{P}(S)} l_{next2}$ )
          ; // forward  $(!e(p), \theta)$  to Partner
13         $q_{next2} := (l_{next2}, A_2(\theta \cup v'))$ ;
14         $r'' = r'.(!e(p), \theta).q_{next2}$ ;
15         $RUNS = (RUNS \setminus r) \cup r''$ ;
16 return the trace set  $T = proj_{\Sigma}(RUNS)$ ;

```

Algorithm 2 reflects exactly the parallel execution definition of Section 3.1. It actually constructs traces of $\|(Env, P, I)$ by supposing that Env and I are iOLTS suspensions. In particular, when a location Fail is reached (line 8), the proxy-tester has constructed a run, from its initial state which belongs to $RUN_{Fail}(\|(Env, P, I))$. From this run, we obtain a trace of $Traces_{Fail}(\|(Env, P, I))$. So, we can state the correctness of the algorithm with:

Proposition 9 *The algorithm has detected Fail*
 $\Rightarrow Traces_{Fail}(\|(Env, P, I)) \neq \emptyset \Rightarrow \neg(I \text{ ioco } S)$.

These algorithms are currently under development for two well-known Paas, Windows Azure and Google AppEngine. With Windows Azure, the entry point is implemented as a transparent proxy which behaves as it is described in Algorithm 1. No modification of composition codes is required. Web services and clients need to be configured to pass through an external proxy only. In summary, this development part is not raising particular issues. It is quite different with Google AppEngine since the use of proxy is prohibited. In this PaaS, we intend to implement the entry point of the passive tester as a Java Filter Servlet. This kind of application can filter the messages exchanged between Web applications and clients. All these implementations and experimentation will be proposed in future works.

5. Conclusion

We have proposed an original approach for passive testing Web service compositions in PaaS environments. Our approach is based upon the notion of transparent proxy and is able to construct implementations traces from several composition instances deployed in virtualised environments, without requiring any modification of code. This approach also offers the advantage of checking the satisfiability of the ioco relation. An immediate line of future work is to implement the proposed passive tester and to experiment existing compositions for different Clouds, each having its own possibilities and restrictions.

References

- [1] J. Tretmans, "Test generation with inputs, outputs and repetitive quiescence," *Software - Concepts and Tools*, vol. 17, no. 3, pp. 103–120, 1996.
- [2] A. Cavalli, S. Maag, and E. M. de Oca, "A passive conformance testing approach for a manet routing protocol," in *Proceedings of the 2009 ACM symposium on Applied Computing*, ser. SAC '09. New York, NY, USA: ACM, 2009, pp. 207–211. [Online]. Available: <http://doi.acm.org/10.1145/1529282.1529326>
- [3] T. Lin, S. Midkiff, and J. Park, "A framework for wireless ad hoc routing protocols," in *Wireless Communications and Networking, 2003. WCNC 2003, New Orleans, LA, USA*. IEEE society press, 2003.
- [4] D. Lee, D. Chen, R. Hao, R. E. Miller, J. Wu, and X. Yin, "Network protocol system monitoring: a formal approach with passive testing," *IEEE/ACM Trans. Netw.*, vol. 14, pp. 424–437, April 2006.
- [5] A. Lenk, M. Klems, J. Nimis, S. Tai, and T. Sandholm, "What's inside the cloud? an architectural map of the cloud landscape," in *the 2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing*, 2009, pp. 23 – 31.
- [6] L. Frantzen, J. Tretmans, and T. Willemse, "Test Generation Based on Symbolic Specifications," in *Formal Approaches to Software Testing – FATES 2004*, ser. Lecture Notes in Computer Science, J. Grabowski and B. Nielsen, Eds., no. 3395. Springer, 2005, pp. 1–15.
- [7] Oasis-consortium, "Ws-bpel," 2007. [Online]. Available: <http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html>
- [8] F. Montesi and M. Carbone, "Programming services with correlation sets," in *ICSOC*, ser. Lecture Notes in Computer Science, G. Kappel, Z. Maamar, and H. R. Motahari-Nezhad, Eds., vol. 7084. Springer, 2011, pp. 125–141.
- [9] B. Jeannot, T. Jéron, and V. Rusu, "Model-Based Test Selection for Infinite-State Reactive Systems," in *Formal Methods for Components and Objects*, Amsterdam, Netherlands, 2006. [Online]. Available: <http://hal.inria.fr/inria-00564604>
- [10] V. Rusu, H. Marchand, and T. Jéron, "Automatic verification and conformance testing for validating safety properties of reactive systems," in *Formal Methods 2005 (FM05)*, ser. LNCS, J. Fitzgerald, A. Tarlecki, and I. Hayes, Eds. Springer, July 2005.
- [11] S. Salva, "An approach for testing passively web service compositions in clouds," 2011, IIMOS Research report RR-12-03. [Online]. Available: <http://sebastien.salva.free.fr/RR-12-03.pdf>

Design of Safety Critical Survivable Systems Using Autonomic and Semantic Web Methodologies

Michael C. Lindsey¹, David J. Coe¹, Jeffrey H. Kulick¹, Letha Etzkorn², Wei Li², and Yujian Fu³

¹Dept. of Electrical and Computer Engineering, University of Alabama in Huntsville, Huntsville, Alabama, USA

²Dept. of Computer Science, University of Alabama in Huntsville, Huntsville, Alabama, USA

³Dept. of Electrical Engineering and Computer Science, Alabama A&M University, Huntsville, Alabama USA

Abstract - *Safety critical software systems are designed to be able to respond to failures and continue to operate correctly. Systems designed for long duration missions such as space flights need to be able to reconfigure themselves after failures and continue to be demonstrably safe. Autonomic self-optimization and self-healing principles provide the basic ideas for a system capable of recovering from a partial failure event. The first part of the paper discusses some approaches that have been taken to detection and repair of autonomic systems including incremental, formal controller verification allow for the creation of an on-the-fly safety verification analysis. The second part discusses a framework, based on the semantic web, for implementing such as system.*

Keywords: compositional verification; autonomic; self diagnosis and healing; state space reduction; adaptive safety critical systems

1 Introduction

As technology advances safety critical computing systems are becoming more complex. While this complexity adds features it also causes systems to be difficult to manage and optimize. Furthermore, these systems are being designed to be adaptable and to reconfigure after a failure event. Such systems must automatically manage themselves because human operators are not always present such as in long duration space flights. Also, adaptable safety critical systems must be verified in order to ensure safe operation. The verification process should be performed quickly as to not further endanger the system.

In 2001 IBM presented the Autonomic Computing paradigm, which described a means to design systems, that self-heal, self-optimize, self-configure, and self-protect. Environmental influence, workload changes, and internal failures would all be managed and accounted for [1]. While few systems have been created that fully meet this lofty goal, some of the constructs popularized in the paradigm are useful in creating safe, adaptive systems.

Safety critical systems must be verifiable as well as highly reliable. Traditionally the verification process has

been carried out offline before the system is fielded. The traditional approach has required the production of hazard analyses, fault trees, and failure mode and effects analyses [18]. Formal methods are also used to verify the system's functionality. If the system is re-configured after failure these calculations become invalid. If a complete re-verification is attempted online in a complex system the calculations become overwhelming. This is commonly referred to as state space explosion [16]. In this situation, the reconfiguration and re-verification processes cannot take place in a timely manner and the system is ultimately unusable. To combat this problem, researchers are developing compositional verification methods that begin with a verified system and during reconfiguration only analyze aspects of the state space that have been modified or pertain to the controller's next step.

In the first two sections that follow, fault detection and fault repair are discussed. In the final section, a framework based on the semantic web for organizing the necessary programs, data, and proofs will be presented.

2 Detecting and handling failures

2.1 Detecting a failure event

The first step in responding to a system failure is detection. Many systems must change operation when their environment is altered, but more often a system must change to cope with a fault. In [8] and [9] Byttner et al present a method for discovering faults that are not predefined. In a typical safety analysis for an airframe, for example, an all-encompassing fault model is constructed before fielding the system. In Byttner's work a behavior signature is constructed from a fleet of operational systems. Numerous inexpensive sensors are placed on the system's most important components and linear principle encoding analysis is used to reduce the sensor information before transmitting it to a central location. Once collected the data for all systems are analyzed and a multivariate unimodal Gaussian distribution is used to determine which systems are faulty by comparing the current behavior to previously known good behaviors. This approach allows for the detection of unexpected faults that likely would have been

missed in a static comparison model.

Consequence-oriented self-healing presented in [7] is an attempt to diagnose a condition based on symptoms in the system and take corrective action before the fault occurs. A static multivariate decision diagram is used to determine the severity of an impending fault in order to assign a priority value to it. A hybrid fuzzy logic and neural network diagnosis tool is then used to determine what preemptive actions should be taken to avoid the fault. While the neural network has offline batch training before the system is fielded, it also incorporates online learning in order to better diagnose a changing system. An adaptive diagnosis tool such as either of the aforementioned is necessary in a safety critical system that attempts to detect unanticipated failure events.

2.2 Responding to a failure event

Once a failure event has been detected an adaptive system must take action to reconfigure itself, continuing to meet its goals regardless of any failed components. Many methods have been developed to achieve system healing. This section describes some of these methods including planned reconfiguration, genetic algorithms, and learning classifier systems. Additionally, the challenges and merits of each approach are reviewed in regards to online safety critical reconfiguration.

2.2.1 Planned reconfiguration systems

Planned reconfiguration systems are designed with duplicate components and a rigid protocol for adaptation. In simple systems, when a primary component fails the secondary is dynamically switched into the control flow to take the primary's place.

In [5] Khalgui introduces an agent-based system for safe, dynamic reconfiguration. Each device in the system has an associated agent responsible for monitoring it. When the agent detects a failure it requests a reconfiguration from the central coordination agent. If the restructuring is possible, the coordination agent permits the change and notifies other related agents in the system. Organized change in this manner ensures all devices are operating in a harmonious manner. For example, both duplicate components know they are processing in parallel or in a hierarchical manner. Furthermore, the coordinating agent contains a set of coordination matrices, each defining a valid system configuration. By commanding each device agent to its defined state in a coordination matrix the entire system can be put into a known good state. Since each coordinate matrix has been validated offline by a model checker, all the possible system configurations are known to be acceptable.

The major disadvantage of this reconfiguration approach is that all possible system states must be predetermined offline. Any unforeseen failure or environmental conditions will cause the system to stop functioning or perpetually remain in a request condition where an agent is reporting a failure and requesting reconfiguration but the coordination agent repeatedly denies it because no alternative exists.

2.2.2 Genetic algorithms

Genetic algorithms (GA) are search and optimization tools that are able to cope with difficult problem domains with attributes such as discontinuity, time-variance, randomness, and noise [6]. Where planned reconfiguration is rigidly defined, GAs are practically the opposite. A GA attempts to mimic natural evolution. It is composed of individuals each of which is a solution to the given problem. In each cycle of the algorithm a new set of individuals are constructed. A fitness value is given to each individual quantifying how well it solves the problem. Like in nature, the fittest ones are more likely to "breed" new individuals for later generations. Breeding occurs when individuals swap data in the crossover event, creating offspring, and a probabilistic mutation is introduced. These offspring and the very fittest "parents" become the next generation and the process repeats. GAs terminate once a predefined number of generations is reached or a quality solution is found. [4]

GAs have traditionally been applied to system design or analysis rather than online reconfiguration. Real world problems that involve multiple conflicting objectives are well suited for GAs because an algorithm can be created to find partially optimal solutions that best improve a single objective without detriment to the others. By simply changing objective priorities, other partially optimal solutions can be found [6]. Given a cost function based on component reliabilities, cost, and weight a GA can also develop an optimized series-parallel system or fault tree with the greatest possible reliability [6]. Despite these benefits the inherent stochastic nature of GAs has restrained their use in safety critical systems.

Given the ability to verify their solutions, the major disadvantage of GAs is the relatively large amount of computation time required to produce a good solution. In time critical systems, such as aircraft, that experience failure events a GA would take too long to respond resulting in system failure. In rapidly changing systems a good GA solution for one state of the system may not be valid when it has finally been produced because the system has again changed. Modern parallel processing technology may finally be able to address the real-time computational problem, but the issue of specialization remains.

2.2.3 Learning classifier systems

One recovery approach that attempts to marry the rigid structure of planned reconfiguration and nondeterministic nature of GAs is a Learning Classifier System (LCS). In [3] Zeppenfeld describes an Autonomic System on a Chip utilizing a micro-rollback shadow register technique to eliminate CPU errors and an LCS variant to apply machine learning to system changes.

The simple LCS variant used in this design was a learning classifier table (LCT). The table contains a list of rules, each with an associated condition, action, and fitness value. When environmental monitors detect system strain or malfunction, the LCT is addressed for corrective action. The set of rules that relate to the current condition are isolated and from those a single rule is chosen based on a fitness weighted random draw. This selection approach allows the system to learn how well all the related actions correct the problem while still relying most heavily on the traditional solutions. When an action has been taken on the system its fitness value is updated to reflect its effect on the problem.

A hierarchical design of autonomic elements was also described to deal with larger systems and still provide information sharing between branches. Each branch would share performance metrics and fitness calculations in an attempt to keep a global view of system optimization and avoiding local maximums or conflicting branch activity.

An LCT approach is an improvement over the fully scripted healing program of planned reconfiguration because it allows the controller to discover in real time which rules best address the operational and environmental conditions the system experiences. Unlike a GA approach, the LCT system still requires preliminary knowledge of the system in order to define the LCT actions during development. Furthermore, since the creation of new rules is not capable online, LCT is limited in its ability to handle drastic events that greatly modify the system.

2.2.4 A hybrid healing approach

The three aforementioned paradigms incorporate the possible ways to address the issue. A completely defined reconfiguration plan, while most easily enacted and verified, does not allow for handling unforeseen problems. Evolutionary methods such as GAs can provide the most optimum solutions, but they require large amounts of time for computation and re-verification. In non-safety critical systems the use of GAs for complete system reconfiguration may be exactly what is called for. However, the need for system stability at all times in the safety realm means a trade off is required. Some reconfiguration ability must be relinquished in order to make verification and stability possible.

Nevertheless, this trade off does not mean that some form of GAs cannot be used. As defined in [3], the LCT system is a decent combination of controlled reconfiguration and online learning. Its inability to handle larger failure events stems from its unchanging set of rules. The authors briefly described a full LCS system, which employed a GA to develop new rules when a system event occurred. Such a hybrid system would have the ability to evolve its healing capabilities, the rule set, when drastic events occur but still perform system modifications in a controlled manner through the rule selection table. In order to validate such a system, however, re-verification must be performed after new rule sets are introduced to ensure a reliable system. The next section describes such a process that can achieve validations while avoiding the traditional state space explosion problem.

3 Dynamic controller verification

Thanks to advances in fault tolerance, reconfiguration, and autonomic principles it is feasible to create a system that can adapt to a great deal of environmental stress. However, from a safety critical perspective these systems are inadequate. Traditionally, safety critical systems undergo rigorous checks before being fielded, either through a formal methods development process, FMECA and Fault Tree analysis, or integrated safety monitors. An ever-changing state space, the directed graph representing all possible operational state for a system, forces a traditional verification scheme to continually recheck the entire space.

The remainder of this section will introduce some attempts to solve this problem. Formal methods for autonomic systems attempt to include the model verification software within the controller in such a way that adaptive behaviors are noticed and evaluated. Reachability analysis creates a system model and uses vector trajectories and boundaries to calculate reachable domains of a state space representation. Finally, runtime and on-the-fly verification techniques attempt to recheck an online system by reducing the state space in some manner.

3.1 Formal methods

In [12] Vassev and Hinchey describe a development framework for formally specifying, validating, and generating autonomic systems. This autonomic system specification language (ASSL) is a multi-tiered framework that treats autonomic elements (AE) like software agents that are capable of managing their own resources and relationships with other elements. ASSL takes a formal specification and generates JAVA code containing inline guards to evaluate activities and carry out definitive actions. The described autonomic nano technology swarm (ANTS) system is intentionally a high level model. Each AE is a

hardware device of a certain class, either a worker or manager. Workers are required to regularly transmit status messages to their manager, and the self-healing action of the system is to replace inoperable units with new elements from another system with a greater population or from a base station. While this approach can indeed produce verified software, the variability of the target system is very limited and therefore the controller and state space will experience little or no change.

ANTS research is continued in [10] with a greater focus on verification. The four methods of Communicating Sequential Processes, Weighted Synchronous Calculus of Communication Systems (WSCCS), Unity Logic, and X-Machines are evaluated as assurance techniques. A combination of WSCCS and X-Machines is ultimately deemed a possible way to model emergent behaviors of the swarm system, but such a model would only verify the behaviors after the fact not before use by the controlling AE.

3.2 Reachability analysis

Reachability analysis attempts to provide safety verification of continuous-time hybrid systems. Two categories of analysis are presented in [13]. The first is the decision whether or not a continuous region is reachable from another without explicated computation of the state space. The second is over-approximation of reachable space in order to provide a general idea of safety properties. Because of the infinite number of states in a continuous system, analysis is impossible without a discrete event model.

Both categories are centered on the idea of simplifying the calculated space in order to avoid the traditional state space explosion. Techniques such as addressing only barriers between regions or approximating irregular regions with convex polyhedrons are presented. However, simplification of these state space regions results in difficulties proving properties are not satisfied. Ultimately, even after these tradeoffs, reachability analysis is too complex for application to real world problems because the dimensionality is too great [13].

3.3 Runtime or on-the-fly verification

The basis of runtime verification is monitoring an online system's actions and ensuring safety properties are observed, either by forbidding illegal actions or forcing corrective action when an unsafe condition is reached. In [11] Alotaibi and Zedan employ a simple, expressive rule set composed of condition, action, and consequence tuples to create a safety policy. Safety assertions are inserted into agent implementation software to allow for monitoring, but the safety policy and behavioral software are developed

independently. This allows for easy policy modification simply by updating the rule set and rerunning the insertion process.

Alotaibi and Zedan do not address adaptive systems but their descriptions of control mechanisms are valid for any monitored system. These mechanisms can either be enforcement or validation structures. Enforcement is more invasive as it prevents actions from being carried out if an assertion fails. Validation, on the other hand, is recovery oriented and system actions are allowed to take place if an assertion fails. The failure is then reported to the controller so any possible corrective action can be taken. While enforcement provides stricter safety control, validation allows the system more autonomy. For example, in the event of a severe, unrecoverable failure an enforced system would likely be unable to take any further action whereas a validation approach may allow a system to be able to gracefully shutdown despite assertions failures.

A preliminary verification framework similar to [11] for adaptive systems is presented in [17]. The paper proposes a framework consisting of a configuration model that defines invariant system requirements in an abstract manner and a reconfiguration activity model that defines proper management actions for the system. Accordingly, a proposed configuration is verified against the configuration model to ensure the resulting system functions properly, and that configuration's deployment is validated against the activity model to ensure reconfiguration is achieved safely and completely. While the framework in [17] is only a preliminary design, its current disadvantage is the type of adaptable system it addresses. While a configuration is selected and validated at runtime against these models, the configurations from which to choose are static. Therefore, this framework requires a planned reconfiguration system as described in the previous section.

A model-based framework for runtime verification is presented in [14] which employs both the enforcement and validation paradigms. The framework executes a checker on a system model in an interleaved manner with the system implementation's execution. To reduce computation requirements, the system is only checked in critical states and reduces the explored state space by starting from the corresponding state in the model.

Since the model is being verified rather than the implementation, the checker can advance in the model past the currently executing state in any idle time. By checking the future path of execution in an enforcement manner problems can be avoided by directing the system to another path. However, if execution demands are too high the verification process can allow the system to proceed ahead of the checker. In this situation a post-checking, validation role is taken by the checker. Any encountered problems are

reported to the execution thread to be dealt with.

While the model-based framework provides versatility in verification techniques and greatly reduces computation requirements by reducing the searched state space, the possible disconnects between the model and implementation are a substantial disadvantage. Errors detected in the model will not necessarily be encountered in execution. Therefore the checker is conservative by nature, possibly overly so. Component implementations that deviate from the model will enhance this conservative nature and possibly create inconsistency cases during runtime. Another disadvantage is that the checker requires component models to be created and proven correct offline prior to use. Consequently, this verification framework must be used on a planned reconfiguration system or component modeling must be at a high enough level to allow for unanticipated component reconfiguration that will not invalidate the model.

A multi-agent runtime validation approach is presented in [15]. Each agent in the system is a Finite State Automaton that learns through an evolutionary algorithm (EA). The system consists of either many agents acting on a single execution plan or many agents with individual plans. In development, offline learning is applied to each unique task in order to develop the most efficient plan of execution. Once fielded, the evolutionary algorithm's population size is reduced to one and allowed only a single change per generation. While this restraint produces a more predictable learning system than a complex EA, it also restricts the system to use in gradually changing environments.

A final incremental verification system that also uses the validation paradigm is presented in [16]. The Cooperative Intelligent Real-Time Control Architecture (CIRCA) uses partial verification techniques in each phase of online controller synthesis to ultimately produce a verified controller, which ensures both state transition paths and logical timing requirements. A controller algorithm executed at each action point selects safe actions for the reachable states at that moment, using a model checker to ensure the safety of each selection. Therefore, once the controller has determined which states will possibly be selected next in execution it has also determined that those states are safe. Unsafe paths discovered in this process are reported as counter examples and used by the algorithm to revise the set of reachable states. A second algorithm is used to verify reachability with respect to timed actions. Essentially, the first algorithm verifies each portion of the controller synthesis and the second algorithm verifies the system by checking these portions as a whole.

To overcome the state space explosion this process would normally experience Musliner and Pelican have taken an on-the-fly approach to the state space calculation.

An implicit state space consisting of the initial state and set of transitions is employed rather than an explicit representation. Furthermore, only reachable successor states and automaton products are computed in the synthesis algorithm, rather than every discrete state in the system. Once a reachability result has been calculate it is stored in a continuation table for use by the next verification search if that controller path has not been modified. These three measures produce an efficient verification technique when the controller algorithm is not forced to backtrack and flush the cached results due to an unsafe controller path. They report that the incremental verification tool saves up to 97% of time on a sample problem versus the traditional complete verification. Even when backtracking occurs and the results cache is cleared, CIRCA produces little additional overhead compared to a complete verification because, like any verification tool, the majority of computation time is in the state space calculations. [16]

CIRCA's incremental verification process combines a time-abstract state space search algorithm and a timed automaton reachability algorithm to produce an on-the-fly verification tool. Unlike Spears' tool in [15] this process includes timing restrictions allowing it to provide real-time guarantees. However, no manner of system evolution is specifically presented by Musliner and Pelican. In order to employ this verification technique in an evolving system a means to inform the controller of new states and system capabilities must be implemented.

4 Semantic net framework

As we have discussed earlier, there are a variety of properties that a potentially defective component of a safety critical system may have and must be identified and confirmed in any replacement component with similar but not necessarily identical properties. We are examining a framework based on semantic web methodologies for organizing the properties of reusable computational components that may be selected during initial system design or during a recovery process for components that fail while in service.

“The semantic web is an extension of the current Web in which information is given well-defined meaning, better enabling computers and people to work in cooperation” [22]. Just as the current web has been based on ever increasing complex description languages including HTML, XML, and VRML the semantic data in the semantic web has been represented by an ever increasingly complex specification language ranging from RDF (Resource Description Framework), DAML (DARPA Agent Markup Language) to OWL (Web Ontology Language).

Extensive work has been done in building frameworks for software engineering applications using the semantic

web together with formal semantics [19, 20, 21].

In our work, the behavioral view of a component might be a model such as a Simulink or UML model. Alternatively, the behavioral view might be represented by a C language implementation of the component if targeting a normal processor is desired while alternative behavioral views might be represented in VHDL or Verilog if hardware targets are planned.

Each of these behavioral views will have descriptions that characterize the different capabilities of the component. These capability descriptions can be searched for using either exact or partial match capabilities of semantic web systems. For example, a component might be a camera mounted on a motion platform and one could locate the component either by searching for a camera capability, a motion platform capability, or both. Similarly, one might search for a FFT component that had a specific latency or power consumption, depending which property was paramount.

For each component there will be formal description of the precursor conditions and consequents that are associated with a component. Thus for a computational component there may be a "proof" of it's worse case timing if used as a C program and a different "proof" of it's worse case timing if used as a hardware component. An example of the former might be an algorithm to estimate the timing on a microprocessor while an example of the latter might be a petri-net model containing unit delays of the critical paths. The pre-conditions for re-using the "proofs" will be clearly laid out so that compositional verification of an assembly of components can be carried out without necessarily having to do a complete verification of the entire system when the components are used in accordance with their "proofs" thus avoiding the state space explosion problem discussed earlier.

Similarly, proofs about other properties such as safety properties may be maintained in a petri-net representation of the component so that the properties of live-ness and deadlock can be ascertained without having to prove these properties in the specific context of reuse. There will be times when a component is used in ways that might require partial proof reconstruction, such as when a component is used not exactly in the way it was originally intended or when some pre-conditions do not hold.

5 Conclusions and future work

Many challenges still exist for the development of a safety critical adaptive system. While a general approach for such a system is desirable, the tradeoffs involved in selecting the system's capabilities are currently too polarizing for a panacea approach to be designed.

Extensive work has been done in the field of fault diagnosis and therefore a selection of diagnosis techniques for a system can be relatively straightforward. If each hardware element is controlled by its own agent, as with an autonomic element, the agents ensure proper operation of the hardware and proper interaction with the rest of the system. This hierarchical approach allows the system controller to only be concerned with global system states and transitions and not everything down to the hardware level.

The methods for handing a failure in an adaptive system vary considerably. A simple system can employ a planned reconfiguration approach that is validated offline and easily implemented in the controller. However, most systems are too complex for this approach or they require the ability to react to unforeseen environmental changes. GAs allow more reconfigurable systems than any other approach, but their nondeterministic nature can endanger time critical systems that must ensure adaptation in a short period of time. While a learning classifier system is more adaptive than a planned reconfiguration system is still evolves in a controlled manner that can have difficulty coping with a drastic system event. Ultimately, a hybrid of a small population GA and a LCS would provide the best way for a system to carry out changes to controller after a wide range of failure events. The GA would enable the system to develop new rule sets to cope with environmental changes, and the LCS would execute those rules on the system in a timely, controlled manner.

The state space explosion problem has traditionally plagued online V&V systems. An incremental approach of only addressing a small portion of the state space in each verification cycle can overcome this issue. While all three runtime verification tools presented above have merit, the on-the-fly controller in [16] provides the most versatile online verification technique. A combination of a hybrid adaptive controller and this on-the-fly validation may result in a successful, general use adaptive safety critical system.

Finally, our approach to using a semantic web representation of the design space will allow for inclusion of a variety of semantic knowledge, from behavioral descriptions including software codes to formal specifications of the expected pre-conditions for behavioral code reuse.

6 References

- [1] J.O. Kephart and D.M. Chess, "The vision of autonomic computing," *Computer*, vol.36, no.1, pp. 41-50, Jan 2003.
- [2] H. A. Muller, L. O'Brien, M. Klein, and B. Wood. Autonomic Computing. Technical report, Carnegie Mellon

Univeristy and Software Engineering Institute, April 2006.

[3] J. Zeppenfeld, A. Bouajila, A. Herkersdorf, W. Stechele, "Towards Scalability and Reliability of Autonomic Systems on Chip," *Object/Component/Service-Oriented Real-Time Distributed Computing Workshops (ISORCW)*, 2010 13th IEEE International Symposium on , vol., no., pp.73-80, 4-7 May 2010.

[4] F. Siefert, F. Nafz, H. Seebach, and W. Reif, "A genetic algorithm for self-optimization in safety-critical resource-flow systems," *Evolving and Adaptive Intelligent Systems (EAIS)*, 2011 IEEE Workshop on , vol., no., pp.77-84, 11-15 April 2011.

[5] M. Khalgui and H.M. Hanisch, "Reconfiguration Protocol for Multi-Agent Control Software Architectures," *Systems, Man, and Cybernetics, Part C: App. and Reviews, IEEE Trans. on*, vol.41, no.1, pp. 70-80, Jan. 2011.

[6] P.J Fleming and R.C Purshouse, Evolutionary algorithms in control systems engineering: a survey, *Control Engineering Practice*, Vol. 10, Issue 11, November 2002, pp. 1223-1241.

[7] Yuanshun Dai; Yanping Xiang; Yanfu Li; Liudong Xing; Gewei Zhang, "Consequence Oriented Self-Healing and Autonomous Diagnosis for Highly Reliable Systems and Software," *Reliability, IEEE Transactions on*, vol.60, no.2, pp.369-380, June 2011.

[8] S. Byttner, T. Rögnavaldsson, and M. Svensson, Consensus self-organized models for fault detection (COSMO), *Engineering Applications of Artificial Intelligence*, Vol. 24, Issue 5, Aug 2011, Pages 833-839

[9] T. Rognvaldsson, G. Panholzer, S. Byttner, and M. Svensson, "A self-organized approach for unsupervised fault detection in multiple systems," *Pattern Recognition, 2008. ICPR 2008. 19th Int. Conf. on* , pp.1-4, 8-11 Dec. 2008.

[10] C. Rouff, M. Hinchey, T. Truszkowski, and J. Rash, (2004). "Formal Methods for Autonomic and Swarm-Based Systems". *1st Int. Symposium on Leveraging Applications of Formal Methods (ISoLA 2004)*. Cyprus.

[11] H. Alotaibi and H. Zedan, "Runtime verification of safety properties in multi-agents systems," *Intelligent Systems Design and Applications (ISDA)*, 2010 10th Int. Conference on, pp. 356-362, Nov. 29 - Dec. 1 2010.

[12] E. Vassev and M. Hinchey, "ASSL Specification and Code Generation of Self-Healing Behavior for NASA Swarm-Based Systems," *Engineering of Autonomic and*

Autonomous Systems, 2009. *EASe 2009. Sixth IEEE Conference and Workshops on* , vol., no., pp.77-86, 14-16 April 2009.

[13] Hervé Guéguen, Marie-Anne Lefebvre, Janan Zaytoon, Othman Nasri, Safety verification and reachability analysis for hybrid systems, *Annual Reviews in Control*, Vol. 33, Issue 1, Apr 2009, pp 25-36,

[14] Yuhong Zhao, Franz Rammig, Model-based Runtime Verification Framework, *Electronic Notes in Theoretical Computer Science*, Volume 253, Issue 1, 6 October 2009, Pages 179-193,

[15] D.F. Spears. "Assuring the Behavior of Adaptive Agents". *Agent Technology from a Formal Perspective*. Springer. 2006. Pre-Publication
URL: <http://www.cs.uwoy.edu/~dspears/papers/chap.pdf>

[16] D. J. Musliner, M. J.S. Pelican, and R. P. Goldman, Incremental Verification for On-the-Fly Controller Synthesis, *Electronic Notes in Theoretical Computer Science*, Vol. 149, Issue 2, 14 Feb 2006, pp. 71-90.

[17] L. Akue, E. Lavinal, and M. Sibilla, "Towards a validation framework for dynamic reconfiguration," *Network and Service Management (CNSM)*, 2010 *International Conf. on* , vol., no., pp. 314-317, 25-29 Oct. 2010.

[18] Rolf Isermann. *Fault-Diagnosis Systems: An Introduction from Fault Detection to Fault Tolerance*. Springer 2006.

[19] J. S. Dong, J. Sun and H. Wang, "Z Approach to Semantic Web", *International Conference on Formal Engineering Methods (ICFEM'02)*, LNCS, Springer-Verlag, pp. 156-167, Shanghai, 2002

[20] J. S. Dong, "Software Modeling Techniques and the Semantic Web", *The 26th International Conference on Software Engineering (ICSE'04)*, ACM/IEEE Press, Edinburgh, Scotland, 2004

[21] Ontologies and the Semantic Web,
<http://www.cs.man.ac.uk/~horrocks/Publications/download/2008/Horr08a.pdf>

[22] "The Semantic Web Made Easy",
<http://www.w3.org/RDF/Metalog/docs/sw-easy>

Everyday Cloud Computing with SaaS

Robert F. Roggio¹ and Tetiana Bilyayeva¹ and James R. Comer²

¹School of Computing, University of North Florida, Jacksonville, FL 32224

²Computer Science Department, Texas Christian University, Fort Worth, TX 76129

Abstract - This paper consists of three major sections that describe cloud computing services. The first section explains what is cloud computing and also summarizes the: (1) benefits of cloud computing from a business prospective and (2) different levels of cloud computing architecture such as Software as a Service (SaaS), Platform as a Service (PaaS), and Infrastructure/Hardware as a Service (IaaS/HaaS). The second section compares three major SaaS providers' services. The following companies were selected for a comparative analysis: Google, Zoho and Salesforce. The comparative analysis compares applications of the three companies that are similar products; such as: 1) Zoho and Google online office suit applications, 2) Zoho CRM and Salesforce CRM and 3) application markets that been launched by all three companies. The final section describes the future of software-as-a-service provided by cloud computing.

Keywords: Cloud Computing, SaaS, Zoho, Salesforce.com, GoogleApps

1 Background

1.1 Cloud Computing Definition

In the Information Technology world, cloud computing is one of the hottest and most popular topics. With the appearance of Web 2.0 and Web 3.0 technologies, the management of applications and data storage has begun to shift from the personal servers to "the cloud." For many users, cloud computing is not a new service as Google Docs has been in use for many years. National Institute of Standards and Technology (NIST) currently defines cloud computing as "a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g. network, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction." [1] This Cloud model promotes availability and is composed of five essential characteristics: On-demand self-service; Broad network access; Resource pooling; Rapid elasticity; Measured Service. Figure 1 shows a typical cloud computing system.

Cloud Computing provides hosting different applications and information technology services so that

they can be deployed and scaled quickly. Cloud providers accomplish this by investing in large, general-purpose compute infrastructures and using virtualization to divide this infrastructure up between multiple consumers and services. Cloud capacity can be easily added or removed to a specific service. As total capacity demands grow, the cloud provider deploys additional low-cost.

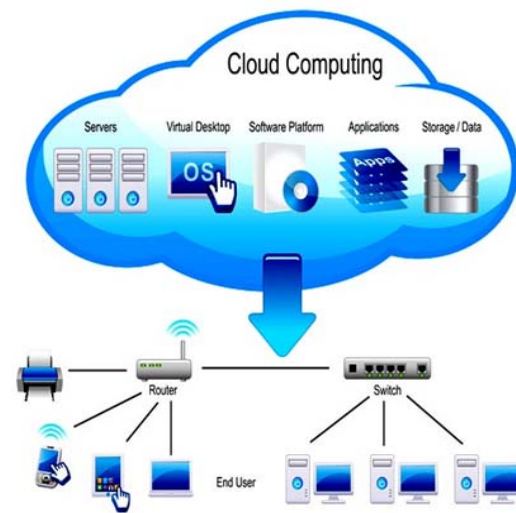


Figure 1: Cloud Computing System. [2]

1.2 Benefits of Cloud Computing

Rajkumar Buyya, James Broberg, and Andrzej M. Goscinski detailed several business benefits to building applications in the cloud [3]:

Almost zero upfront infrastructure investment: If a company is considering the development of a large-scale system, it may cost a *fortune* to invest in real estate, physical security, hardware (racks, servers, routers, backup power supplies), hardware management (power management, cooling), and operations personnel. Due to the high upfront costs, the project would typically require several rounds of management approvals before the project could even be initiated. With utility-style cloud computing, there is no fixed cost or startup cost.

Just-in-time Infrastructure: In the past, if your application became popular and the systems or the ... infrastructure did not scale, you became a victim of your own success. Conversely, if you invested heavily and did not get a ROI, you became a victim of your failure. By

deploying applications in-the-cloud with just-in-time self-provisioning, you do not have to worry about pre-procuring capacity for large-scale systems. This increases agility, lowers risk and lowers operational cost because you scale only as you *grow* and only pay for what you use. Source of these ideas?

More efficient resource utilization: System administrators usually worry about procuring hardware (when they run out of capacity) and higher infrastructure utilization (when they have excess and idle capacity). With the cloud, they can manage resources more effectively and efficiently by having the applications request and relinquish resources on-demand. Insufficient or excess computing power / resources are not issues...

Usage-based costing: With utility-style pricing, you are billed only for the infrastructure that has been used. You are not paying for allocated but unused infrastructure. This adds a new dimension to cost savings. You can see immediate cost savings (sometimes as early as your next month's bill) when you deploy an optimization patch to update your cloud application. For example, if a caching layer can reduce your data requests by 70%, the savings begin to accrue immediately and you see the reward right in the next bill. Moreover, if you are building platforms on the top of the cloud, you can pass on the same flexible, variable usage-based cost structure to your own customers.

Reduced time to market: Parallelization is the one of the great ways to speed up processing. For example, assume that for one compute-intensive or data-intensive job that can be run in parallel takes 500 hours to process on one machine. With cloud architectures, it would be possible to spawn and launch 500 instances and process the same job in 1 hour. [4] Having available an elastic infrastructure provides the application with the ability to exploit parallelization in a cost-effective manner reducing time to market.

1.3 Cloud Computing Architecture

Cloud Computing aims to be global and to provide such services to the masses, ranging from the end user that hosts its personal documents on the Internet to enterprises outsourcing their entire information technology infrastructure to external data centers.

Figure 2 represent the layered organization of the cloud stack from physical infrastructure to applications. These abstraction levels can be viewed as a layered cloud computing architecture where services of a higher layer can be composed from services of the underlying layer. The cloud computing architecture consists of: *Software as a Service (SaaS)*, *Platform as a Service (PaaS)*, and *Infrastructure/Hardware as a Service (IaaS/HaaS)*.

Infrastructure as a Service (IaaS) provides general on-demand computing resources such as

virtualized servers or various forms of storage (block, key/value, database, etc.) as metered resources. It is also known as *Hardware as a Service (HaaS)*. This is often viewed as a direct evolution of shared hosting with added on-demand scaling via resource virtualization and use-based billing.

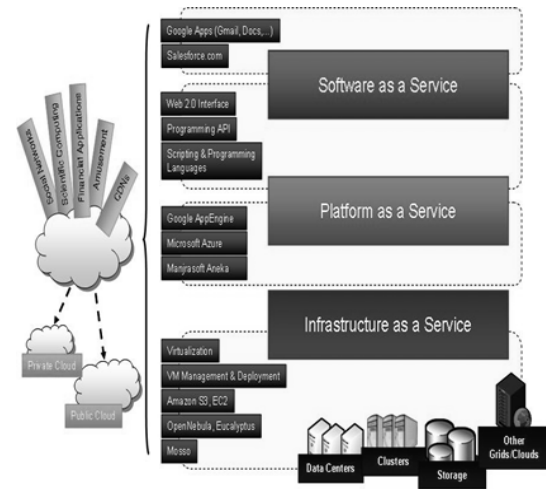


Figure 2 - Cloud Computing Architecture. [5]

Platform as a Service (PaaS) provides an existent managed higher-level software infrastructure for building particular classes of applications and services. The platform includes the use of underlying computing resources. PaaS offers a faster, more cost-effective model for application development and delivery. PaaS provides the entire infrastructure needed to run applications over the Internet.

Software as a Service (SaaS) is a software distribution model in which applications are hosted by a cloud service provider and made available to consumers over a network (usually the Internet). Organizations do not install software on their own computers; instead, they simply use their browser to access the software as it is provided over the Internet (software provided as a service). SaaS is becoming an increasingly prevalent delivery model as underlying technologies that support web services and service-oriented architecture (SOA) mature and new developmental approaches become popular. SaaS is most often implemented to provide business software functionality to enterprising customers at a low cost while allowing those customers to obtain the same benefits of commercially licensed, internally operated software. In addition to be cost efficient, it allows users to avoid the complexity of installation, management, support, licensing, and high initial cost of commercially licensed, internally operated software.

As a part of a business model of cloud computing, SaaS has significant value. *Controlling software licensing costs* are reduced by utilizing a software service provider licensing, patches, upgrades. An organization pays for what it needs

(software elasticity). A cloud service provider allows the customer to establish an approved applications list and keep it enforced. Also IT departments do not need to support random applications specific to one of users. Further, streamlining application support improves efficiencies, expertise and keeps everyone working and *controlroque software installations*. Finally, *infrastructure expenses* are reduced because web-based application access, which allows companies to purchase only the amount of desktop power needed for company service requirements.

According to a study by Mertz, SaaS is becoming increasingly important in most enterprise application software (EAS) markets. Worldwide software revenues for SaaS delivery are forecasted to grow by 19.4% overall between 2008 and 2013. [6] Researchers see promising opportunities for the successful adoption of SaaS, especially in those application markets requiring low levels of system customization (e.g., Office suites).[7]

Today, SaaS is visible in such applications as Salesforce.com and Google Apps. Salesforce.com, which relies on the SaaS model, offers business productivity applications (CRM) that reside completely on their servers. This allows costumers to customize and access applications on demand. Companies have also unveiled SaaS applications for individual customers. Examples include Google's spreadsheets and Microsoft's OneCare service; the latter provides virus and spyware cleanup for personal computers.[8] Other examples offered by variety of cloud providers include: Customer Service, HR, Project Management, Web Conferencing, Helpdesks, Wikis, Blogs and other intranet like applications.

There are several different cloud providers. Depends on your business goal, needs and budget, you always can find a suitable cloud computing provider.

2 Google, Zoho, and Salesforce Comparison

According to David Hilley, cloud computing is a promising industry on the cusp of high growth that is attracting many potential entrants.[9] There exists a variety of software-as-service providers and vendors that offer tailored services depending on client's business needs and budget. Figure 3 shows major players in cloud computing world.

The business and economical reasons for cloud computing are a direct result of various advantages enjoyed by Cloud Computing technology heavy weights such as Google, Amazon, Salesforce, Zoho and others. Three competitive companies were selected for this study: Google Apps, Zoho and Salesforce. All three companies

offer wide spectrum of services that are competitive with each other. For example, both Google and Zoho offer office suite applications that provide consumers virtually everything needed in terms of office software that are available online through their respective websites. The services are comprehensive, reliable, and surprisingly inviting. These services include e-mail service, calendar, document editor, spreadsheet, and other secondary office applications. Zoho also offers Customer Relationship Management (CRM) that is similar to CRM services offered by Salesforce (i.e., accounts, contacts, quotes, etc.). However, Salesforce offers more comprehensive and advanced services versus ZoHO. From the consumer's prospective, it is prudent to compare services and pick the perfect service matching one's own business needs.

Cloud Computing Provider	Layer
Akamai	PaaS, SaaS
Amazon Web Services	IaaS, PaaS, SaaS
EMC	SaaS
Eucalyptus	IaaS open source software
Google	PaaS(AppEngine), SaaS
IBM	PaaS, SaaS
Linode	IaaS
Microsoft	PaaS (Azure), SaaS
Rackspace	IaaS, PaaS, SaaS
Salesforce.com	PaaS, SaaS
VMware vCloud	PaaS, IaaS
Zoho	SaaS

Figure 3: Major Cloud Computing Providers. [10]

Table 1 (See Appendices) shows the compatible services of three different cloud computing providers. The table was created after detailed research of service spectrum of three SaaS providers.

What follows below is more focus comparative review of the major cloud computing services; such as: 1) Zoho and Google online office suit applications, 2) Zoho CRM and Salesforce CRM and 3) application markets that have launched by all three companies.

2.1 Zoho and Google Office Suit Applications

With the advancement of web technologies over the past several years, Internet tools offer many of the same features as MS Office at a greatly reduced cost or for free. These tools or applications work within any web browser. This past year, Google received significant attention with its Google Docs and Spreadsheets, and Gmail applications. The Zoho Suite, probably a better integrated but less-known, offers the same tools as Google suite free to individuals at a lower price. For example, the Google Apps suite is priced at \$50 per user/year and the equivalent Zoho Business service is priced at \$40 per user/year. Zoho supporter Raju Vegesna has stated that "Zoho is targeted at business users while Google's aiming

for consumers, Zoho offers 20-30% more functionality than Google". [11]

Both Google Apps and Zoho Business Service offer browser-based office applications such as word processor and spreadsheet; communication tools like chat and email, and collaboration tools like project managers and wikis. However the business-oriented Zoho and the consumer-oriented Google applications do vary slightly in offered services. This section of the study compares Zoho and Google's currently offered office web services.

Both Google Docs and Zoho Business Service are available as stand-alone products and offer reasonably full-featured word processing capabilities. Google Docs has a simple look to it, but it contains most of the basic formatting options available in Word. More advanced features such as pagination and adding footnotes are not available with Google Docs. Fortunately, users can save their Google documents in different formats—including Word, RTF, PDF, HTML and OpenOffice. Zoho Writer has the appearance of a traditional word processor interface offering all the features of Google Docs plus a few more. On the file-sharing side, both Google and Zoho have the ability to collaborate document production with other users including in real-time. Zoho makes it easier to work with MS Word with its recently introduced plug-in, which lets a user create a document in Word or Excel then exports it directly to Zoho so other participants can collaborate. In addition, both Zoho and Google also have a Publishing feature, which sends your document directly to a Web site or blog without requiring any knowledge about HTML or other Web software.

Generally speaking, the Google Calendar is easier to use than Zoho's. It is simple to add a meeting to the calendar and update scheduled events. One of its finer features is the ability to receive notifications by e-mail or by SMS text message for event reminders, daily agenda and invitations. The disadvantages include not being able to sync it with other calendar programs.

Unlike Google Calendar, the Zoho Calendar is not available as a stand-alone product. To use it, you must register for the Zoho Virtual Office Suite. Although it is similar to Google Calendar, Zoho's Calendar is not as easy to use. However, Zoho does provide a task list as well as the ability to export your calendar to other applications; features Google Calendar does not provide.

Google Spreadsheets and Zoho Sheet offer basic spreadsheet functions as well as more advanced formulas. They differ, though, in a couple of respects. Google does not allow the creation of charts while Zoho provides four basic chart templates. Also, Google permits real-time collaborative editing, but Zoho does not.

Zoho offers Zoho Show, which lets users create, edit, publish and show presentations remotely. It is important to note that Zoho, as with other web-based

applications, cannot match the features offered by PowerPoint. Perhaps Google understands this because it does not provide a presentation application.

Gmail was one of Google's first efficient tools, and it continues to be the most popular. All users receive at least 2.8GB of mail storage, if not more. GMail is quick and is accessible on your mobile phone. Zoho's E-Mail component, like its Calendar, is offered only through the Zoho Virtual Office Suite, but Zoho E-mail offers many more options than GMail. In addition to being able to organize your e-mail by folders, you can save an e-mail as a task, calendar item, note, document or other item.

2.2 Zoho and Salesforce CRM Comparison

The following table (Table 2) compares different characteristics of the Zoho CRM and Salesforce CRM services:

Both Zoho and Salesforce CRM are similar in functionality and base features. Therefore, cost and business needs are the driving factors in determining which one to utilize. Zoho is a good choice for small companies with limited budgets as it is free for the first 3 users. Additionally, it is also scalable allowing for future growth while providing many initial features. On the other hand, larger companies tend to benefit from Salesforce and the integration that has been developed between Salesforce and other business applications. Salesforce is a building out platform development offering excellent and comprehensive enterprise solutions.

While Salesforce is not a better option than Zoho as far as a CRM service is concerned, Salesforce is a safer choice for a larger companies and enterprises. However, Salesforce cloud services do potentially offer superior ROI than Zoho. Salesforce CRM can be integrated with Google Apps (Docs, Gmail, etc) if a company is using Gdocs.

2.3 Application Market from Zoho, Google and Salesforce

Table 3 (See Appendix) compares the cloud service applications offered by the three companies. The comparative table is based off of information provided several resources including websites of each provider [12], [13], [14] and web-based articles concerning the Zoho Marketplace, [15] Google Apps Marketplace [16] and AppExchange [17].

All three companies generated the cloud service market in order to expand their own already existing services. Zoho, Google and Salesforce have increased their presence by letting third-party companies or developers create, sell or buy applications that integrate with their cloud service applications. Understanding the dominate presence of cloud giants like Google, Zoho has

collaborated with Google Apps creating one of the biggest partners of the Google Marketplace. Applications are created using Zoho Creator that easily integrate with Google Apps. Salesforce has not collaborated with Google and has its own platform to create custom applications for AppExchange.

The cloud computing market continues to grow each year with AppExchange having the most number of existing applications on market; around 1300 applications. AppExchange leads the market due in part to the fact that Salesforce launched it in 2006; several years before Zoho and Google started their cloud application marketing.

The corporate environments and business strategies are very different among these three companies. Zoho offers complete freedom to developers and does not charge for transactions. Their main revenue stream is generated from license fees for another edition of Zoho Creator. Google is not as generous to developers; third-party companies keep just 80% of revenue and pay a \$100 to become a vendor. Out of the three companies, Salesforce has the most expensive service.

3 Future: Cloud and SaaS

Cloud computing has the potential to be as routine as traditional services such as electrical, telephone or water utilities. Cloud computing users would simply subscribe and pay for the usage in the same manner as traditional utilities. The primary attraction likely for continued growth is that it allows users and businesses to have access to needed technological infrastructure without having to invest in costly servers and information technology. The saving in capital expenditures allows users to concentrate on their core businesses.

The universal application of new applications is also likely to result in cloud computing use and growth. Developers would simply making new applications available to the cloud computing service provider who in turn place the new applications on its servers making new resources instantly available to every subscriber at the same time. The cost of marketing and selling new applications would be avoided significantly reducing indirect and direct overhead costs. This universal application of the new resource would lead to huge savings in computer time as newer software would be instantly available without any need for downloading it on to individual computers and system specific reconfigurations would be avoided.

In addition, the system requirements of end users would be less of a factor. Computers would not require high capacity hard drives as storage would be provided by the service providers. This could greatly reduce the cost of setting up information technology departments, which again need their own service setups. It will also allow users to keep what would otherwise obsolete computers in

production be for much longer. The key infrastructure needs would be simple upgrades of present systems to ensure fast access to the internet and the ability to use all the services that are on offer.

As all users will be using the same applications. The portability of data and information will be easy allowing companies to work worldwide by having all the same information and data available to all departments/offices/employees. This will greatly reduce the investment of time and resources traditionally expended on logistically managing uniformity across a given company's landscape.

The need for very costly high end servers for individual companies and institutions will become almost zero, as they would have full access to the servers of the service provider. What is expected is that the development of services will include providing huge resources for parallel computing. This expectation is just one aspect of the possible future development of cloud computing.

In Janna Anderson's report, she mentions that some experts predict that by 2020 most people will access software applications online and share and access information through the use of remote server networks rather than depending primarily on tools and information housed on individual, personal computers.

The highly engaged, diverse set of respondents to an online survey included 895 technology stakeholders and critics. The study was fielded by the Pew Research Center's Internet & American Life Project and Elon University's Imagining the Internet Center. Approximately 71% of the respondents agreed that "By 2020, most people won't do their work with software running on a general-purpose PC. Instead, they will work in Internet-based applications such as Google Docs, and in applications run from smart phones. Aspiring application developers will develop for smartphone vendors and companies that provide Internet-based applications, because most innovative work will be done in that domain, instead of designing applications that run on a PC operating system." [18]

The 27% that disagreed with the above statement believed that "By 2020, most people will still do their work with software running on a general-purpose PC. Internet-based applications like Google Docs and applications run from smartphones will have some functionality, but the most innovative and important applications will run on (and spring from) a PC operating system. Aspiring application designers will write mostly for PCs." [19]

Among the other observations made by those taking the survey were: "large businesses are far less likely to put most of their work 'in the cloud' anytime soon because of control and security issues; most people are not able to discern the difference between accessing

data and applications on their desktop and in the cloud; low-income people in least-developed areas of the world are most likely to use the cloud, accessing it through connection by phone". [19]

So cloud computing software-as-service has a promising future. However, this promising future is contingent on infrastructure of a very high caliber so that connections to the Internet are not interrupted because of poor power or other problems.

4 Conclusions

It is expected that Cloud Computing will be the wave of the future in terms of computing. It is only logical that the cloud's economies of scale and flexibility will impact how technology evolves and how users implement these technologies.

Cloud computing offers real alternatives to IT departments for improved flexibility and lower cost. Markets are developing for the delivery of software applications, platforms, and infrastructure as a service to IT departments over the "cloud." These services are readily accessible on a pay-per-use basis and offer great alternatives to businesses that need the flexibility to rent infrastructure on a temporary basis or to reduce costs.

Architects in larger enterprises find that it may still be more cost effective to provide the desired services in-house in the form of "private clouds" to minimize cost and maximize compatibility with internal standards and regulations. If so, there are several options for future-state systems and technical architectures that these users should consider in order to find the right trade-off between cost and flexibility. Using an architectural framework will help users evaluate these trade-offs within the context of the business architecture and design a system that accomplishes the business goal. All of which will increase the return on investment as well as decrease operational costs normally involved with in house systems processing the same data as in the cloud.

Cloud platforms are not yet at the center of most people's attention. The odds are good, though, that this will not be the case in the next five years. The attraction of cloud-based computing, including scalability and lower costs, are significant and will unlikely be ignored. Application developers should expect the cloud to play an increasing role in the future.

5 References

- [1] Peter Mell and Tim Grance, "The NIST Definition of Cloud Computing," National Institute of Standards and Technology, 9/2011
- [2] <http://www.sandiegoitsystems.com/hosted-services/attachment/san-diego-cloud-computing/>
- [3] Rajkumar Buyya, James Broberg , Andrzej M. Goscinski, "Cloud Computing: Principles and Paradigms", A John Wiley & Sons, Inc. Publication (March 2011)
- [4] J. Varia, Cloud Architectures, <http://jineshvaria.s3.amazonaws.com/public/cloudarchitectures-varia.pdf>, 2007-07-01.
- [5] <http://texdexter.wordpress.com/2010/09/08/cloud-computing-architecture-part-1/>
- [6] S.A. Mertz, C. Eschinger, T. Eid, H.H. Huang, C. Pang, B. Pring "Market Trends: Software as a Service", Worldwide, 2008–2013, Gartner, Stamford, CT (2009)
- [7] C. Pettey Gartner says 25 percent of new business software will be delivered as Software as a Service by 2011 available at <http://www.gartner.com/it/page.jsp?id=496886>
- [8] Richmond, R. 2005, "Microsoft to offer free trial of computer-security service", The Wall Street Journal (November 30).
- [9] David Hilley, "Cloud Computing: A Taxonomy of Platform and Infrastructure-level Offerings", College of Computing Georgia Institute of Technology, April 2009.
- [10] Yan Han, "Cloud Computing: Case Studies and Total Costs of Ownership", University of Arizona Libraries, Tucson, Arizona, 2009.
- [11] <http://lifelifehacker.com/315256/zoho-suite-vs-google-docs>
- [12] www.zoho.com/creator/marketplace/marketplace.html
- [13] www.google.com/enterprise/marketplace/
- [14] www.salesforce.com.
- [15] <http://www.eweek.com/c/a/Enterprise-Applications/Zoho-Apes-Salesforcecom-With-Apps-Marketplace/>
- [16] <http://googleblog.blogspot.com/2010/03/open-for-business-google-apps.html>
- [17] <http://www.zdnet.com/blog/saas/how-is-appexchange-really-doing/324>
- [18] Janna Quitney Anderson, Elon University and Lee Rainie, Pew Internet & American Life Project June 11, 2010;
- [19] Janna Anderson, Lee Rainie, "The future of cloud computing", June 11, 2010; <http://pewinternet.org/Reports/2010/The-future-of-cloud-computing/Overview.aspx>

	Google	Zoho	Salesforce
Word Processor	Google Docs	Zoho Writer	
Spreadsheet	Google Spreadsheet	Zoho Sheet	
Slideshows	Google Presentation	Zoho Show	
Web Clippings	Google Notebook	Zoho Notebook	
Email	Gmail	Zoho Mail	Email and calendaring (one of CRM ¹ features)
Chat	Google Talk	Zoho Chat	Chatter
Wiki	(Google acquired JotSpot)	Zoho Wiki	
Application Market	Google Marketplace	Zoho Marketplace	AppExchange
Online Database		Zoho DB	
Project Management		Zoho Projects	
Web Conferencing	(Google acquired Marratech)	Zoho Meeting	
Customer Relationship Manager		Zoho CRM	Salesforce CRM solutions
Personal Organizer		Zoho Planner	Email and calendaring (one of CRM features)
Web site hosting	Google Page Creator		
Feed Reader	Google Reader		
Personalized Homepage	iGoogle	(Zoho Start Page only for Zoho Apps)	

Table 1: Comparative table of providing services of three main SaaS providers

Characteristics	Zoho CRM	Salesforce CRM
<i>Charges</i>	\$12 or \$25 per user per month	\$65 per user per month
<i>Free service</i>	Free version	Free 30-day trial
<i>Integration</i>	Integration with their complete suite of Zoho Apps	Integration via a web services API
<i>Business focus</i>	Small to medium business	Medium to large business
<i>CRM Features</i>	Sales force Automation, Marketing Automation, Support Management, Order Management, Reporting & Analysis, Workflow Management, Outlook Edition	Sales force Automation, Marketing Automation, Document Management, Contract Management, Customer Service & Support, Analytics, Mobile CRM, AppExchange
<i>Company strengths</i>	<ul style="list-style-type: none"> - Lowest TCO with a rich feature set at a competitive price - Flexible deployment options (On-Demand as well as On-Premise CRM solution) - Ease of usage with web-based user interface, and Total customizability - Affordability 	<ul style="list-style-type: none"> - Sales force automation - Ease of use - Intuitive interfaces - Established status - Flexible customization - Integration and extensibility

Table 2: Comparison of the Zoho CRM and SalesForce CRM

¹ Customer Relationship Management (CRM)

Beyond Traditional Disaster Recovery Goals – Augmenting the Recovery Consistency Characteristics

Octavian Paul ROTARU
American Sentinel University

Octavian.Rotaru@ACM.org

Abstract

For most organizations the disaster recovery goals are limited to Recovery Point Objective (RPO) and Recovery Time Objective (RTO). This perspective on disaster recovery overlooks very important factors that can contribute to the successful implementation of a Disaster Recovery plan. Evaluating metrics beyond recovery time (RTO) and recovery point (RPO) is essential to meet the recovery commitments of an organization.

The purpose of this paper is to review existing Disaster Recovery metrics that can augment the Recovery Point and Recovery Time and to propose new metrics for Recovery Consistency.

Recovery Consistency Objective is measuring the total data consistency of your Disaster Recovery solution post recovery. Recovery Consistency Objective (RCO) ads data consistency objectives to the disaster recovery objectives of an organization, but often RCO is not enough to evaluate consistency.

Going beyond the traditional disaster recovery goals, this paper introduces an assessment method and metrics for the consistency of the module interfaces in addition to the module consistency. The RCO as well as the proposed interface consistency metrics are evaluated in the context of the seven disaster recovery tiers defined by SHARE User Group.

Keywords: Business Continuity (BC), Business Resilience, Disaster Recovery (DR), Disaster Recovery Goals, Recovery Consistency Objective (RCO).

1. Introduction

A successful disaster recovery plan has well defined goals that are in line with the business requirements. Defining the recovery objectives is one of the most important steps in creating a disaster recovery plan and the objectives are the result of the Business Impact Analysis (BIA).

The maximum acceptable downtime in case of a disaster will vary depending on the nature of the business and the financial impacts of the downtime. Depending on the criticality of the data handled and

service rendered, the business continuity approaches cover a wide range of options. Most organizations today depend heavily on their IT infrastructure and their data in order to be able to provide service to their customers and the recovery objectives will drive the selection of the disaster recovery strategy and the cost of the IT infrastructure required to support it.

Cyber-infrastructure protection, business continuity and disaster recovery, includes safeguarding and ensuring the reliability and availability of key information assets, including personal information of citizens, consumers and employees [13], but reliability and availability need to be backed by data consistency in order to provide proper recovery.

Susanto [9] considers IT to be the most important issues of all when discussing BC and DR, not only for being the foundation and backbone of the business but also because IT can play important roles in strategies development and improving efficiency of the whole BCP plan. In today's complex business environment data and application consistency is becoming more and more important.

As outlined in [14], managing a combined store consisting of database data and file data in a robust and consistent manner is a challenge for large scale software systems. In such hybrid systems, images, videos, engineering drawings, etc. are stored as files on a file server while meta-data referencing/indexing such files is created and stored in a relational database to take advantage of efficient search. Consistency between database content and files is required for the application to function properly post recovery.

Defining a Recovery Point and Recovery Time objective is often not enough to insure successful recovery following a disaster event. Recovery Consistency Characteristics (RCC), as well as Recovery Object Granularity (ROG) and Recovery Time Granularity (RTG) need to be assessed in order to discover risks to which the environment is exposed.

2. Metrics that augment the Recovery Point and Recovery Time Objectives

Proper evaluation of a disaster recovery solution requires well defined metrics and risk assessment. The Recovery Time Objective (RTO) and Recovery Point Objective (RPO) are usually driven by Service Level Agreements (SLA) that the organization is contractually or legally bound to.

RTO defines the time required to recover the lost data while RPO define the potential loss of data (the time gap between the most recent data point that can be recovered and the disaster event).

Even if RTO and RPO are enough to measure SLAs, these two metrics do not measure the overall consistency of the data or the risks to which the organization is exposed in case of a disaster event. Meeting the defined RTO and RPO doesn't mean that processing can be resumed. The recovered data may be inconsistent if components are recovered at different points in time. More comprehensive metrics are needed to assess the quality of the recovery plan.

The recovery metrics used by most organizations in their business continuity plans fall into three main categories:

1. Recovery Time Characteristics
 - 1.1. Recovery Time Objective (RTO) is the main Recovery Time Characteristics define for any DR solution and defines how quickly service (data and application) is recovered following a disaster scenario.
 - 1.2. Recovery Time Granularity (RTG) measures the time spacing required for selecting a recovery point. RTG defines a logical recovery point selection.
2. Recovery Data Characteristics
 - 2.1. Recovery Point Objective (RPO) defines the time gap between the disaster event and the point in time where data can be recovered. It is essentially a measurement of how much data (measured in time updates) is estimated be lost following a disaster.
 - 2.2. Recovery Object Granularity (ROG) is measuring the granularity of the objects that a disaster recovery solution is capable to recover.
3. Recovery Consistency Characteristics
 - 3.1. RCO measures the usability of recovered data by the associated applications. RCO is defined as percentages, evaluating the number of entities that are consistent after recovery.

RTG complements RTO and RPO in situations in which logical failures are encountered. For example, a data replication solution with a zero RPO and well

defined RTO will recover from a physical failure but not from a logical failure. Data corruption that is not detected in time will be replicated and compromise the ability to recovery. In such a situation, if no other way to recover exists, the RTG will be undermined, and another recovery solution need to be put in place to provide a recovery point in time in the past prior to the disaster event. As a result, RTO will highly increase.

The object granularity defined by ROG can be a storage volume, a file system, a database, a cluster package or service (including all associated storage), etc. Going below a volume or a file system in terms of granularity proves in most situations to be very expensive and requires manual intervention (labor intensive).

Measures the Recovery Consistency Characteristics in terms of only RCO is often not enough and the purpose of the next sections of this paper is to assess RCO and introduce new metrics to complement it.

3. Why RCO is not enough

Data is point in time consistent only if all of the interrelated data components are exactly as they were at any single instant in time.

Disaster recovery plans usually define only recovery point and recovery time objectives. The Recovery Time Objective (RTO) is the duration of time within which a business process must be restored after a disaster in order to avoid unacceptable consequences associated with business continuity disruptions. The Recovery Point Objective (RPO) is the maximum tolerable period in which data may be lost. In many circumstances, the consistency of the data may be compromised even if the RPO and RTO are met.

In this context the introduction of the Recovery Consistency Objective (RCO) is necessary in order to evaluate the data consistency following recovery. RCO is defined as a percentage measuring the deviation between the actual and the targeted state of business data across systems.

RCO is calculated as a percentage that measures the number of consistent modules of the system after recovery reported to the total number of modules of the system:

$$RCO = \frac{c}{t} = \frac{t - i}{t}$$

c = Number of Consistent Modules

t = Total number of modules

i = Number of Inconsistent Modules

where $t = c + i$.

Even if the recovery point objective and recovery time objective are properly evaluated and can be met,

the system can be restored in an inconsistent state, and some of the applications may not be able to properly recover.

Let's consider as an example a complex system that is spanning across multiple storage systems. In case replication is synchronous for all storage systems the alternate site will always be in sync with the main production site and data and application consistency will be preserved. However, if the storage systems are using different replication techniques, then the recovery point will be different for each of them.

Even if the general RPO is preserved (all storage frames at the alternate site have data at a point in time lower than the general RPO), the difference between the recovery points of different storage devices may result in data inconsistency between applications.

Modules may try to access data assuming that it is in sync and fail to find the records that are needed.

Similar inconsistencies may occur in case different replication techniques are used for data inside the same frame (synchronous and asynchronous).

Depending on the criticality of the application and the way applications or modules are implemented, such application data inconsistencies (different recovery points) may prolong the recovery time and may require manual intervention. Evaluation and correction will take time and extend the recovery time beyond what the business can tolerate.

The recovery consistency objective is very important in such situations and reflects the individual requirements of corresponding business data cross-system consistency.

An unplanned IT outage can equate to a disaster, depending on the scope and severity of the problem. RCO is more important than RTO and RPO in the context of BCP processes. RPO and RTO emphasize the traditional IT Disaster Recovery Planning, while RCO goes beyond DRP. An important part of preparing for a disaster is to understand the type of risks your organization is facing. The risk of data inconsistency following disaster recovery is measured by RCO.

The cost associated with creating and assuring availability for the enterprise rise dramatically as you approach the requirement for 100% availability. In certain contexts data inconsistency is acceptable as long as system availability is maintained and downtime is reduced. In other business environments data inconsistency may have staggering costs.

4. Improved consistency assessment metrics for disaster recovery

RCO measures the usability of recovered data by the associated applications. RTO calculated as described above does not take into consideration interfaces/links between application modules.

Let's assume for example that our system has 3 modules, and each module is consistent after recovery. In this example the RCO is 100%. However, if the point in time recovery for the 3 modules is different, some of the links between modules may be inconsistent.

In my view interfaces/links between modules of the same system need to be considered when calculating the overall consistency of the system following recovery. If all modules except one are recovered at the same point in time and one module at a different point in time, the interfaces between the module with the different recovery point and the others may be compromised.

A better way to assess overall consistency of the system is to combine RCO with a consistency objective for module interfaces (Recovery Interface Consistency Objective).

The proposed RICO can be calculated as a percentage as well based on the following formula:

$$RICO = \frac{ci}{ti} = \frac{ti - ii}{ti}$$

ci = Number of Consistent Module Interfaces

ii = Number of Inconsistent Module Interfaces

ti = Total Number of Module Interfaces

Combining RCO and RICO into a single measurement for consistency is more practical and can be achieved by merging RCO and RICO into one metric. The proposed measurement for recovery consistency is Recovery Total Consistency Objective and is calculated based on the following formula:

$$RTCO = \frac{m * \left(\frac{ci}{ti}\right) + n * \left(\frac{c}{t}\right)}{m + n}$$

where n and m are weigh parameters that can be defined based on the number of modules and interfaces and the importance of interfaces vs modules.

RTCO is covering both module consistency and interface consistency providing a better measurement than RCO.

5. Disaster Recovery Tiers and Goals

The SHARE User Group established 7 tiers of Disaster Recovery solutions. Each of these tiers addresses different requirements and corresponds to a different set of disaster recovery goals. The table below provides disaster recovery goals estimated for each of

the 7 Disaster Recovery tiers. Understanding the 7 tiers of Disaster Recovery and the goals associated helps organizations evaluate the DR solution that they currently have in place and determine what level is matching their business requirements.

		RPO	RTO	Consistency
Tier 0	No off-site data	X	X	X
Tier 1	Off-site vaulting	Defined	High	?
Tier 2	Off-site vaulting with hot site	Defined	Medium	?
Tier 3	Electronic vaulting	Defined	Low	?
Tier 4	Electronic vaulting to hot site	Defined	Low	?
Tier 5	Transaction integrity	Defined	Very Low	Yes
Tier 6	Zero or near-zero data loss	Defined	Almost Instant	Yes

Tier 0 - No offsite data

Tier 0 has no DR solution in place. There is no alternate location (hot site) available, no saved information, and no documentation and DR plans. Tier 0 offers no recovery options following a disaster. The ability to recover following a disaster is completely unpredictable and exposes the business to the risk of not being able to recover.

No DR goals can be defined and recovery time and recovery point are unpredictable.

Even if backups are done, the solution is categorized as Tier 0 if the backups are stored at the same location as the production environment and no proper vaulting procedure is in place.

Tier 1 – Off-site vaulting

Tier 1 DR is relying on backups that are stored at an offsite storage facility (vaulting). No alternate environment (location, hardware, etc.) is available where to restore the data in case of a disaster.

RCO depends on the consistency of the backups. Backups taken at different times for different application modules may render the application or some of its modules and interfaces inconsistent.

RPO is defined depending on the frequency of the backups. RTO is very hard to define - as a new site needs to be built from scratch (location, infrastructure and equipment), and usually RTO is higher than a week. Recovery time is dependent on when hardware can be supplied or when a building for the new infrastructure can be located and prepared, and can take months.

Tier 2 – Offsite vaulting with a hot site

Tier 2 is relying on backups sent offsite for recovery (same like) Tier 1 plus a hot site. An alternate location is available and backups can be transported there from the offsite storage facility in the event of a disaster.

The availability of a hot site reduces the recovery time. RTO can be estimated and is lower than in Tier 1. No time is required to locate an alternate location, purchase and install hardware. The RTO is driven by the time required to recall the backups at the hot site available and load them (restore). RCO and RPO are similar with those offered by Tier 1.

Tier 3 - Electronic Vaulting

Tier 3 includes everything offered by Tier 2 plus electronic vaulting of a subset of the critical data. Electronic vaulting requires communication lines between the 2 sites and the creation and transmission of backups (traditional backups or data replication) more frequently than traditionally in the regular backup process. The recovery time improves and can be as low as one day. The recovery point improves for the critical data that is electronically vaulted to the remote site. The recovery consistency objective is the same like in Tier 2 – the solution continues to be exposed to the risk of inconsistency. There is no notable RCO improvement when compared with Tier 2.

Tier 4 - Electronic vaulting to secondary active site

Tier 4 is comprised of two data centers with electronic vaulting between the two sites. The secondary site is also active and recovery can be bi-directional.

The workload is shared between the two active sites and critical data is continuously transmitted between them, while the recovery of non-critical data continues to rely on off-site vaulting.

Data loss is still possible in Tier 4 so the recovery point depends on the frequency with which the two sites are synchronized

The recovery time is lower, but the risk of inconsistency still exists – between critical and non-critical data.

Tier 5 - Transaction integrity (two-site, two-phase commit)

Tier 5 maintains selected data in sync between the 2 sites. Transactions involving the selected critical set of data will be committed in the same time at both locations (single commit scope). Both primary site and the secondary site need to be updated before the update request is considered successful. A high bandwidth connection is required between the two sites.

Recovery consistency is improved, but it is unsure if transaction integrity of critical data is enough to make the whole consistent.

Tier 6 - Zero or near-zero data loss

Tier 6 involves immediate transfer of data to the alternate site. Data replication is used in order to maintain the two locations in sync.

The recovery time is very low as well as the recovery point (data is in sync – so the data loss is minimal or zero).

There is not dependence on the applications or other resources to provide data consistency. Recovery consistency is assured. Data is considered lost only if a transaction has commenced but the request was not satisfied. This tier encompasses zero loss of data and almost immediate transfer to secondary platform. In such a configuration RCO is usually 100%.

6. Conclusions

Disaster recovery architecture and plans are driven by business requirements and consistency is often overlooked and not properly evaluated as a disaster recovery parameter.

The goal of this paper was to introduce new metrics for consistency following disaster recovery. Interface consistency is introduced and evaluated in addition to module consistency. RICO and RTCO are proposed as alternate metrics to complement RCO and enhance the Recovery Consistency Characteristics.

The disaster recovery goals (RPO, RTO and RCO) are evaluated in the context of the 7 tiers of disaster recovery defined by the SHARE User Group.

7. References

- [1] Octavian Paul Rotaru, *Architecting for Disaster Recovery – A Practitioner View*, WORLDCOMP, Proceedings of SAM 2011, July 2011.
- [2] Cathy Warrick, John Sing, *A Disaster Recovery Solution Selection Method*, IBM RedBook, <http://www.redbooks.ibm.com/redpapers/pdfs/redp3847.pdf>, February 2004.
- [3] C. Brooks, M. Bedernjak, I. Juran, J. Merryman, *Disaster Recovery Strategies with Tivoli Storage Management*, IBM RedBook, <http://www.redbooks.ibm.com/redbooks/pdfs/sg246844.pdf>, November 2002.
- [4] Philip Clark, *Contingency Planning and Strategies*, Proceedings of InfoSecCD 2010, October 2010.
- [5] Richard Cocchiara, *Beyond disaster recovery: becoming a resilient business*, IBM Global Services, ftp://ps.boulder.ibm.com/common/ssi/rep_wh/n/BUW03014USEN/BUW03014USEN.PDF, June 2009.
- [6] David Rudawitz, *Enterprise Architecture and Disaster Recovery Planning – On the way to an effective Business Continuity Planning Philosophy*, Antevorte Consulting LLC, http://www.antevorte.com/whitepapers/Enterprise_Architecture_and_Disaster_Recovery_Planning.pdf, November 2003.
- [7] N. Arshad, D. Heimbigner, A. Wolf, *Dealing with failures during failure recovery of distributed systems*, Proceedings of DEAS'05, NY, USA, 2005.
- [8] Y. Edmund Lien, P. J. Weinberger, *Consistency, concurrency, and crash recovery*, Proceedings of the 1978 ACM SIGMOD international conference on management of data, NY, USA, 1978.
- [9] Lukman Susanto, *Business Continuity/Disaster Recovery Planning*, 2003, <http://www.susanto.id.au/papers/bcdrp10102003.asp>
- [10] U.S. Department of Commerce – National Bureau of Standards, *FIPS PUB 87 – Federal Information Processing Standards Publication, Guidelines for ADP Contingency Planning*, 1981 March 27.
- [11] Geoffrey Wold, *Testing Disaster Recovery Plans*, Disaster Recovery Journal, Vol. 3, No. 3, p. 34.
- [12] Guy Witney Krocker, *Disaster Recovery Testing: Cycle the Plan, Plan the Cycle*, SANS Institute – InfoSec Reading Room, 2002.
- [13] Constatine Karbaliotis, *Critical Interests: Business Continuity, Disaster Recovery and Privacy*, Symantec, September 2009
- [14] Suparna Bhattacharya, C. Mohan, K W Brannon, I Narang, Hui-I Hsiao, M Subramanian, *Coordination backup/recovery and data consistency between database and file systems*, Proceedings of the 2002 ACM SIGMOD International Conference on Management of data.

A Smart Approach to Conference Registration and Payment Processing

J. Vandersall, K. Schwarting, and R. Lee

Software Engineering Information Technology Institute (SEITI), Department of Computer Science
Central Michigan University, Mount Pleasant, MI, USA

Abstract - *As the global web continues to expand there is an increased desire for a multitude of companies, varying greatly in scope and size, to have a secure payment processing infrastructure in place. Currently there are many options readily available for the completion of such a task; however, it is not uncommon for large fees to be associated with their initial setup. Our goal is to develop a smart, user friendly approach to aid in the registration payment process of academic conferences. In order to accomplish this, our web form must be able to be dynamically altered during initial setup. Our form will establish a connection to First Data Global Gateway Connect 2.0, allowing for secure payment processing. After comparing our system to other available options it is apparent that implementing our system has the potential to, decrease overall costs while simultaneously improving and quickening the initial set up process.*

Keywords: Smart Payment Processing, Simplified Registration, Easy Web Form Generation.

1 Introduction

We have set out to develop a secure payment processing system that is specifically geared to improve the conference registration process all the way from the initial set up, to the processing of the payment. What we have chosen to create must place a strong emphasis on both security and usability. This will allow for smooth and seamless payment processing that keeps all sensitive data secure. The final product must also have the ability to be regenerated with updated values so that it can be used for conferences in the future that have unknown price values for the various registration options.

There are numerous options that have been previously created to address the issue of payment processing for websites in need of such a service. However, the majority have set their focus on simply providing a way in which to accept a dollar value, process it, and return the approval or denial. If you are seeking a custom form that provides specific options for registrants to choose from, then often times there are large initial set-up fees involved with the creation of a form that will be static in nature. By this we mean that the form used to collect data regarding the customers choices will not have the ability to be altered, barring once again entering into the initial set-up and re-incurring said fees.

This process commonly begins by the conference manager receiving a form that they must fill out specifying the costs associated with the different registration options. A large problem with this is the time that it takes for this form to get back, undergo processing and become a final converted web form that is capable of accepting payments. We believe that many conference managers would prefer if they had an option that allowed them to simply answer these questions, and then have their form immediately generated. This is exactly what we have set out to accomplish.

One of the key factors that many conference managers may find appealing is the ability to link this service to a majority of banks. This provides for a means to not only expedite the reception of payments, but to also save on other more costly transaction/set-up fees. What makes it specifically ideal for conference registration is that it is coupled with a simple script that requests a series of inputs from the conference manager. The data collected during this initial setup covers everything from the name of the conference, to the price of registration and various other prices that correlate with all of the predefined different registration options. Upon completion of this series of questions you will have generated an HTML form that is contained within a PHP file, allowing for the ability to securely connect to First Data for payment processing.

In this research paper we propose the use and configuration of a pre-existing Electronic Payment Processing system, First Data Global Gateway Connect 2.0 (FDGG), as well as the use of a custom built python script that allows the conference manager to set the price of different registration options.

1.1 Sections

The subsequent sections will provide an overview of the research and work behind developing an application that will meet or exceed the expectations of conference managers. Section 2 outlines the specific methodology employed within this project. Section 3 discusses the overall project requirements for a successful system to be complete. Section 4 details the security features employed within our payment processing infrastructure. Section 5 conveys the faults found after performing rigorous testing of the web form, payment processing gateway and registration generation script. Section 6 reviews the results of this project and discusses possible

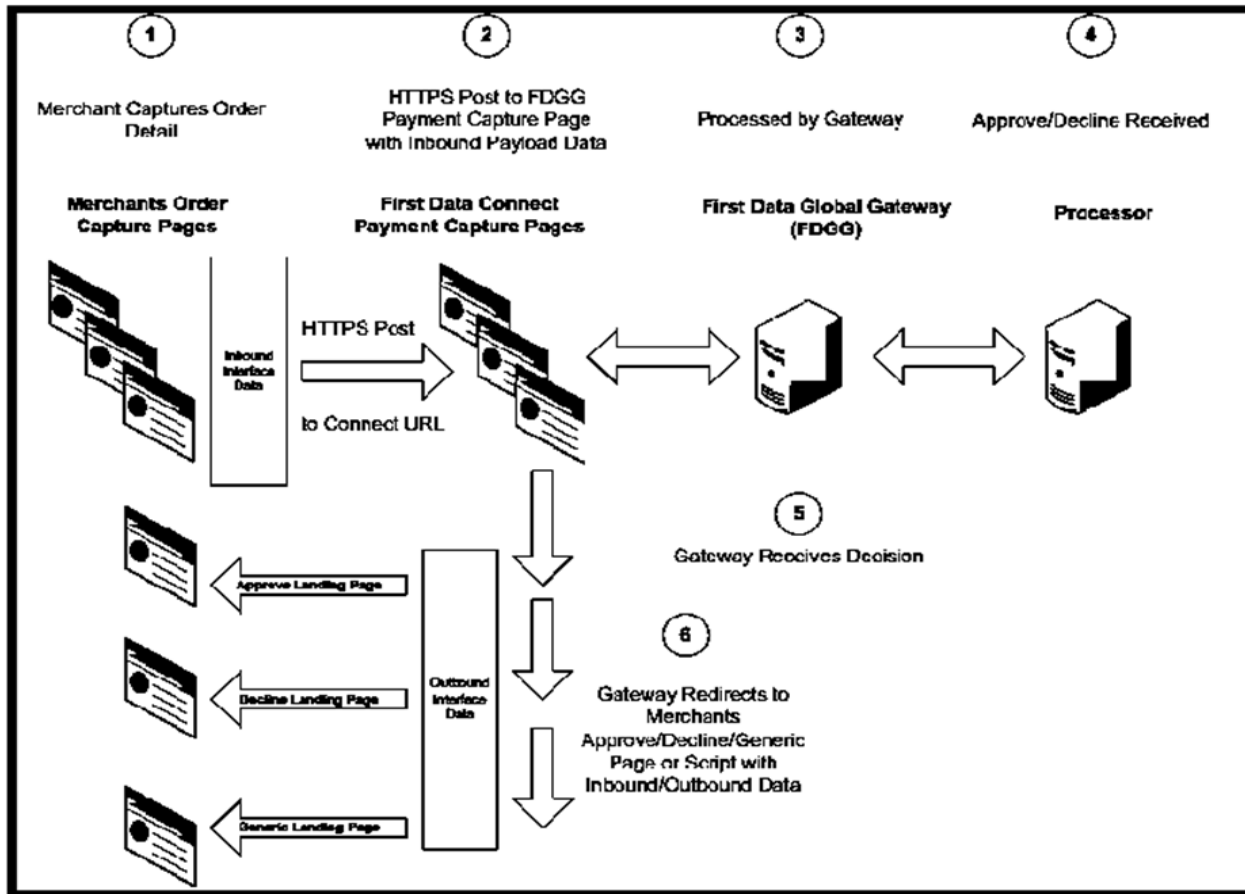


Figure 1- Data Flow Diagram for First Data Global Gateway Connect 2.0 [1]

modifications that we believe could be introduced to improve the overall usability, as well as features that could be added in future versions.

2 Methods Employed

In order to accomplish our goal of developing a smart approach to conference registration payment processing, we needed to be able to establish a secure connection to the FDGG. In order to do this we need to be certain that the connection between FDGG and the conference web server is one that cannot be replicated for illegitimate use. To ensure that only our site has the ability to connect we must send a hashed string that contains information that is specific to our conference site. This process is carried out by the following function. What is happening in the code below is the generation of a hash value. The store name, date and time, charge total, and shared secret are all concatenated into a single string. After this a For Loop is activated that measures the length of the new string and then steps through it character by character converting them into hexadecimal. These hexadecimal characters are placed in order within another new string that is then passed to FDGG. Once received, FDGG will verify that specific contents of the string correspond to a valid store within their databases. If a match is found then the payment information collection page specific for our individual store will be displayed. This page is hosted on the FDGG servers. [1][2]

```
function createHash() {
    $str = $storename . getDate() . $chargetotal .
    $sharedSecret;
    for ($i = 0; $i < strlen($str); $i++)
    {
        $hex_str.=dechex(ord($str[$i]));
    }
    return hash ('sha256', $hex_str);
}][2]
```

2.1 PHP Hypertext Processor

This powerful server side scripting language was employed to carry out the majority of tasks involved with actually getting the payment processed through FDGG. In order to do this many features of the language were used. Most notably is the use of the \$_POST variable that is built into the language. The \$_POST method is quite literally collecting all form data sent via a \$_POST and storing it into the \$_POST array. This array has its keys set to equal the form fields' names and the values set correspondingly. We are then able to invisibly pass the data to a page that is specified in the form action. On the associated page it is possible to collect any information passed via a \$_POST, by using a \$_REQUEST. [2][3]



The International Association for Computer and Information Science (ACIS)

RECEIPT

Author Name

April 27, 2012

Your paper number ____, entitled “ ____ ” has been registered for presentation at the 11th IEEE/ACIS International Conference on Computer and Information Science (ICIS 2012) at Shanghai, China, to be held on May 30 – June 1, 2012.

The following Fees have been received:

ACIS Member conference registration	\$500.00
Additional Page	\$100.00
Additional Lunch Tickets	\$ 50.00

Total fees	\$650.00 USD
-------------------	---------------------

Figure 2- Receipt Page as Displayed to Registrants

The predefined `$_REQUEST` method is very powerful as it has the ability to fetch data that is sent by both the `$_GET` and `$_POST` methods. We again took advantage of this feature in our receipt page that is displayed to the user after a successful payment has been processed. After the initial non-sensitive information is collected from the registration web form, it is passed to the FDGG servers which collect and process the sensitive payment information. We then have the FDGG pages properly configured to send the non-sensitive data back to our receipt page, allowing for the collection and generation of an accurate and personalized receipt. This receipt data is then stored on the conference server for proper record keeping. Figure 2 above depicts the receipt page, as seen by the registrant upon confirmation of payment. [2]

2.2 JavaScript

In our efforts to create a form that allows for users to make the selections necessary for their specific needs, and then use these selections to calculate a cumulative grand total, we have chosen the client side language JavaScript. We have determined that the ability to auto calculate the grand total will increase the user friendliness of our form while at the same time greatly reducing the possibility user error. This reduction in user error will allow the process to be more efficient and

will also eliminate the need for additional charges and refunds to be issued. This is a great feature to help aid in reducing the quantity of unnecessary transaction fees incurred. [3] In order to make this a reality the following method was employed.

```
function calcT() {
  reg = document.mainform.presenterReg.value;
  stu = document.mainform.studentReg.value;
  paper = document.mainform.addPaper.value;
  page = document.mainform.addPage.value;
  lunch = document.mainform.addLunch.value;
  dinner = document.mainform.addDinner.value;
  document.mainform.chargetotal.value = (dinner
  * 1) + (lunch * 1) + (page * 1) + (paper * 1) +
  (stu * 1) + (reg * 1);
}
```

This simple, yet incredibly effective method grabs the values associated with the various form options and stores them each individually into an appropriately named variable for further processing. It then takes the newly created and populated variables and proceeds to add them altogether to calculate a charge total. This calculated charge total is then set as the value of the form field displaying the total amount that is representative of the choices made on the form. It is also important to note that the displayed charge total is only

updated upon calling the above function. As such we have set each of the form options to have an event listener. More specifically, all drop down style input fields are set to have an onChange event listener make a call to the function. This creates a page that will update the total charge value displayed to the user immediately after a choice is made or altered.

In order to properly keep track of those who have registered for the conference we must generate a unique identifier for each individual registrant. This was done in a manner similar to the way that we have chosen to calculate our charge total. However, rather than using an onChange event listener we have chosen to go with an onKeyUp event listener.

For the case of the drop down style input, an onChange event was perfect for displaying a newly calculated charge total after a selected option was chosen. This was not the case for the text box input style. It was meant to be very clear that the users order ID was being generated based on the users input into the three categories that were chosen for their unique traits: Last Name, First Name and Submission Number. Specifically in the order just presented, these three traits are simply concatenated into a single string. This string is what we have chosen to use as an order ID. The following function performs what we have just described. [3]

```
function track(){
  fname = document.mainform.firstN.value;
  lname = document.mainform.lastN.value;
  order = document.mainform.subN.value;
  document.mainform.order.value = (lname) +
    (fname) + (order);
}
```

2.3 Python

Python is a versatile high level programming language and is used in the generation of the PHP form, as well as in the gathering of user input for the pricing of registration options. We used multiple features of this language to write the program. Through the use of many simple but powerful attributes of the language, we are able to ask specific questions to the conference manager, in an easy to understand and prompt manner. The responses gathered from the conference manager's answers to these questions are the prices that will be assigned as the values of the specific individual registration options. This allows for our form to both quickly and accurately collect all required responses from the associated fields. These variables are then directly injected into the framework of the PHP. This allows us to gather the responses quickly and accurately. [4][5]

After the answers are properly collected they are then injected into a PHP framework and the resulting file is saved, and ready to be uploaded to the web server. With the use of this small program, the entire conference registration setup is

streamlined to require only a few minutes from start to finish, saving the managers weeks of waiting in some cases. [4][5]

3 Ideal Project Requirements

This project required several steps of development in order to properly and satisfactorily meet our requirements. There are several key aspects that are considered to be primary requirements for the fully functioning system as outlined below.

3.1 Requirement 1

One of the earliest requirements of this project was getting the web form to properly and securely communicate with First Data Global Gateway. This was in part accomplished through the aid of a test server that FDGG has in place. In order to securely communicate with the server the web form must pull in a PHP script that generates a hash value. This hash value is a product of the combination of multiple unique values. A few of these values are as follows: store name, shared secret key and other variables. The generation of this hash value is discussed further in the following section V Security Features. After communication was properly established with the test server we were able to make the appropriate modifications required in order to switch over and begin communicating with the Production Server. This required the configuration of the FDGG Virtual Terminal. The Virtual Terminal is an online interface that allows for the management of many settings that correspond to the payment gateway. [1]

3.2 Requirement 2

The second requirement was to find a way to complete transactions, while at the same time never storing or handling any sensitive data regarding our clients' payment details. Security is considered a mission critical priority. To aid in achieving this, all sensitive data handling is carried out on the FDGG secure web servers. This allows us to assure our conference registrants a much more secure method of payment processing, while at the same time maintaining a speedy and user-friendly web form interface, which allows for the easy selection of registration options.

3.3 Requirement 3

A third requirement that was essential to the longevity of this project was the ability to generate new forms, with modified values, for future events. This was considered to be one of the more important tasks in our opinion as it truly sets this project apart from other similar payment processing options. As previously mentioned, there are often times large fees associated with the configuration of a unique web form, which will be shortly outdated. By allowing for the form to be simply regenerated by the conference manager, we are able to save a significant amount of both time and money. This will eliminate the need of bringing in a web developer to set up a new form for each individual event, which adds many benefits.

4 Security Features

The FDGG allows customers to select registration options on the conference site prior to being transferred to the payment form. This payment form is hosted on the FDGG secure payment gateway, which allows for maximum security when dealing with sensitive information, such as credit card numbers and various other payment details. This also allows us to accomplish our goal of keeping our site completely separate from anything that is gathering credit card numbers, greatly increasing our security. This security is accomplished through the use of 128 bit SSL (Secure Sockets Layer) protocols, coupled with both strong encryption and server authentication via a shared secret. Despite the high level of security involved with this process, First Data states that the secure payment gateway has an average response time of less than 6 seconds. Additionally keeping all credit card information off of the conference server provides dissociation that further aids in security. [6]

4.1 Encryption

Data security and integrity are vital aspects of encryption and hashing. Essentially when we send our traffic to the FDGG server, we first must send the server our store name and shared secret. Directly before this transfer is conducted, we convert our data into hash values in order to keep the data both secure and to ensure integrity. If the store name and shared secret were sent in plain text it would create the possibility of potentially fraudulent charges being made to and from the account. As such, we chose to use one of the encryption methods that are accepted by the FDGG. When establishing communication with the FDGG server, our script uses the SHA-2 algorithm to encrypt the Store Name, Date, Time, Charge total and the Shared Secret Key. After this hash value is created the encrypted information is sent to the FDGG server to be processed and authenticated against their records. Sending our store name informs FDGG who is sending the data and thus who to send a response to. The charge total that is sent to the FDGG allows for the payment collection process to know how much to charge the registrant. This is all protected by a shared secret key that was also a portion of the generated hash value that was sent to properly confirm our individual store's identity. [6]

4.2 SHA Algorithms

SHA is short for Secure Hashing Algorithm which is a current and highly trusted family of algorithms. The family of

SHA algorithms started in 1993 with SHA-0, a name retroactively given to the original SHA algorithm that was removed from service early on due to a significant flaw that created hash collisions very easily. The SHA-1 algorithm was introduced in 1995 and is still in use today, and while it was originally the standard hashing algorithm of the United States Government, it was phased out in 2010 by the National Institute of Standards. Many websites (including the popular source control system Git) still use this algorithm, either for data encryption or data integrity. [6]

4.2.1 Shared Secret Key

A shared secret is a piece of data that is exchanged during secure communication. This piece of data can be a password, a passphrase or commonly a long string of randomly generated numbers. The key can be created at the start of communications or shared beforehand, though that is commonly called a pre-shared key. The shared secret is generated via a key-agreement protocol (such as Kerberos) and then used to authenticate with the other party, in our case the FDGG server. By sending the FDGG server this secret key, we can prove our identity thus adding another layer of protection from data fraud. [1][6]

5 Fault Testing

It was important to ensure that our web form met all of the requirements mentioned above in section IV. In order to make this a reality we needed to be certain that all errors were either removed or corrected on our form. This was done by rigorous testing of the form, which was aided by the availability of a test server provided by First Data. This allowed us to continually submit the form to the test server hosted by FDGG. It was helpful in determining flaws due to the fact that it functioned identically to the one used in final production, with a single exception; it would not present the form for payment information collection and therefore would not process the payment. The other nice feature that this provided was a means of testing while not incurring any usage fees for the test submissions. [1][2][3][4]



ACIS Registration Page

First Name Jon	Last Name Vandersal
Submission Number (This was issued to you by ACIS when your paper was submitted) 12	
OrderID	VandersalJon12
Registration Date	May 18, 2012

Presenters and Attendees Conference Registration ** (Includes Proceedings, Meals and Social Functions) ** "Existing Member" means you registered for a previous conference this year as a member. "New Member" has the membership fee included in the \$580 USD.	Existing Member ▾	@ 500
Student Non-Presenters (Includes Proceedings, Meals and Social Functions) (NO Paper to Present)	▾	@ 0
Registration of Additional Papers	0 ▾	@ 0
Do You Require Additional Pages (Eight Pages are included in regular registration)	1 ▾	@ 100
Do You Require Additional Lunch Meal Tickets (1 for every event is included with registration)	2 ▾	@ 80
Do You Require Additional Dinner Meal Tickets (1 for every event is included with registration)	2 ▾	@ 120
Charge Total		\$ 800
<input type="button" value="Submit This Form"/>		

Figure 3- Registration Form as Displayed to Registrants

5.1 Common Faults

The faults most commonly found within our forms and Python script were as follows:

- Unwanted additions, deletions or modifications of key aspects of the form.
- Non-essential user access to form options or information.
- Errors in syntax on the Python side and the HTML / PHP / Java Script side.
- Variables with incorrectly associated values being dynamically produced via the Python script.

6 Results

Our results are on par with our expectations. By removing the unnecessary costs and time most commonly associated with the initial set-up of such a payment processing system, we are able to reduce the burden placed on conference managers as well as the expenses incurred. We have found our alternative method to not only be a viable solution, but also one that offers more flexibility while remaining aware of future implementations.

6.1 Conclusions

The final product can be best described in the following four steps:

6.1.1 Step 1:

The conference manager begins by running our python script which gathers input based on a series of questions. This input is then used to fill the PHP framework with the necessary pricing for the various options. The page is then generated and hosted on the conference web server which is viewed by the conference registrants.

6.1.2 Step 2:

When the user loads the conference registration page, they are presented with a user friendly form that details all possible registration options. Upon the user inputting the appropriate variables, they will have an order ID generated live on the page. Next, the user will begin to select the relevant options for their registration. The selecting of these options will call the function that is responsible for calculating the charge total. This charge total is updated and constantly displayed to the user. Once all options have been chosen the user will submit the form causing them to be redirected to the FDGG website for payment processing. We implemented JavaScript here, as previously mentioned in section III, to calculate the total cost for the user, thus removing the option to under/over pay and reducing overall transactions to and from the conference's bank.

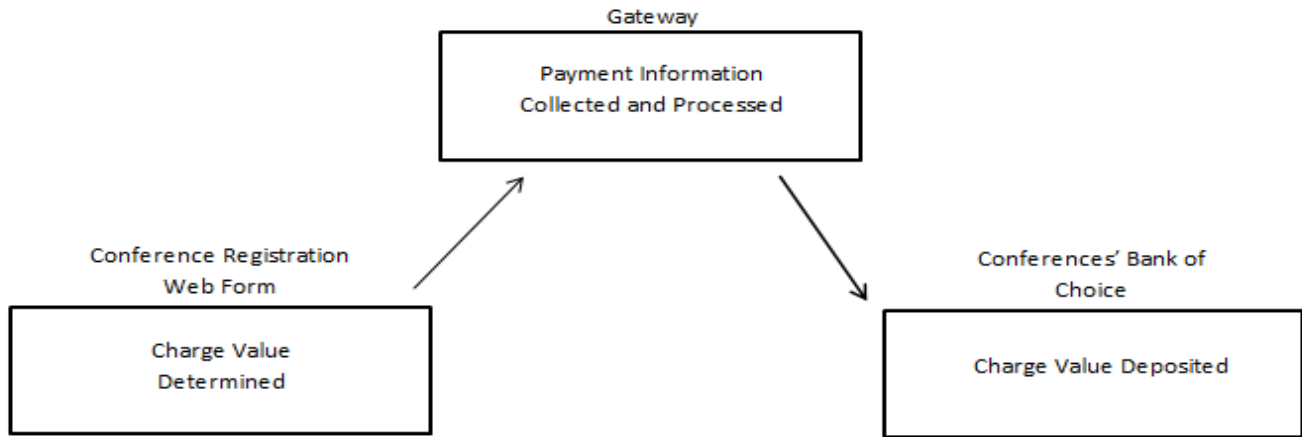


Figure 4- Payment Flow Diagram [1]

6.1.3 Step 3:

Once the user is transferred to the FDGG payment collection form they will be prompted to enter their payment information. Payment information will be comprised of the sensitive data that we wish to separate ourselves from in efforts to promote maximum security. Generally the data collected by FDGG will be the credit card number and other relevant information required for processing. The amount to be charged is passed to FDGG using a `$_POST`, as described in section 2.

6.1.4 Step 4:

If the transaction is successfully processed then the user is redirected back to the conference server where a custom receipt page is displayed. This receipt page includes details regarding information pertaining to their purchase including costs to them for particular options, paper number, title and order ID. This receipt is printable for the registrants' records and a copy of the data is stored on the webserver for conference records. If the transaction is declined the user is redirected to a denial page describing the issue and from there they can repeat the process and correct the errors.

6.2 Afterthoughts

With the final completion of this payment processing system, we have begun a complete evaluation of all features. This has allowed us to determine that all core features are presently functioning properly; however, the generation of the web form by means of the python script shows room for further improvement. It is important to keep in mind that one of our main goals is not only to process payments, but also to provide for ease in initial setup and to add the ability for changes to be made to the cost of individual form options that will vary from one conference to another. In future versions of this payment processing system we feel that a graphical user interface (GUI) version of the web form generation script would increase the overall user friendliness. In fact, it would

appear that contrary to our original development plans, which called for the use of a separate python script in order to generate the web form, we may be better off with a simple web interface management system. Such a management system may be developed for future releases.

7 Acknowledgments

Our thanks go to Lucas Falsetta for his assistance with some of the more subtle syntax errors. Also we would like to thank Dr. Roger Lee for being our supervising faculty member for this project.

8 References

- [1] F. Data, First Data Global Gateway Integration Guide Connect 2.0, 2009
- [2] W3Schools, PHP Tutorials, 2012
- [3] W3Schools, HTML Tutorials, 2012
- [4] D. M. Beazley, Python Essential Reference, 4th ed., 2009.
- [5] B. Dayley, Python Phrasebook: Essential Code and Commands, 2007
- [6] National Institute of Standards and Technology, Secure Hashing, 2012

Extensible Web-based Learning Architecture

Takao Shimomura and Kenji Ikeda

Dept. of Information Science and Intelligent Systems
University of Tokushima, Tokushima, JAPAN

Abstract— *Computer-aided learning helps teachers estimate how effective their lessons are and students know to what extent they have learned. This paper describes an extensible e-Learning architecture, which teachers can easily conduct, and with which students can enjoy learning. Students can view all subjects at a time only by accessing the home page of the system, and they can start practicing any of the subjects without logging in the system. The system utilizes several mechanisms to promote students' motivation and to encourage them to compete with each other. Teachers can easily design the contents of Web pages by using HTML edit menus, and those menus can also be easily defined and enhanced. Custom links and the programming interface the system provides make it possible to enhance the system itself.*

Keywords: Customizable; E-learning; extensible; games; installation; motivation.

1. Introduction

E-Learning is not almighty, nor can it teach everything [1], [2], [3]. The sensation of frustration, loneliness and confusion, which some participants express, is the result of both, lack of involvement and participation of some students as their peers opted out, and lack of involvement and feedback on the part of the facilitator. It is said that women are more strongly influenced by perceptions of computer self-efficacy and ease of use, and that men's usage decisions are more significantly influenced by their perception of usefulness of e-learning. We believe that what are most required for students to study a subject and understand it as far as they get satisfied are a good teacher and a good book. It is a good teacher who teaches students plainly enough and step-by-step. It is a good book that helps a student when he or she reviews the subject and wants to understand it more deeply. On the other hand, e-Learning is suitable for playing such auxiliary roles as to help teachers estimate the effectiveness of their lessons or to help students know the current status of their abilities.

Currently, several e-Learning systems are available [4], [5], [6], [7], [8]. However, when we install and conduct conventional e-Learning systems, we fell some difficulties. It may take a lot of time to develop the contents of courseware, or we might be forced to use ready-made courseware. Because of a complicated e-Learning system architecture, students need to perform several operations to visit the Web

pages of interest. To get over those difficulties in the existing systems, we have developed an e-Learning system, Apty [9], [10], [11], which teachers can easily conduct, and with which students can practice any subject of interest. This paper presents an enhanced version of Apty in which several mechanisms are embedded to promote students' motivation and custom links and the programming interface the system provides make it possible to enhance the system itself. In addition, any language can be installed to display Web pages. Teachers can easily design the contents of Web pages by using HTML edit menus, and those menus can also be easily defined and enhanced. The paper also describes the system implementation of Apty and demonstrates one example of custom links for a collaborative network-based game. Only a password is required for a teacher to install the system as an administrator. An installation tool automates the complicated settings of a Web server and a database server. Students can view all subjects at a time only by accessing the home page of the system, and they can start practicing any of the subjects without logging in the system. The system utilizes several mechanisms to promote students' motivation and to encourage them to compete with each other. Students can enjoy games together and if some of them have studied harder, it may be advantageous in the games.

2. Automated installation of the system

An installation tool we have developed makes it easy to install the Apty system. Only a password is required for a teacher to install the system as an administrator. A default directory will be automatically displayed for the installation directory. This installation tool automatically finds appropriate vacant ports to determine the port numbers of Tomcat and Postgresql servers, and sets up their configuration files. Well-known ports such as 80 (http) and 443 (https) are not used at the default setting to enable an ordinary user (an unprivileged user) to install and conduct the system.

(1) Inserting an Apty DVD into a drive of a computer will start the Apty installer automatically. (2) A language option, English or Japanese, can be selected if necessary, and a password is entered for an administrator. (3) The installer automatically finds five appropriate vacant ports and displays their numbers. Three ports are used for Tomcat Web server. One port is used for Postgresql server. The other one port is used for a game server. (4) When the "Install" button is clicked on, it will load Java and IDE from the DVD.

(5) It will install Tomcat Web server by setting three ports as http, https and shutdown ports, and load the Apty Web application. It will also set the default locale of the Apty system based on the selected language option. (6) It will then install Postgresql database server. It will set a port for postmaster, start the database (DB) server, and create Apty DB and Trivia DB for an example subject Trivia. These databases are created based on the chosen language option. Then, it will set the Apty administrator's password as the DB user's password. Only the Apty administrator is privileged to access and modify these databases by using psql commands. (7) It will create an Apty system information file (README.html), which displays the installation information of the Apty system such as the URL of the Apty system, the Apty administrator's password, and some commands to control Tomcat and Postgresql servers. This file can be accessed only by the user who has installed this Apty system. (8) It will start Tomcat Web server, and it will wait for all the ports to open. (9) It will then start a standard Web browser to display both of the Apty home page and the Apty system information page (README.html). (10) It will exit, and instead start an Apty administration tool, (11) by which the administrator can easily start and stop Tomcat and Postgresql servers.

The Apty system consists of a Web server (Tomcat) and a database server (Postgresql). The database server administers aptydb database that stores the subject information of each subject, and subject databases, each of which will be created when a new subject is created. By using a Web browser, students access the Web site of the Apty system.

3. Accessible subjects to anyone

3.1 Students' Web page transfers

In this system, students can access any subject of interest. Figure 1 shows the state transition of Web pages conducted by a student. Students can access all pages shown in Fig. 1 (a) without login. They first choose a subject they are interested in by checking a list of subjects shown in the left frame of the Apty home page. Then, they display the corresponding subject home page to practice it, play a game, or view top 10 rankings in exams.

While students are playing the game, a chance problem will be set at random. The first player who answers it correctly can get an extra character and will be able to get more points. The chance problem is selected at random among questions which have been set up as problems for practice. If students have practiced those problems harder, they can gain a higher ascendancy in the game. The pages shown in Fig. 1 (b), on the other hand, are accessed after login. Login is required for students to take exams, review the questions answered wrongly, and submit their papers for assignments.

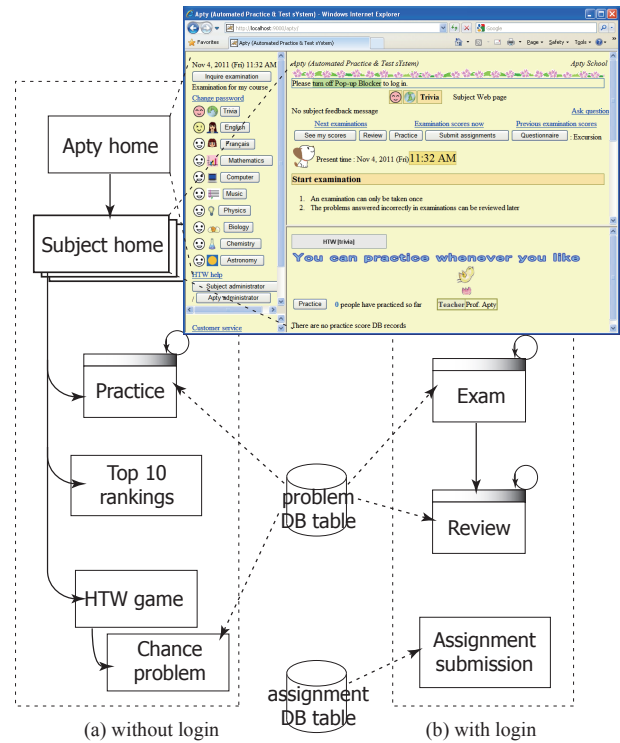


Fig. 1: Student's state transition of Web pages

3.2 Administrator's Web page transfers

Only the Apty administrator can create a new subject and control all information of the Apty system including DB tables. If a teacher is given a subject administrator ID and a subject administrator's password by the Apty administrator, he or she will become the subject administrator of the corresponding subject, and they can control all information of that subject. Students practice problems, take exams, and submit their papers as answers to the assignments given by their teachers. Although the system can automatically calculate exam and practice scores, it cannot automatically grade students' papers. Students' papers need to be graded by their teachers. To ease the teachers' work, another type of administrators, that is, assignment graders have been introduced. Teachers can delegate their jobs to the assignment graders. The assignment graders are teachers' assistants, and instead of teachers, they are left to grade papers.

When the Apty administrator and subject administrators (teachers) log in as an administrator, the Apty administration page will be displayed, which shows a list of all subjects. The Apty administrator can operate on any of these subjects. On the other hand, when teachers (subject administrators) log in, only the subjects they are allowed to operate on will be displayed as clickable buttons. When they click on one of the buttons, the corresponding Subject administration page will be shown. They can designate assignment graders, create assignments, create questions for practice and exams,

conduct a confirming test to make sure of the created questions, create tests for practice, exams, and questionnaires, and check several kinds of students' scores such as exam scores, correct answer ratios, and review completion ratios. Assignment graders are only allowed to grade students' papers. To prevent them from acting as administrators, they are forced to log in as graders from subject home pages in the same way as students. When they log in, a list of all assignments will be displayed with all papers submitted for those assignments. For each assignment, a teacher (subject administrator) or one of the graders is assigned to grade papers submitted for that assignment. The teacher (subject administrator) has a privilege to grade the submitted papers of any assignment even if it is assigned to another grader.

3.3 Four methods for student authentication

To take exams, students need to be authenticated by entering their passwords. When they want to review the questions answered wrongly, when they want to see their own exam scores, or when they want to submit their assignment papers, they also need to be authenticated. For this system, four authentication methods have been developed as shown in Fig. 2. First, the Apty administrator registers students in the student list file to record the account names (student IDs), the passwords, the course names, the mail addresses, and the full names of the students. The authentication methods are first classified into two categories, file authentication and account authentication, which is further classified as custom authentication, PAM authentication and NIS authentication.

The file authentication shown in Fig. 2 (a) authenticates students by their account names and passwords recorded in the student list file. The account authentication shown in Fig. 2 (b) can authenticate students more flexibly. The custom authentication enables the Apty administrator to determine how to authenticate students. The administrator can implement any authentication method by defining the `canLogin()` method of the `CustomAuthenticate` class. The PAM authentication uses the Pluggable Authentication Modules to authenticate students, where the passwords they entered to log in their computers are used. The NIS authentication authenticates students by using the account information obtained from a NIS server by issuing a "ypcat passwd" command. This NIS authentication enables an ordinary user (not a super user) to administer this system. To make it unnecessary to specify a specific account authentication method, the Apty system automatically judges which account authentication method is available in turn in the order of (1) custom, (2) PAM, and (3) NIS authentications, and determines an appropriate one. The left bottom frame of the Apty home page shows an authentication method currently in use.

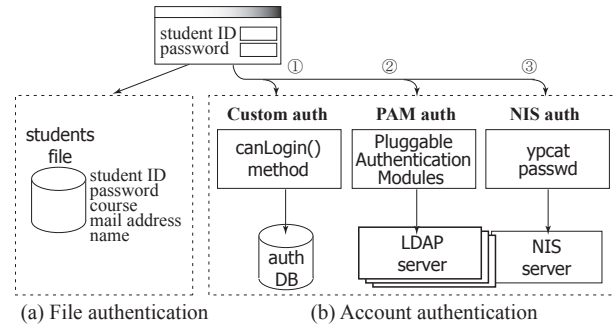


Fig. 2: Authentication of students

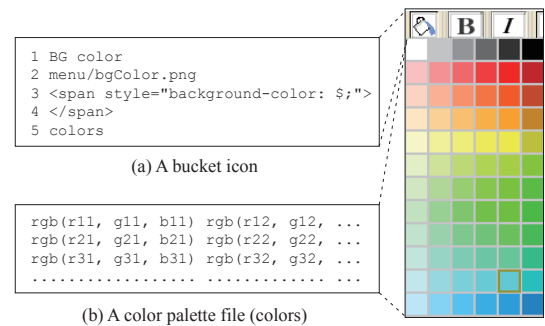


Fig. 3: Definition of a bucket icon

4. Extensible learning architecture

The Apty system enables users to customize several functions and enhance the system itself. This section demonstrates some examples of the customization and enhancement of the system.

4.1 Extensible HTML edit menus

To easily create questions using a Web browser, two edit modes, HTML view mode and HTML code mode, have been introduced. The HTML view mode enables teachers to edit the contents of Web pages while they are viewing the Web pages that will be actually displayed. For example, if you want to change the background color of the text "sea" to blue, you first select this text with the mouse. When you then click a bucket icon from the edit menu, a color palette will appear. As soon as you choose a blue color cell, the background color of the text will turn blue. You can see the result immediately because this system uses JavaScript and does not communicate with the server to update the display of Web pages. The HTML code mode enables you to enter more complicated HTML code directly, and you can easily make sure of the result by switching from the HTML code mode to the HTML view mode.

Here, let's consider customizing menu items of the edit menu, and adding some new menu items. For example, the bucket icon is defined as illustrated in Fig. 3 (a). The first line in Fig. 3 (a) specifies the tool chip text of the menu item, and

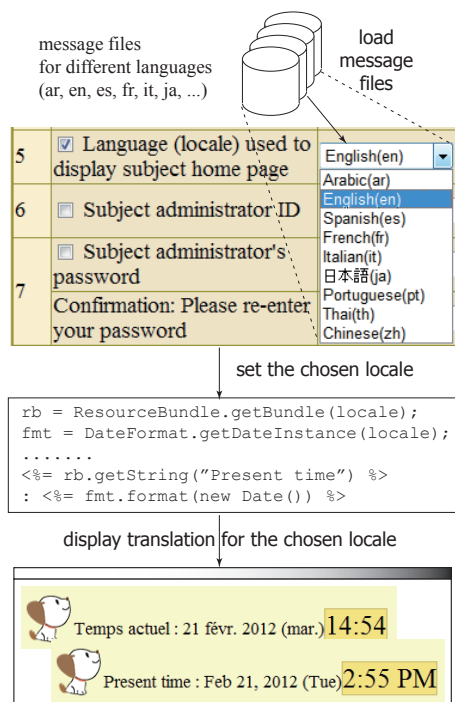


Fig. 4: Extensible locales

it will be displayed when the mouse cursor lingers over the menu item icon. The second line specifies the name of the menu item icon file. The third and fourth lines specify the prefix and suffix HTML code that will be inserted before and after the selected text, respectively. The part \$ will be replaced by the color selected with the mouse such as `rgb(100, 255, 140)`. The fifth line specifies the name of a color palette file. In this way, any kind of edit menu items can be defined, and even color palettes can be extended as shown in Fig. 3 (b).

4.2 Extensible pull-down menus for language selection

Administrators can set up any language in which the Apty home page or subject home pages are displayed. For this purpose, message property resource files for several languages have been prepared. In addition, a message property resource file for a new language can be easily added. If an administrator adds a new message property resource file, a pull-down menu to which the corresponding new language is added will be automatically displayed wherever a language needs to be selected as demonstrated in Fig. 4. When the administrator chooses a language, according to the corresponding locale, a message will be translated and date and time will be displayed in an appropriate format. In addition, the administrator can select any code to be used when subject home messages and feedback messages are uploaded or downloaded.

4.3 Extensible custom links for system enhancement

Administrators' original hyperlinks and buttons can be pasted on some of the main pages to enhance the system. Figure 5 (a) shows how to define a custom button that will be pasted on the top of the bottom frame of a subject home page "trivia". This custom button equips the system with a game named "HTW". The first line specifies the location of the custom button. The following lines specify the HTML code that will be displayed at the specified location of the page, where \$subjectName will be replaced with the corresponding subject name. In this example, a Java applet will be displayed at the top of the bottom frame of the subject home page, which will display a clickable button named "HTW [trivia]". When students click on this button, they can start to play this game as a game of subject "trivia".

Figure 5 (b) illustrates how to define a custom hyperlink that will be pasted in the center of the Apty home page. When students click on this link, the help Web page of the HTW game will be displayed in another window. The first line specifies the location of this custom link. The following lines specify the HTML code that will be displayed at the specified location of the page, where \$locale will be replaced with the selected locale such as ar (Arabic), en (English), es (Spanish), fr (French), and ja (Japanese). The help Web page will be displayed in the language currently selected as described in Section 4.2. Figure 5 (c) illustrates the Apty programming interface. This page is displayed by clicking on another custom link "Apty Tips" and used to access the Apty database and enhance the Apty system.

5. Learning stimulated by collaborative games

The system has been enhanced by a game named HTW (Hunting The World). Anyone can start and join in the game anytime. If some people join in the game at the same time, the first two people will become a pair and start the game. At the same time, multiple pairs can play the game and chat with all the players. During each game, a chance problem will be automatically given. The first player who answers it correctly can get an extra character, which will result in a higher score.

Figure 6 shows the system configuration of the game. The Java applet is running on the client side while communicating with several programs on the server side. When a student clicks on the HTW button, the applet will send a subject name to the server, and receive a port number for later connection. This port number was determined in advance as one of the five vacant ports when the Apty system was installed as described in Section 2. A new window will be opened to display a game board, and a client player thread will start and connect with a player server by using the returned port number. When the student joins in the game,

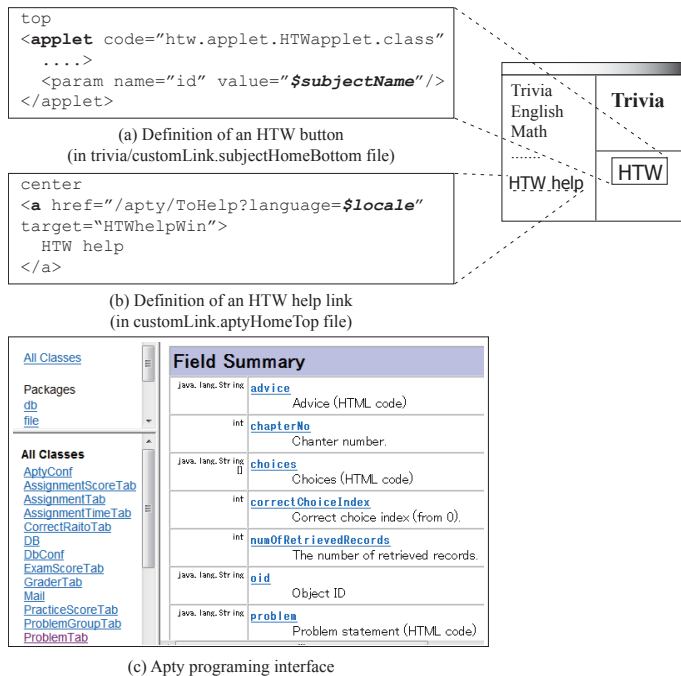


Fig. 5: Definition of a custom link to enhance the system

the player's information will be recorded on the server side. When his or her opponent joins in the game, the game will start. During the game, when the player server informs the client player thread of a chance problem being set, the client player thread will open a Web browser window and send a request to the Chance servlet (server-side program). The Chance servlet will retrieve an appropriate question from the practice problem DB table, and return a response to show the question in a Web page of the browser window. The same question will be shown to the pair of players at the same time. If one player answers the question correctly earlier than the other, he or she will gain an extra character. If a player clicks on the Chat button, a chat window will open, and the client player thread will connect with the Chat server by using the same port. The player can chat with all current players including his or her opponent by making use of all players' information recorded on the server side.

6. Evaluation

6.1 Easy installation with only a password

When a user inserts an Apty DVD to a computer, the Apty installer will start automatically. The user chooses a language and specifies an installation directory, if necessary. What the user has to do is only entering a password as the Apty administrator. Clicking on the "Install" button starts installation. Unlike the conventional systems, this tool automatically sets up appropriate port numbers for Tomcat and Postgresql servers, creates a subject database that contains an example subject Trivia, sets the database password the

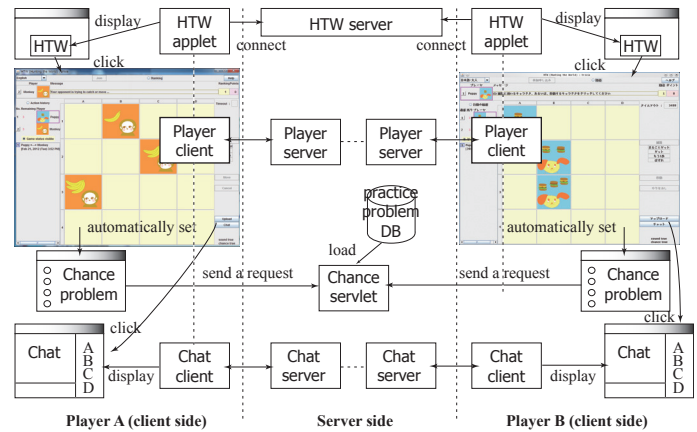


Fig. 6: HTW game structure

user specified, starts the Apty system, and then displays the Apty home page by automatically starting a standard Web browser.

6.2 Open access to subjects

To enable anyone to begin practicing any subject without login, we have arranged a list of all subjects in the left frame of the Apty home page, which is the top page of the system, and arranged the home page of each subject in its right frame as shown in Fig. 1. When a student clicks on a subject name shown in the subject list, the home page of the corresponding subject will be shown in the right frame. In the top frame of the subject home page, students can view subject information and take exams. In the bottom frame of the subject home page, students can view the practice scores. When they click on the "Practice" button displayed in the frame, they can begin to answer questions for practice. By this facility, anyone can view the examples of questions of the subject they are interested in, and come to understand what they can learn from that subject without the registration of their courses, which simplifies the usage of the system.

When a student clicks on the "Ask question" link in the top frame, he or she can ask the teacher in charge of that subject anonymously. As shown in Fig. 1, this system is basically open to anyone. Anyone can access this system to start practicing, see top 10 scores of examinations and practice, ask questions, see feedback messages, and start to play a game. On the other hand, only registered students are allowed to take examinations, review, answer a questionnaire, and see their own scores. At present, the home page of subject Trivia is shown in English. Actually, all messages and images displayed by the Apty system can be localized to any language for any country as described in Section 4. In addition, when text files that consist of the Apty home message, a subject home message, or a subject feedback message are uploaded or downloaded, any code for that country can be applied.

6.3 Collaborative games

A game button "HTW [trivia]" is displayed in the top of the bottom frame in the subject home page "Trivia". To start the game, first, arrange your characters on the board as shown in Fig. 6. A player can catch his or her opponent's characters on the board. You can catch your opponent's character that occupies your empty cell which is one cell away from one of your characters. If you successfully catch your opponent's character, you will get some points and hold a higher ranking. If all your characters are lost, you will lose the game. You can join in the game again and again any time. During the game, a chance problem will be given at random. The chance problems given at random will promote students' motivation of learning. When you click on the Chat button on the board, you can chat with all the players currently joining in the game including the players who have lost the game.

6.4 Experiments

We have been making use of the Apty system in some classes at the university to check the effects of the classes and to promote students' motivation. We here briefly describe the results of these experiments. We prepared 4-choice questions, which were classified into 3 problem groups such as fundamental, intermediate, and advanced. The numbers of the questions were 70, 15, 10, respectively, and 95 in total. We conducted a series of 5 tests (jip1 to jip5). For each test, we set 5 questions, and let the answer time be 20 minutes. For example, for the first test (jip1), 5 questions were given at random among 70 questions of the fundamental problem group. As the class was proceeding further, some of the questions in the intermediate and advanced problem groups were given in the fourth test (jip4) and the fifth test (jip5). For practice, 5 questions were given at random among 60 questions of the fundamental problem group. The answer time was set to 25 minutes, which was a little longer than that of the tests. Anyone was able to begin to answer the questions for practice without login. We administered these 5 tests from the beginning of June to the beginning of July. A little while after the class started, some students gradually appeared who tried to begin practicing. Immediately before the first test was given, a lot of students did practicing again and again. Although we had 102 students in one class, they tried to practice 3790 times in total. We can see that the average scores of practices were gradually being raised.

6.5 Students' motivation

Students can see the top 10 scores and the distribution of scores for the current test and the previous test in a subject homepage at any time. It was possible that some students ranked as the top 10 in the current test even though they did not rank as the top 10 in the previous test. It is because the top 10 scores were shown based on not the total scores but the scores of each small test. The top 10 scores

of practice are also displayed in the bottom frame of the subject homepage in real time. Those scores indicate the rankings in the latest short period. When it comes to a new period (a new month or a new week), it will be possible for anyone to rank as the top 10 or even the top score. Students' names are not displayed in the top 10 scores for practice because anyone can start practicing without login. Instead, the time when a student completed practicing is recorded in seconds. Therefore, if someone ranks as the top 10, they can know that it is actually their own score. Judging from the top 10 scores of practices, it seems that practicing was done during not only the class time but also other times. When we check the top 10 scores and the dates carefully, we can know that there were some students who came to the university to do practicing even on holidays. A histogram of review completion indicates what percent of students completed reviewing the questions they had answered wrongly in the tests. Students can review those questions later again and again. If there are some questions they have not yet reviewed, they can easily check how many those questions are left. We can know that 71 out of 102 students completed reviewing at 91 to 100 %. It seems that almost all students did their best to review the results of these 5 tests.

6.6 Teachers' loads

Teachers can give some assignments to their students. When a student submits a paper for an assignment, an assignment grader will score the submitted paper and can make a comment on it if necessary. A teacher determines the range of scores in advance that a grader can choose, for example, which are 0, 10, 20, 30, 40, and 50 points. In this case, 50 points is a perfect score, which students are informed of in advance. In addition, the teacher can provide a bonus points +10, which makes the highest score 60 points. Graders can select these bonus points if the contents of a submitted paper is excellent. This mechanism enlarged graders' discretion and promoted both of graders' and students' motivation. We were able to make sure that students' ability of understanding and their motivation of learning were promoted in the classes where the Apty system was applied. The Apty system turned out useful and helpful because students studied hard voluntarily through the free practices and professors were able to save some time that would have been required to hold the tests.

7. Conclusion

This paper has presented the extensible learning system Apty with which teachers can easily create questions and assignments through a Web browser, and can automatically administer practices and exams. By introducing some mechanisms such as free practices, bonus points, rankings, and practice-based collaborative games, students' ability of understanding and motivation of learning were promoted. In the future, by introducing the custom links this paper

has presented, we are going to enhance such functions as promoting communications between students and their cooperation in learning.

References

- [1] R. A. Martinez, M. M. Bosch, M. H. P. Herrero, and A. S. Nuno, "Psychopedagogical components and processes in e-learning. lessons from an unsuccessful on-line course," *Computers in Human Behavior*, vol. 23, no. 1, pp. 146–161, 1 2007.
- [2] J. Alty, A. Al-Sharrah, and N. Beacham, "When humans form media and media form humans: An experimental study examining the effects different digital media have on the learning outcomes of students who have different learning styles," *Interacting with Computers*, vol. 18, no. 5, pp. 891–909, 9 2006.
- [3] C. S. Ong and J. Y. Lai, "Gender differences in perceptions and relationships among dominants of e-learning acceptance," *Computers in Human Behavior*, vol. 22, no. 5, pp. 816–829, 9 2006.
- [4] *Blackboard Inc.: Blackboard*. <http://www.blackboard.com/>, 2012.
- [5] M. Y. Yi and Y. Hwang, "Predicting the use of web-based information systems: self-efficacy, enjoyment, learning goal orientation, and the technology acceptance model," *International Journal of Human-Computer Studies*, vol. 59, no. 4, pp. 431–449, 10 2003.
- [6] K. Romanov and A. Nevgi, "Learning outcomes in medical informatics: Comparison of a webct course with ordinary web site learning material," *International Journal of Medical Informatics*, vol. 75, no. 2, pp. 156–162, 2 2006.
- [7] J. Lu, C.-S. Yu, and C. Liu, "Learning style, learning patterns, and learning performance in a webct-based mis course," *Information & Management*, vol. 40, no. 6, pp. 497–507, 6 2003.
- [8] M. Dougiamas, *Moodle*. <http://moodle.org/>, 2012.
- [9] T. Shimomura, *Easy, Enjoyable, Effective E-Learning*. Nova Science Publishers, Inc., 2008.
- [10] T. Shimomura, M. Takahashi, Q. L. Chen, N. S. Lang, and K. Ikeda, "Easy enjoyable effective automated practice and test system," *INTERNATIONAL JOURNAL of EDUCATION AND INFORMATION TECHNOLOGIES*, vol. 1, no. 1, pp. 1–7, 2007.
- [11] T. Shimomura, K. Ikeda, Q. L. Chen, N. S. Lang, and M. Takahashi, "Apty: Easy enjoyable effective e-learning," in *Proceedings of the 7th WSEAS International Conference on Applied Informatics and Communications*, 8 2007, pp. 211–216.

Goal-Based Reengineering of Web Business Applications

Hicham H. Hallal, May Haidar
Fahad Bin Sultan University
Tabuk, Saudi Arabia
{hhallal, mhaidar}@fbsu.edu.sa

Abstract

We present an automated approach to reengineering Web Business Applications from execution traces based on predefined functional goals of the applications. The approach uses model checking to filter the execution traces based on their satisfaction of desired goals and a framework for model inference (implemented using Data Mining algorithms) to infer behavioral models that depict both the data and control flows of an application.

Keywords: Reengineering, Web Business Application, Model Checking, Modeling.

1. Introduction

Advances in Web technologies have turned Web Applications into an integral part of daily life. In particular, automated development techniques have largely contributed to the sharp increase in the production of all sorts of Web Applications including plain informative Web Sites; simple Web Applications, which consist mainly of content pages plus some “shopping” pages; and complex Web Business Applications (WBAs), which are actually information systems deployed over the Internet and linked to sophisticated databases with elaborate user interfaces. Meanwhile, ensuring high quality of the produced applications has remained a concern for designers and developers who have to guarantee features like correctness, high performance, security, and usability. This is true in particular for WBAs, where both personal and financial information of users are constantly at stake.

Over the past two decades, model driven techniques have made the development process of WBA less tedious and error prone and have provided efficient means for the guarantee of quality features in the developed and deployed application. Techniques like model transformation, model based testing, reverse engineering, and property testing and validation are increasingly used in the development and analysis of web applications in general. In particular, behavioral model extraction from deployed applications is of high interest since it facilitates several activities like testing and test case generation, property validation, and model transformation. Actually, a behavioral executable model makes it possible to apply formal methods and

techniques like requirement specification, test derivation, and model checking to automate main development and analysis activities, which adds to the efficiency of the development process. However, the inherent complexity of applying formal methods to industrial scales and the persistent questions about scalability of any relevant approach pose major challenges to researchers and practitioners in the field to optimize the models generated from applications to alleviate the complexity and scalability problems. However, in many cases, an executable model cannot be built from design or requirement documents due to, e.g., the fact that such documents do not exist or are not complete or accessible. Therefore, relying on actual executions to infer models that describe how a system actually behaved becomes a more practical choice. In general, monitoring and logging techniques and tools exist for almost all types of applications. A monitor would record execution traces from the actual use of an application and store them in logfiles. These traces are then used to infer a model of the application. The inferred model can serve to analyze the behavior of the application and improve its design and implementation if needed (Figure 1). The problem with trace based models, however, is twofold. First, the expressiveness of the generated models is always limited by what is observable during the execution of the application. Second, the size of the model can reach limits which affect the scalability of any analysis approach that relies on the generated models.

In this paper, we explore an approach to reengineer customized behavioral models of WBAs based on the actual executions of an application while being accessed by real users or by testers. The reengineered models are customized to depict the intentions of the users of a WBA. By default, each user accesses a WBA to fulfill a specific task: purchase a ticket, book a reservation, buy food, execute a banking transaction, etc. The main idea in the proposed approach is to reengineer models that depict the different intentions (goals) of users who interact with the WBA over a period of time. The behavior of the WBA in response to user stimuli is collected in execution traces observable through monitoring (using a proxy server for example). Then, for each predefined goal a model is reengineered that includes the behavior recorded only in the traces that satisfy the goal.

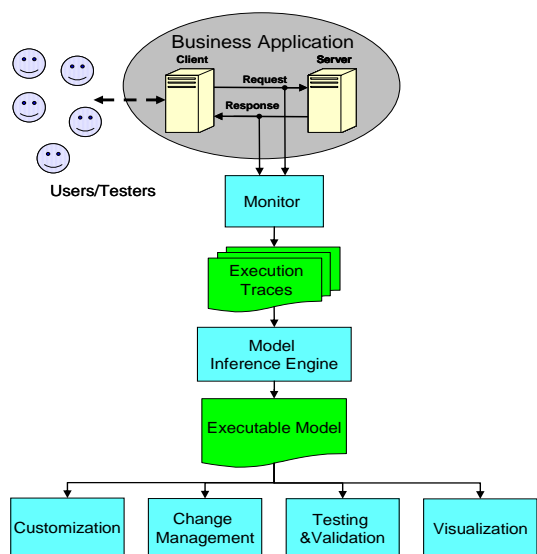


Figure 1. Workflow of the model inference approach.

This paper is organized as follows. In Section 2, we review the related work. In Section 3, we describe WBAs and how to collect execution traces. In Section 4, we introduce a definition of functional goals and discuss the formulation method. In Section 5, we describe the approach to infer behavioural models from traces of WBAs based predefined functional goals. Finally, we conclude the paper in Section 6 and discuss potential extensions of the work.

2. Related Work

Inferring behavioural models of software applications has been the focus of many research efforts over decades, e.g., [2, 7, 8, 10, 15, 16, 18], where models are either inferred mainly from system requirements [9, 15, 16], depicted as scenarios, or extracted from execution traces [2, 7, 8, 10] collected by monitoring. The approach presented in this paper can be compared to the work in [10], [13], and [23]. In [13], a method is proposed to learn HTTP request models for intrusion detection, where the signatures of known attacks are used in enhancing the learning. On the other hand, in [10], a trace recorded during a browsing session is used to infer a model of a web application. The obtained model in [10] consists of communicating automata representing windows and frames of the application, thus resulting in a hierarchical model that describes the control flow of the application, but does not address the data variations that are revealed by traces collected in different browsing sessions. In [23], the focus is mainly on predicting simple low level intentions of users of applications based on the features extracted from the user interaction such as user's typed sentences and viewed content. The work does not consider high level goals and structured intentions that relate more to the functionality of the application.

This work builds on the results obtained in [8] in the sense that we reuse the formal framework for model inference, which is capable of inferring models that depict both the data and control flows of a WBA. The implementation of the framework was completed using data mining algorithms applied to random sets of traces generated by actual use of the WBA. Here, we do not consider random traces to generate the behavioral model of a WBA. Instead, we filter traces before using them based on satisfying a pre-defined goal specified as a property tested on the trace. If the trace satisfies the goal, then it is added to the model. Otherwise, it is ignored. The objective is to customize the model and reduce its size to make it more useful in automation of specific tasks like property testing, test derivation and test case generation.

3. Web Business Applications

In general, WBAs are developed in three tiers: a user interface, a server side, and a database at the backend. The database part is usually masked by the server programs which perform the storage and retrieval operations on behalf of the user. Meanwhile, the user and server components are commonly linked through the client-server architecture, where the client can be any typical Web browser. In terms of behaviour, a WBA involves two levels of logic: the application logic, which manipulates the inner part of the application like the interface and the protocols that control it, and the business logic, which is usually seen as the part of the code that controls the data in the application and determines how it is treated based on the user interactions.

Traces of WBAs

To generate traces of WBAs, we adopt a monitoring approach that relies on a HTTP proxy, which intercepts the communications between the server and the client and stores them in execution traces (logfiles). A proxy yields traces that contain only information exchanged between the client and the server during a monitoring session. In other words, any information handled, e.g., variables assigned or scripts executed, on the server and client side cannot be obtained by a proxy and is absent from the collected traces. In addition, we do not consider the interactions between the server and the database, which might actually reside on the same machine. At the same time, we do not consider information related to stateful HTTP communications like cookies, server side sessions, and hidden variables.

Following [8] and [10], we define a WBA trace recorded by a proxy as a set of ordered HTTP request/response pairs exchanged between the client and the server as a result of a sequence of user actions. Each request sent by the client is a link clicked or a form filled on the source

page by the user while the response is an HTML document returned by the server identifying the destination page or frame in a page when frames are used. As a result, a request/response pair may identify one page or frame in the WBA. The proxy identifies the pages of the WBA either directly by their names or by parameters in the request if a generic name is used for all the pages. For simplicity, we assume that each page of an application contains at least one form which is used to make the transition to other pages. Moreover, to identify a page, we use the URI/URL of the page along with a subset of the page attributes, i.e., fields used in HTML forms that are submitted by the client to the server.

Finally, we consider only single window WBAs with non-framed pages and assume that all traces recorded from an application start in its home page.

4. Goals of WBAs

In general, a user interacts with a WBA with a purpose in mind. It basically depends on the type of the application and the functionality it offers. For example, we consider a WBA that has been used in our previous work [8] which consists of small flight reservation system, where customers can use the application to buy tickets and make reservations with different preferences. Hence, a user accessing the flight reservation application would, most probably, want to buy an airplane ticket to travel from one place to another. In this case, the purpose is to “buy a ticket”. This can be identified as the goal of the user in his access to the WBA. By default, achieving the mentioned goal involves completing smaller tasks before actually “buying the ticket” through a confirmation issued by the WBA in the form of receipt, SMS message to a mobile phone number or through an email message. These smaller tasks might involve entering personal information of the passenger for whom the ticket is being bought, source destination information, financial and credit card information, and finally consent for purchase and payment. This means the bigger goal of “buying the ticket” is broken down into smaller goals that are not necessarily an expression of the functionality of the WBA.

This reasoning about goal definition and classification is similar to the work in [22] where the intentions of the user of a web application are classified into two types:

1. Action intentions, which are perceived on a low level. Each action can be a mouse click, keyboard typing, or any other basic action performed on a computer.
2. Semantic intentions, which correspond to what the user wants to achieve at high level. A semantic intention may involve several basic actions on a computer to accomplish it.

Our reasoning is also similar to the reasoning made in the field of automated planning, where hierarchical decomposition of goals is considered to devise and

implement proper plans especially in the presence of contingencies.

We consider the following classification of goals in a WBA:

1. Non functional goals: They relate to completing low level tasks in the WBA. The completed tasks do not need to satisfy a functional requirement of the application. They include tasks like filling personal information on a page, entering login information, navigating from one page to another using various controls (buttons, links, form submissions, etc).
2. Functional goals: They relate directly to satisfying a functional requirement of the WBA. Examples include buying a ticket, reserving a hotel room, buying a book, etc. Each functional goal is achieved through the completion of at least one non functional goal. In other words, each functional goal can be broken down into a sequence of one or more non functional goals that should be achieved in a certain order (usually defined by the developers of the application).

In this paper, we focus on functional goals and describe how to use them in reengineering customized models of the WBA with respect to the various goals that can be achieved when using the application. We describe in the following section the model checking based approach where traces from a WBA that satisfy a specific pre-defined goal can be used to infer a behavioural model of the application.

5. Goal-based Modeling of WBAs

In this section, we describe the goal based modeling of WBAs. The proposed approach is an extension of the inference approach presented in [8]. Figure 2 shows the modifications needed to make the approach goal-based, i.e., modeling is based on knowing the goals of the users of the application under test. A new step in the workflow is added to filter out traces that do not satisfy the desired goal. Trace filtering based on goal satisfaction is detailed in Section 5.1.

5.1 Trace Filtering

Each recorded trace of the WBA is checked against the representation of a goal for satisfaction. The check is performed in the model checker Spin [11], where the trace is modeled by a PROMELA process and the desired functional goal is specified using the Linear Temporal Logic (LTL) formalism. Notice that since the trace of a WBA is a sequence of pages interleaved with HTTP requests, the check for goal satisfaction can be performed more easily through a simple search to match the goal. However, we choose to use model checking in order to keep the approach more generic and capable to treat more sophisticated traces where a total order between

components of a trace is not always present. In some cases, such as in [7], the traces collected from a system under test can be partially ordered sets of events and the simple search to match the goal becomes insufficient.

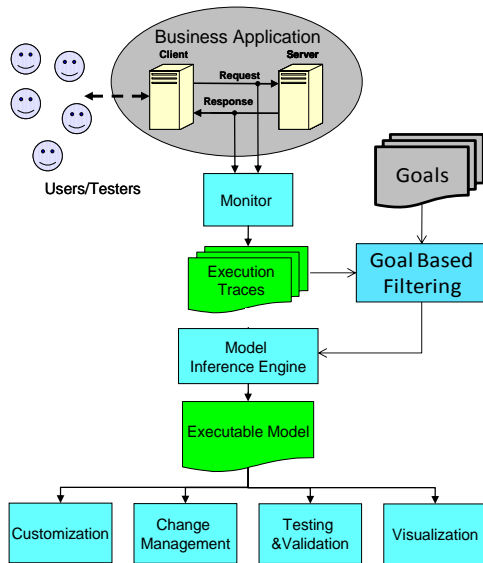


Figure 2. Modified model inference approach.

5.2 Goal Specification

LTL is the main language for property specification in the model checker Spin [11]. Other forms of specification like automata (never claims) are possible but not considered in this work. Following the discussion in Section 4, we consider that each functional goal is a sequence of smaller non-functional ones. As an example, consider the function goal “purchase a ticket for a minor” that is adopted with minor modifications from [8]. Such goal involves several steps including:

1. Providing the name of the passenger
2. Providing the age
3. Providing the name of the guardian traveling with the child
4. Providing the source and destination information
5. Providing seating preferences
6. Providing payment information
7. Confirming purchase

However, formulation of the goal does not require an LTL formula that involves all the non-functional steps. Instead, one can choose key steps to use as indicators of the goal. For example, “purchasing a ticket for a minor” must always involve the two non-functional goals: “providing guardian information” and “confirming the purchase”. For simplicity, we denote the functional goal *A* and the two non-functional goals *b* (for providing guardian information) and *c* (for confirming purchase). In addition to the decomposition of *A* into *b* and *c*, we know that *b* must always come before *c*.

In LTL, this specification of *A* can be expressed in several forms. For illustration purposes, we show one of the simplest forms:

$$!c \text{ U } b, \text{ which reads: NOT } c \text{ UNTIL } b.$$

This means that *c* does not happen until after *b*.



Figure 3. Single trace satisfying ticket purchasing goal.

On the other hand, a single trace that features purchasing a ticket for a child is visualized in Figure 3. The trace is represented as an automaton, where states represent the pages visited in the WBA to fulfill the goal and the transitions are the transitions between the pages. Consequently, the main assumption on the applicability of the modeling approach is to be able to identify the states of the process representing a trace of the application.

After selecting the trace to be added to the model, the existing framework for model inference [8] is used to deduce a model based on the filtered trace and any traces chosen previously.

Figure 4 shows the model of the WBA corresponding to the goal *A* generated from 200 traces. The presented model includes, in addition to states and transition, the conditions on the data submitted to the WBA in order to reach the goal. Notice that even though some customers were interested in buying a ticket for a child, they entered wrong information that led them into rejection. This is due to the formulation of the goal itself which states the confirmation has to come after setting a guardian while rejection is reached directly from reservation. This example shows that proper definition of the goals is the key to obtain the optimal model for the

WBA under test. On the other hand, the presented model shows how the behavior of the application from all the processed traces (in this case 200 traces) is aggregated in a single state diagram. While some traces would contribute new states and transitions to the model, other traces might contribute only new conditions on the transitions between states.

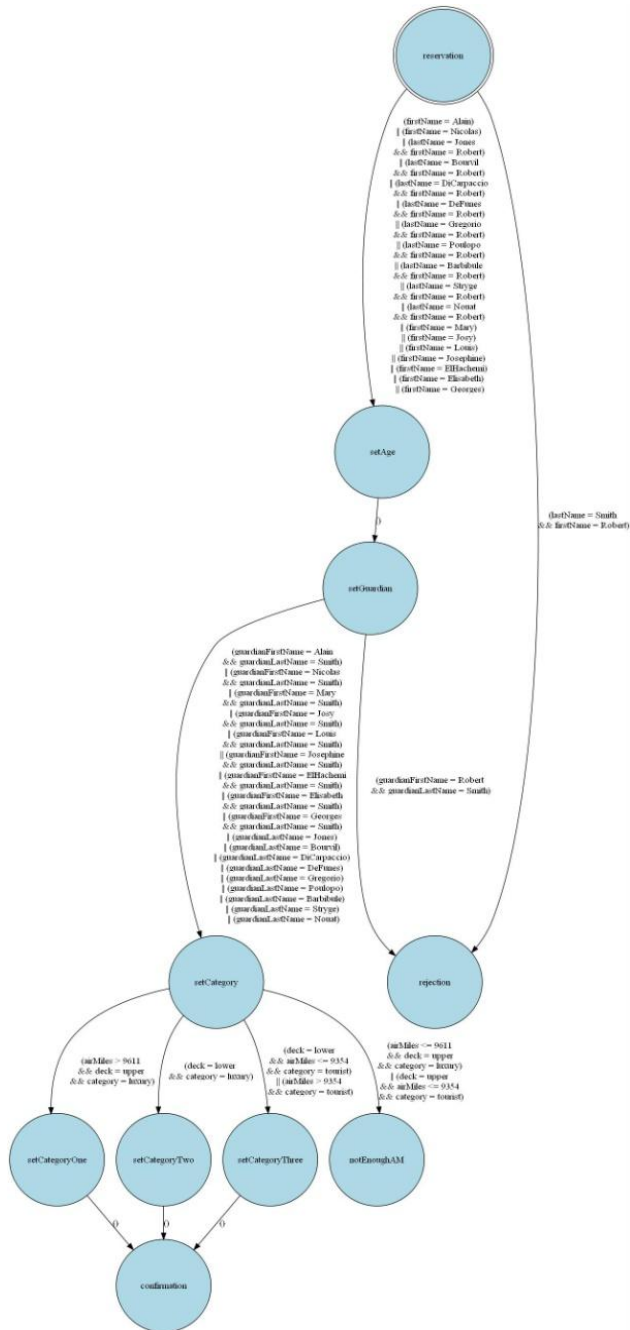


Figure 4. Model of the WBA for the goal “purchase a ticket for a child” (200 traces).

After selecting the trace to be added to the model, the existing framework for model inference [8] is used to deduce a model based on the filtered trace and any traces chosen previously.

Figure 4 shows the model of the WBA corresponding to the goal A generated from 200 traces. The presented model includes, in addition to states and transition, the conditions on the data submitted to the WBA in order to reach the goal. Notice that even though some customers were interested in buying a ticket for a child, they entered wrong information that led them into rejection. This is due to the formulation of the goal itself which states the confirmation has to come after setting a guardian while rejection is reached directly from reservation. This example shows that proper definition of the goals is the key to obtain the optimal model for the WBA under test. On the other hand, the presented model shows how the behavior of the application from all the processed traces (in this case 200 traces) is aggregated in a single state diagram. While some traces would contribute new states and transitions to the model, other traces might contribute only new conditions on the transitions between states.

Another example of a model generated from the WBA, we show, in Figure 5, a model corresponding to another type of goals. This time we model the behavior of the users who try to detect the names that are on the no fly list used in the application. Such model can be used for security purposes.

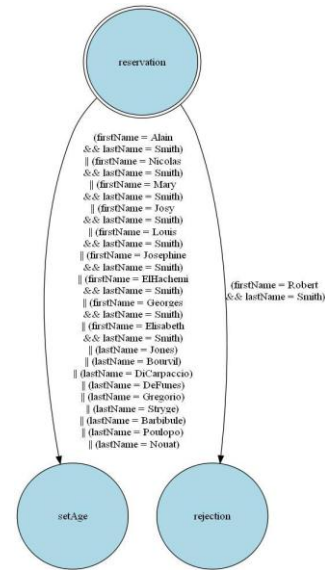


Figure 5. Model of the WBA for the goal “identify no fly list” (100 traces).

6. Conclusion

We presented an approach to reengineer WBAs through inference of behavioral models from execution traces of the applications under test. The proposed approach relies on identifying functional goals of an application and filtering execution traces based on satisfaction of selected goals. The inference is performed in the framework for model inference already presented in [8] and implemented using Data Mining algorithms. The

inferred models, which depict the data and control flows of an application, can be useful in various development activities like testing and validation. Being goal based, the models are customized to reduce the complexity of handling them in automated environments. In addition, goal-based models of WBAs can have practical uses in security analysis of such applications.

The proposed approach can be extended by considering combining multiple models to infer a global model of an application which offers more coverage of the behavior of the application. In addition, multiple goals that are satisfied in collected traces can themselves be used to reengineer partial specifications of the original application.

Acknowledgment. We thank the insightful contributions from Prof. Alexandre Petrenko and Arnaud Dury.

References

- [1] Angluin, "Learning regular sets from queries and counterexamples". *Information and Computation*, 1987, v.75 n.2, pp.87-106.
- [2] T. Berg, B. Jonsson, and H. Raffelt, "Regular Inference for State Machines with Parameters". In *Fundamental Approaches to Software Engineering (FASE 2006)*, LNCS 3922:107–121, March 2006.
- [3] J. Chandra, R. P. Bose, S. H. Srinivasan, "Data Mining Approaches to Software Fault Diagnosis". In *15th International Workshop on Research Issues in Data Engineering: Stream Data Mining and Applications (RIDE-SDMA'05)*, pp. 45-52. 2005.
- [4] E. M. Clarke, O. Grumberg, D. A. Peled, *Model Checking*. The MIT Press, 2000.
- [5] T. Denmat, M. Ducassé, O. Ridoux, "Data Mining and Cross-Checking of Execution Traces: A Re-Interpretation of Jones, Harrold and Stasko Test Information". In *20th IEEE/ACM international Conference on Automated Software Engineering*, 2005, pp. 396-399.
- [6] Graphviz: <http://www.graphviz.org>.
- [7] H. Hallal, S. Boroday, A. Petrenko, A. Ulrich, "A Formal Approach to Property Testing in Causally Consistent Distributed Traces". In *Formal Aspects of Computing*, 2006, 18(1): 63-83.
- [8] H. Hallal, A. Dury, A. Petrenko, "Web-FIM: Automated Framework for the Inference of Business Software Models". *Services2009 Competition Conference Associated with ICWS 2009 International Conference on Web Services*, Los Angeles, USA, July, 2009.
- [9] D. Harel, H. Kugler, and A. Pnueli, "Synthesis Revisited: Generating Statechart Models from Scenario-based Requirements". In *Formal Methods in Software and Systems Modeling*, 2005.
- [10] Haydar, A. Petrenko, H. Sahraoui, "Formal Verification of Web Applications Modeled by Communicating Automata" In *24th IFIP WG 6.1 IFIP International Conference on Formal Techniques for Networked and Distributed Systems (FORTE 2004)*, Spain, 2004, pp. 115-132.
- [11] G. J. Holzmann, *The SPIN Model Checker*. Addison-Wesley. 2003.
- [12] J. Hopcroft, R. Motwani, J. Ullman, *Introduction to Automata Theory, Languages, and Computation*. Third Edition Addison-Wesley. 2006.
- [13] K. Ingham, A. Somayaji, J. Burge, S. Forrest. "Learning DFA Representations of HTTP for Protecting Web Applications". In *Computer Networks Journal*, 2007, 51, pp.1239-1255.
- [14] B. Jobstmann and R. Bloem, "Optimizations for LTL Synthesis". In *FMCAD '06: Proceedings of the Formal Methods in Computer Aided Design*, San Jose, 2006.
- [15] Kazhamiakin, M. Pistore, and M. Roveri, "Formal Verification of Requirements Using Spin: A Case Study on Web Services". In *SEFM'04: Proceedings of the Software Engineering and Formal Methods*, 2004, pp. 406–415.
- [16] Kruger, R. Grosu, P. Scholz, and M. Broy, "From MSCs to Statecharts". In *Distributed and Parallel Embedded Systems, DIPES'98*. Kluwer Academic Publishers, 1999, pp. 61–71.
- [17] D. Lo, S. Khoo, C. Liu, "Mining Temporal Rules from Program Execution Traces". In *3rd International Workshop on Program Comprehension through Dynamic Analysis (PCODA'07)*. Vancouver, Canada. 2007.
- [18] L. Mariani, M. Pezzè. "Dynamic Detection of COTS Component Incompatibility". In *IEEE Software* 24(5): 76-85 (2007).
- [19] J. R. Quinlan, *C4.5: Programs for Machine Learning*. Kaufmann, 1992.
- [20] C. E. Shannon, "A Mathematical Theory of Communication", *Bell System Technical Journal*, 1948, 27, pp. 379–423 & 623–656.
- [21] Weka: <http://www.cs.waikato.ac.nz/ml/weka/>
- [22] Web Application Security Consortium Glossary, <http://www.webappsec.org/projects/glossary/>
- [23] C. ZHENG, L. Fan, H. LIU, Y. LIU, W. MA, L. WENYIN User Intention Modeling in Web Applications Using Data Mining World Wide Web: Internet and Web Information Systems, 5, 181–191, Kluwer Academic, 2002.

Web Application Vulnerabilities and Detection

Kangan(Student)¹ and Monika(Assistant Professor)²

¹U.I.E.T, Panjab University, Chandigarh, U.T, India

²U.I.E.T, Panjab University, Chandigarh, U.T, India

Abstract - Web applications cover a range of activities, such as e-banking, webmail, online shopping, community websites, blogs, vlogs, network monitoring and bulletin boards. Web security testing verifies whether Web based applications are vulnerable or secure when they are subjected to malicious input data. This paper presents taxonomy of various security testing techniques and mainstream software tools used for a particular type of security testing. We also provide the brief description of various types of attacks and the security testing tools used to detect these attacks.

Keywords: web application security; security testing tools; types of security testing; security risks;

1 Introduction

Software testing is any activity aimed at evaluating an attribute or capability of a program or system and determining that it meets its required results [1]. Although crucial to software quality and widely deployed by programmers and testers, software testing still remains an art, due to limited understanding of the principles of software. The purpose of testing can be quality assurance, verification and validation, or reliability estimation [2]. There is a plethora of testing techniques [3] as shown in figure 1.

Today, software becomes more complicated and large-scale, which results in more software security problems increasingly. Software security is the ability of software to provide required function when it is attacked [4]. Security testing of any developed system is all about finding out all the potential loopholes and weaknesses of the system, which might result into loss/theft of highly sensitive information or destruction of the system by an intruder/outsider. Security Testing helps in finding out all the possible vulnerabilities of the system and help developers in fixing those problems.

Now a day, almost all organizations across the world are equipped with hundreds of computers connected to each other through intranets and various types of LANs inside the organization itself and through Internet with the outer world and are also equipped with data storage & handling devices. The information that is stored in these storage devices and the applications that run on the computers are highly important to the organization from the business, security and survival point of view.

Any organization small or big in size, need to secure the information it possesses and the applications it uses in order to protect its customer's information safe and suppress any possible loss of its business.

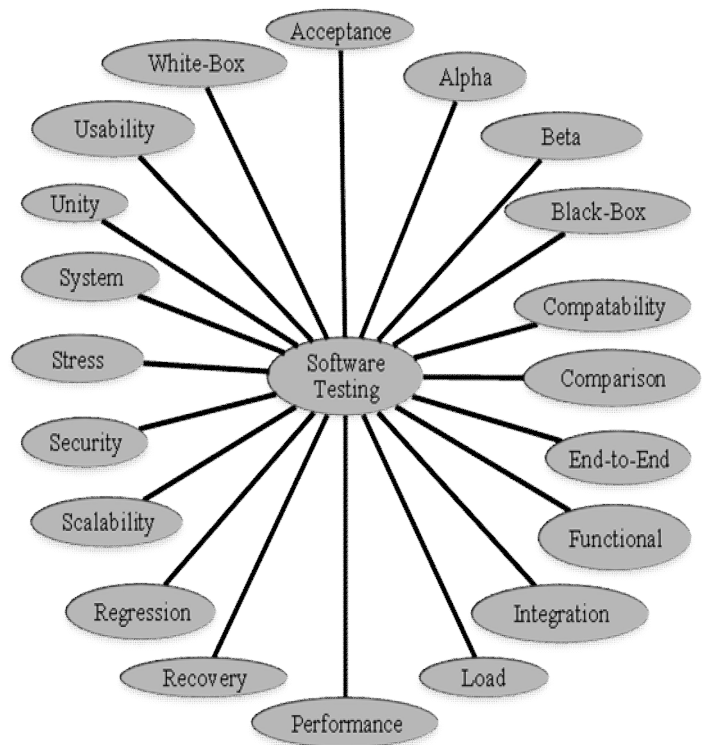


Figure 1. Types of Security Testing

Security testing ensures that the systems and applications used by the organizations are secure and not vulnerable to any type of attack. Finding and fixing security flaws in a legacy web application typically requires detailed knowledge of its behavior. This knowledge is a result of understanding high-level design artifacts combined with an analysis of the source code of the web application. However, it is well known that manual effort spent towards analysis of the source code is labor and cost-intensive and is often error-prone.

Additionally, design level artifacts are often unavailable for legacy web applications and the only available resource is the source code. While source code is the most accurate

description of the behavior of a web application, this description is expressed in low-level program statements. Due to its inherent low-level nature, source code does not readily over a high-level understanding of an application's intended behavior which is necessary to identify and fix security flaws. So, there arises the need for security testing of web application.

Software testing is the process that determines that confidential data stays confidential and users can perform only those tasks that they are authorized to perform.

In the next section we have describes the needs of security testing. Section III presents the different software for various types of security testing. Section IV presents various security risks. Section V provides the security testing tool for each type of security risk. Section VI concludes by highlighting research opportunities resultant from this work.

2 Needs of Security Testing

Exposing systems to the internet increases the risk that security weaknesses in those systems will be leveraged to compromise the system or the underlying data. So, there arises the need for security testing of web application. The various needs of security testing are [10]:

- Security test helps in finding out loopholes that can cause loss of important information and allow any intruder enter into the systems.
- Security Testing helps in improving the current system and also helps in ensuring that the system will work for longer time .
- Security Testing doesn't only include conformance of resistance of the systems your organization uses, it also ensures that people in your organization understand and obey security policies. Hence adding up to the organization-wide security.
- If involved right from the first phase of system development life cycle, security testing can help in eliminating the flaws. This is beneficial to the organization almost in all aspects (financially, security and even efforts point of view).

3 Needs of Security Testing

The purpose of security testing is to identify the vulnerabilities and subsequently repairing them. It ensures that the systems and applications used by the organizations are secure and not vulnerable to any type of attack. Typically, security testing is conducted after the system has been developed, installed and is operational. Security testing can be further classified into various types as shown in table I [10]. It presents the types of security testing, faults detected by them and various tools used for each type of security testing.

TABLE I. TYPES OF SECURITY TESTING

Type of Security Testing	Fault Detected	Mainstream Commercial Software Tools used
Security Scanning	•Detect areas of vulnerability in the OS, applications and network	Nessus (software), ISS
Network Scanning	•Detect active devices • Detect open ports and associated services/ application	Amap, AutoScan, Netdiscover, Nmap, POf, Umit etc
Vulnerability Scanning	•Detects hosts and open ports •Detect known vulnerabilities	Firewalk, GFI LANguard, Hydra, Metasploit, Nmap, Paros Proxy etc
Password Cracking	• Detect weak passwords and password policies	Hydra, John the Ripper, Rcrack, SIPcrack, SIPdump, TFTPBrute, THC etc
Log Review	•Provides historical information on system use, configuration, and modification •Could reveal potential problems and policy deviations	Snort IDS sensor
File Integrity Checkers	•Detect changes to important files; •Detect certain forms of unwanted files, such as well-known attacker tools	Autopsy, Foremost, RootkitHunter, and Sleuthki
Anti-Virus and Malicious Code Detection	•Prevent attacks	avast! Free Antivirus, Veracode
Penetration Testing	• Tests security using the same methodologies and tools that attackers employ •Verifies vulnerabilities	Driftnet, Dsniff, Ettercap, Kismet, Metasploit, Nmap, Ntop, SinFP, SMB Sniffer, and Wireshar
Modem Security/ war dialing	• Detect the use of unauthorized modems that might be used to bypass existing security measures.	Toneloc (freeware), THC-SCAN (freeware), SecureLogix Telesweep Secure (commercial)
Risk Assessment	•Find out and prepare possible backup-plan for any type of potential risk	CRAMM, CORA, COBRA, Risk Check, RiskPAC,
Ethical Hacking	•Detect potential security weaknesses for a client	Cain and abel, Legion, Brutus, Ec-Council
Security Auditing	•Detect common security /config errors	LSAT, Flawfinder, RATS

4 Security Risks

In recent years the development of such applications has been considerable, and today rich internet applications offer complex, real-time interactions with users. For instance, web operating systems such as eyeOS offer much functionality that was previously available only with traditional operating

systems. While web applications have become ubiquitous, they also present new security risks. It is important to identify and understand these risks when developing, hosting or simply using these applications [5]. There are two main reasons that web applications are vulnerable to attack.

First, it is generally difficult for the service manager to keep up to date with security patches. This is a common issue for services in general, but it may be particularly challenging for web applications. This could be improved by better design and packaging but it is often impossible to upgrade web applications automatically.

Second, web applications are often easy targets for attackers. As a relatively recent development, they use non-mature code compared to traditional network services. Unfortunately exploits – malicious code that exploits software vulnerabilities – are generally easy to prepare, remotely executable, cross-platform, and require no compilation. This helps attackers to design effective and scalable automated attacks. Vulnerable installations can be found quickly, easily and silently by using search engines to detect known vulnerable patterns, generally filenames, of specific web applications. In table 2 we focused on identifying the most common vulnerabilities.

In our study we get details for each of the Open Web Application Security Project Top Ten 2011 [6] vulnerability. Table II shows the variations in top 10 security risks from 2007 to 2011.

TABLE II. TOP 10 SECURITY RISKS

Risk Level	Top 10 Security Risks in	
	2007	Till 2011
1	Cross Site Scripting (XSS)	Injection
2	Injection Flaws	Cross-Site Scripting (XSS)
3	Malicious File Execution	Broken Authentication and Session Management
4	Insecure Direct Object Reference	Insecure Direct Object References
5	Cross Site Request Forgery (CSRF)	Cross-Site Request Forgery (CSRF)
6	Information Leakage and Improper Error Handling	Security Misconfiguration(NEW)
7	Broken Authentication and Session Management	Insecure Cryptographic Storage
8	Insecure Cryptographic Storage	Failure to Restrict URL Access
9	Insecure Communications	Insufficient Transport Layer Protection
10	Failure to Restrict URL Access	UnvalidatedRedirects and Forwards (NEW)

Added risks are Security Misconfiguration and UnvalidatedRedirects and Forwards. Security Misconfiguration issue was A10 in the Top 10 from 2004, but was dropped in 2007 because it wasn't considered to be a software issue. However, from an organizational risk and prevalence perspective, it clearly merits re-inclusion in the Top 10; so now it's back. UnvalidatedRedirects and Forwards issue is making its debut in the Top 10. The evidence shows that this relatively unknown issue is widespread and can cause significant damage.

Removed risks are Malicious File Execution and Information Leakage and Improper Error Handling. Malicious File Executions is still a significant problem in many different environments. However, its prevalence in 2007 [9] was inflated by large numbers of PHP applications having this problem. PHP now ships with a more secure configuration by default, lowering the prevalence of this problem. Information Leakage and Improper Error Handling issue is extremely prevalent, but the impact of disclosing stack trace and error message information is typically minimal.

5 Security Testing Tools

Many web application security vulnerabilities result from generic input validation problems. Examples of such vulnerabilities are SQL injection and Cross-Site Scripting (XSS). Although the majority of web vulnerabilities are easy to understand and to avoid, many web developers are, unfortunately, not security-aware[7]. As a result, there exist a large number of vulnerable applications and web sites on the web.

A web application scanner is an automated security testing that examines web applications for security vulnerabilities. In addition to searching for web application specific vulnerabilities, the tools also look for software coding errors, such as illegal input strings and buffer overflows [8]. Most of these scanners are commercial tools (e.g., Acunetix Web Vulnerability Scanner and HP WebInspect), but there are also some free application scanners (e.g., Foundstone WSDigger and wsfuzzer) with limited use.

A method is method to evaluate and benchmark automatic web vulnerability scanners using software fault injection techniques [8]. Software faults are injected in the application code and the web vulnerability- scanning tool under evaluation is executed, showing their strengths and weaknesses concerning coverage of vulnerability detection and false positives. However, this study was focused on a various security testing tools or vulnerability scanner for a particular type of tool. In table III various security testing tools [12] used to detect OWA SP Top Ten attacks are presents and how these attacks occur. Security testing tools presented in table III are both open source and commercially available tools.

TABLE III. TYPES OF ATTACKS AND TOOLS USED FOR THEM

Type of Attack	How they occur	Security Testing Tools
1. SQL Injection[15]	SQL commands are injected into data-plane input in order to affect the execution of predefined SQL commands	ZAP, Francois Larouche, Antonio Parata, icesurfer, ilo
2. Reflected Cross-Site Scripting (XSS) [13]	In this attack doesn't load with the vulnerable web application but is originated by the victim loading the offending URL	XSS-Proxy, ratproxy, Burp Proxy, WebScarab, PHP Charset Encoder(PCE)
3. Stored Cross Site Scripting (XSS)[17]	Occurs when a web application gathers input from a user which might be malicious, and then stores that input in a data store for later use	PHP Charset Encoder(PCE) , Hackvector, BeEF, XSS-Proxy, Backframe, Burp, WebScarab
4. DOM-based Cross-Site Scripting [16]	Occur when active content, such as a JavaScript function, is modified by a specially crafted request such that a DOM element that can be controlled by an attacker.	The DOMinator Tool, DOM XSS Wiki, DOM Snitch
5. Broken Authentication & Session Management Testing [12]	Attacker uses leaks or flaws in the authentication or session management functions (e.g., exposed accounts, passwords, session IDs) to impersonate users.	HackBar
6. Insecure Direct Object References [12]	Occurs when a developer exposes a reference to an internal implementation object, such as a file, directory, database record, or key, as a URL or form parameter.	Burp Suite
7. Cross-Site Request Forgery (CSRF) [12]	Occurs when attacker link or script in a page that accesses a site to which the user is known (or is supposed) to have been authenticated.	Tamper Data, monkeyfist
8. Security Misconfiguration [14]	Occur when the system admins, DBAs, and developers leave security holes in the configuration.	Watobo, Microsoft Baseline Security Analyzer
9. Insecure Cryptographic Storage [12]	Occur when web applications do not properly protect sensitive data, such as credit cards, SSNs, and authentication credentials, with appropriate encryption or hashing.	N/A
10. Failure to Restrict URL Access [12]	Frequently, an application only protects sensitive functionality by preventing the display of links or URLs to unauthorized users. Attackers can use this weakness to access and perform unauthorized operations by accessing those URLs directly.	Nikto/Wikto
11. Insufficient Transport Layer Protection [12]	When Applications frequently fail to authenticate, encrypt, and protect the confidentiality and integrity of sensitive network traffic	Calomel
12. UnvalidatedRedirects and Forwards [12]	Common website functions, such as search results or account logins, frequently use redirects or forwards to send visitors to another destination. If the website doesn't verify the destination, redirects or forwards might be vulnerable.	Watcher

6 Conclusion

In this paper we have provided a review of various security risks to which web applications are vulnerable. A brief description of which type of security testing tool should be used for a particular type of attack is given. Various studies that are given in this paper show that the effectiveness of security testing tools in detection of vulnerabilities varies a lot. So researchers are still focusing on this major issue so as to make the web application more secure.

7 References

- [1] Hetzel, William C., The Complete Guide to Software Testing, 2nd ed. Publication info: Wellesley, Mass : QED Information Sciences, 1988.
- [2] Jiantao Pan, 18-849b Dependable Embedded Systems,"Software Testing",Spring1999.
- [3] Types of security testing , Available : <http://www.softwaretestingnow.com/types-of-software-testing>
- [4] Gary McGraw, Bruce Potter. "Software Security Testing"[J]. IEEE Security & Privacy, 2004, 2(5):81-85.
- [5] OWASP Top Ten Project. Open Web Application Security Project. [Online]. Available: http://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project
- [6] Romain Wartel. Security Risks. [Online]. Available: <http://cern.ch/security>.
- [7] Jan-Min Chen; Chia-Lun Wu; Dept. of Inf. Manage., Yu Da Univ., Miaoli, Taiwan, "An Automated Vulnerability Scanner for Injection Attack Based on Injection Point" . In Computer Symposium (ICS), Dec 2010 .
- [8] Elizabeth Fong; Vadim Okun, "Web Application Scanners: Definitions and Functions". In System Science, 40th Annual Hawaii International Conference on Jan 2007.
- [9] Security risks 2007, Available: https://www.owasp.org/index.php/Top_10_2007
- [10] Security Testing, Available: <http://www.buzzle.com/editorials/7-14-2006-102344.asp>.
- [11] Security risks 2010, Available: https://www.owasp.org/index.php/Top_10_2010-Main.
- [12] Security Testing Tools, Available: <http://resources.infosecinstitute.com/owasp-top-10-tools-and-tactics>.
- [13] Cross-Site Scripting, Available: <http://www.opensource-testing.org/security.php>.
- [14] Securitymisconfigurations, Available: <http://www.makeuseof.com/tag/test-computer-securitymisconfigurations-microsoft-baseline-security-analyzer>.
- [15] Tesing SQL Injection, Available: [https://www.owasp.org/index.php/Testing_for_SQL_Injection_\(OWASP-DV-005\)](https://www.owasp.org/index.php/Testing_for_SQL_Injection_(OWASP-DV-005)).
- [16] DOM-based Cross-Site Scripting, Available: https://www.owasp.org/index.php/DOM_Based_XSS.
- [17] Cross site scripting tools, Available: https://www.owasp.org/index.php/Testing_for_Cross_site_scripting.

Verifying the Behavioral Contracts among Components by means of Semantic Web Techniques

Francisco-Edgar Castillo-Barrera
Engineering Faculty
Universidad Autónoma de San Luis Potosí
Dr. Manuel Nava 8, Zona Universitaria poniente
78290 San Luis Potosí, México
ecastillo@uaslp.mx

Carolina Medina-Ramírez
Department of Electrical Engineering
Universidad Autónoma Metropolitana, México
Av San Rafael Atlixco 186, Col. Vicentina
09340 Distrito Federal, México
cmed@xanum.uam.mx

Héctor A. Durán-Limón
Department of Information Technologies
Universidad de Guadalajara
Periférico Norte 799, Mdulo L-308
45100, Zapopan, México
hduran@cucea.udg.mx

Jose Emilio Labra Gayo
Department of Computer Science
Universidad de Oviedo
C/Valdes Salas s/n 33007-Oviedo, España
labra@uniovi.es

S. Masoud Sadjadi
School of Computing and Information Sciences
Florida International University (FIU)
11200 SW 8th St
Miami, USA
sadjadi@cs.fiu.edu

Abstract

Verification and validation of Component-Based systems are important tasks to do during the whole component life-cycle. In companies, verification techniques for component matching are difficult to integrate into the standard software development process because these can be time-consuming, error-prone, and require specialized expertise. In this paper we describe a semantic web framework for verifying behavioral contracts: invariants, pre- and post-conditions. In addition, we use the CORBA-IDL vocabulary with semantics for this purpose. Our approach relies on a core ontology of software components and SPARQL queries. The ontology captures the concepts, properties, relationships, requirements, and software component functionality. This is encoded using OWL DL, supported by the Pellet reasoner for checking the ontology component consistency. The Resource Description Framework (RDF) triples (describing components content in the form of subject-predicate-object expressions) are queried using SPARQL, in order to complement the matching verification process. We use case exam-

ple and a prototype (a semantic framework called Chichen Itza) to show the feasibility of our approach.

1. Introduction

Crnkovic and Larsson [11] define Component-Based Software Engineering (CBSE) "as an approach to software development that relies on software reuse". The goal of CBSE is the rapid assembly of complex software systems using pre-fabricated software components. In order to achieve this aim, methods for verifying the matching among components are necessary. Such methods can be basically classified in Formal, Semi-Formal, and Informal methods. Formal methods such as Z [27] or VDM [24] require a mathematical background. For that reason, in practice it is not adopted by industry. Parnas says "paraadoxically, success stories reveal the failure of industry to adopt formal methods as standard procedures; if using these methods was routine, papers describing successful use would not be published" [22][26]. Other problems to adopt formal methods

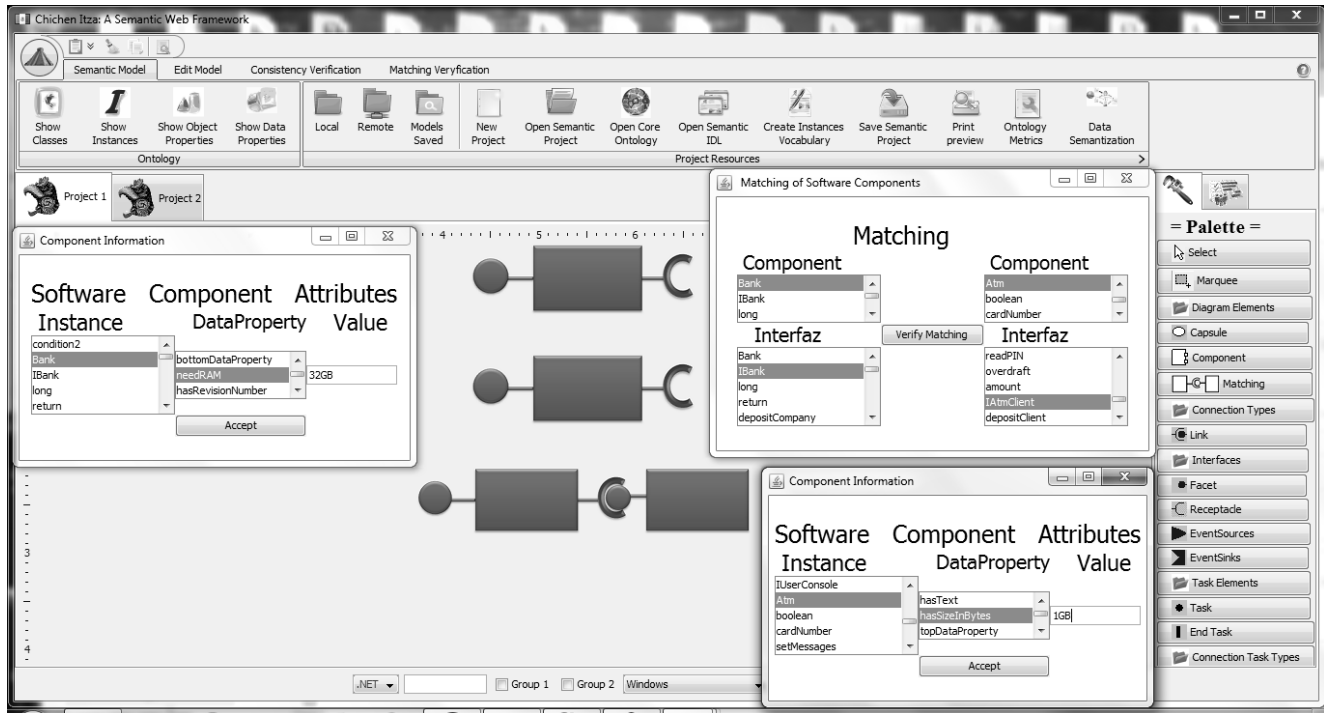


Figure 1. Visual assembled of software components in Chichen Itza framework.

are the time and the cost involved in the analysis of large-scale component-based systems [32]. For the reasons given above, formal methods require new ways to be applied. In this work, we propose a verification technique based on semantic techniques (Ontologies and SPARQL queries) in conjunction with *Chichen-Itza* framework to mitigate this problem. We propose an approach for verifying that *the contract of a required interface matches with the contract of a provided interface*. Our approach is able to check contract conformance of syntactic and behavioral aspects [6]. The former involves verifying the compatibility of the signatures of a provided and a required interface. The latter is in charge of certifying that the values of the parameters are within a valid range and have a proper semantics. In this work, we only consider sequential composition in which the composed components are executed sequentially. We follow an ontology-based approach which involves a formal method but without the complexity of most formal methods. In this work, we consider the following definition: "A component is a reusable unit of deployment and composition that is accessed through an interface"[11]. In practice, we have noted problems related to interface incompatibility are frequent. In particular such as incompatibility with the semantics of operation parameters and interface operations (behavioral contracts [6]). We consider that the use of a semantic matching approach (a software component on-

tology) could help to detect interface incompatibility before the component-based system is deployed.

The rest of the paper is structured as follows. In Section 2 we present some related work. In Section 3 we briefly explain semantic web techniques and its elements (Ontologies and SPARQL queries). Section 4 describes our semantic approach for Verifying the Matching of Software Components. In Section 5 we show the feasibility of our technique by describing an example. In Section 6 we draw some concluding remarks. Finally, acknowledgments are given in Section 7.

2 Related Work

There are several works about techniques for verifying contracts, Brada [8], Mariani and Pezze [19], Tsai and Eric Y.T. Juan [31], Cernuda, Cueva et al [12]. The most closely related work about component contract verification was made by Barnett and Schulte [3]. They propose a method for implementing behavioral interface specifications on the .NET Platform using contracts to check the conformance of an implementation class and they define the AsmL specification language. In contrast with their work, our model is independent of platform and our semantic specification language is supported by a domain ontology about software components. Another related work was made by Lau

and Ukis [18]. They enrich component's interface with metadata which is used for preventing component's conflicts with the target execution environment. Their work is focused on .NET and J2EE frameworks. The ontologies based on software component and matching are mostly represented by work of Claus Pahl at Dublin City University [21] who made an ontology for software component matching. His ontology is based on DAML+OIL [4] logic language. Our ontology has an advance in logic and it is based on OWL-DL logic model. We have found other works about software component ontologies [30][20], but they need extra information to be able to verify contracts among components.

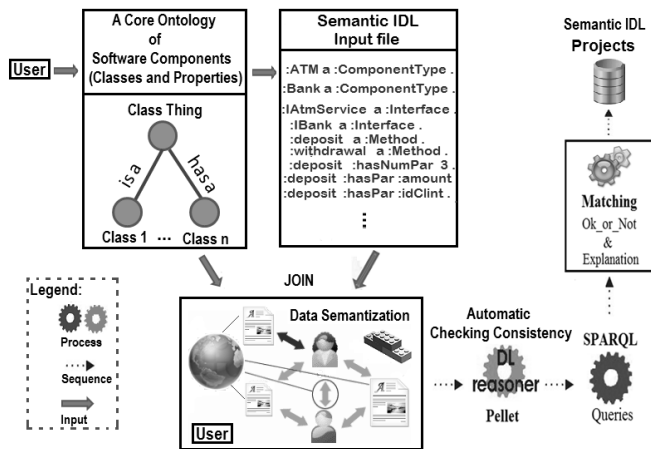


Figure 2. Semantic Verification Process

3. Semantic Web Techniques

3.1 Ontologies

An ontology [15] is a knowledge representation which defines the basic terms and relations comprising the vocabulary of a topic area, as well as the rules for combining terms and relationships used to define extensions to the vocabulary. In our case, the domain area is software components. In particular, the ontology is able to manage all attributes for software components, establishing links between two components that can be connected by means of its interfaces.

3.2 SPARQL Query Language

SPARQL is a query language for the Resource Description Framework (RDF), this is a W3C Recommendation [34]. We use Web Ontology Language (OWL) [35] which extends RDF and RDFS. We use the *disjointWith* property to verify compatibility among the instances created by the

user and its properties. We selected OWL DL language because we can assure that all conclusions given by the reasoner are computable and decidability. Example using RDF triples (*Parameter* class and *hasDataTypeParameter* object property) is showed below.

```
:Parameter a owl:Class .
:hasDataTypeParameter rdfs:domain
                        :Parameter .
:hasDataTypeParameter rdfs:range
                        :DataType .
```

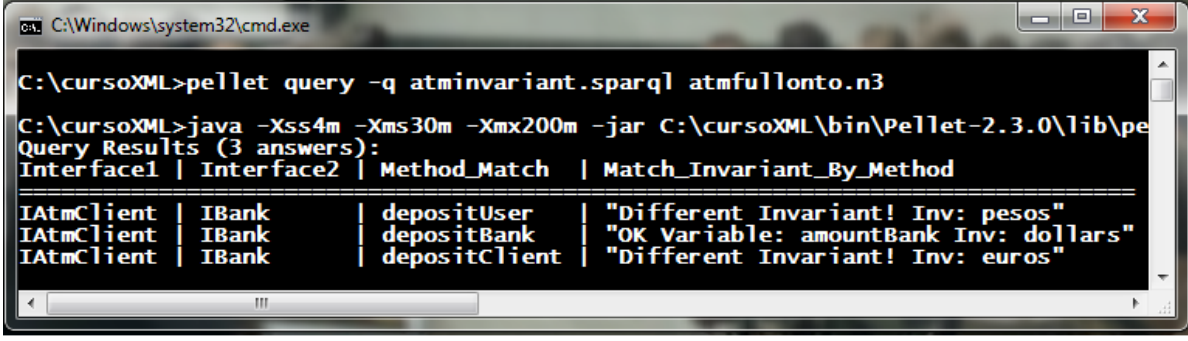
4 Chichen Itza: verifying the matching of software contracts

Chichen Itza¹ is a Semantic Framework which it consists of a visual editor of software architectures. See Fig.1. The tool makes use of the library Flamingo and the Ribbon component [16] implemented in Java. The process to verify a matching among components is very easy for the user.

4.1 A Core Ontology for Software Components

A Software Component Ontology was created for capturing and verifying information about the input domain models during the Architectural Design [14]. It was written using **notation 3 or n3** [5] which is similar to RDF in its XML syntax, but more easy to understand. This ontology consisted of *20 classes*, *28 Object Properties*, *36 Data Properties*. The ontology was written using *n3 notation*, it is used by RDFS and OWL DL logic model. The main classes are: *ComponentType*, *Interface*, *Method*, *DataType*, *Parameter*, *ComponentModel*, *PreCondition* and *PostCondition*. The Ontology is built by means of classes and relations among concepts. These concepts and classes correspond to the specification of an abstract data type and a set of methods that operate on that abstract data type. Each method is specified by an *interface*, *type declarations*, a *pre-condition*, and *post-condition* [11]. In addition, there are two types of interfaces (provided and required). The interface of a method describes the syntactic specification of the method. Interfaces define the methods used in contracts and composition. The typing information describes the types of input and output or both parameters and internal (local) variables. All of the above is represented in our ontology (class Type, class Parameter, etc.). The most important part to consider in our ontology are the Conditions (Pre and Post). The Pre-condition describes the condition of the variables prior to the execution of the method whose behavior is described by the Post-condition.

¹Chichen Itza is the name of a large pre-Columbian city built by the Maya civilization



```

C:\Windows\system32\cmd.exe
C:\cursoXML>pellet query -q atminvariant.sparql atmfullonto.n3
C:\cursoXML>java -Xss4m -Xms30m -Xmx200m -jar C:\cursoXML\bin\Pellet-2.3.0\lib\pe
Query Results (3 answers):
Interface1 | Interface2 | Method_Match | Match_Invariant_By_Method
-----
IAtmClient | IBank | depositUser | "Different Invariant! Inv: pesos"
IAtmClient | IBank | depositBank | "OK Variable: amountBank Inv: dollars"
IAtmClient | IBank | depositClient | "Different Invariant! Inv: euros"

```

Figure 3. Currency invariant verification using a SPARQL query

4.1.1 Evaluating the core ontology created

The ontology developed has been evaluated in an informal and formal way. Regarding the former, the ontology was evaluated by the developers using the *Pellet* reasoner [23] to check the consistency of the ontology. The second evaluation applied to the ontology is based on the work of Gómez-Pérez [4] who establishes five criteria (*consistency, completeness, conciseness, expandability* and *sensitiveness*). The number of concepts and their relations among them, allow us to check the ontology consistency with less steps than other kind of ontologies.

4.2 Verification of Contract Matching

Our definition about matching is based on interfaces as contracts by Szyperski [29]. Interface specifications are contracts between a client of an interface and a provider of an implementation of the interface. A contract states what the client needs to do to use the interface. It also states what the provider requires to implement to meet the services promised by the interface. We define *Contract Matching* among components when we say there is a component interface match when the provided interface of a component satisfies the requirements of the required interface of another component. Such a match is validated for syntactic and functional semantic aspects. In the first case, it is checked whether the provided interface includes at least the same list of methods defined in the required interface. We follow a structural approach whereby the names of the interface operations can be different but the types of the parameters and the order of the parameters must be compliant. In the case of functional semantic it is validated using SPARQL queries about Invariants, Pre- and PostCondition of methods. See Figure 4. Conditions defined for each method has to be matched with the same variable, logic operator and value. We verify restrictions and assumptions at construction time, in a completely static manner, prior to the testing stages. Semantic verification is the process which uses Semantic Web

Techniques (Ontologies and SPARQL queries) to guarantee compliance with contractual agreements. The semantics of an operation are described in an interface (contract). The only task for the user before to apply our model is to define the vocabulary of his domain and semantics. He introduces his model into the framework by means of a file or by the menus that allows us to do an automatic evaluation by using the Pellet reasoner [23] which checks inconsistencies. Chichen Itza transforms his vocabulary from a text file into an ontology instances and its relations. The instances are created from classes defined in the software component ontology.

4.3 Using CORBA-IDL vocabulary with Semantics

CORBA(Common Object Request Broker Architecture)[33] is a standard created by the Object Management Group (OMG)[10] that enables software components written in different computer languages for working among them by means of their interfaces. These interfaces are described using the *Interface Definition Language (IDL)*. In our semantic model, we need to receive the component interfaces written using the concepts and properties defined in the software component ontology and Bradas affirm that "developing CORBA components is rather tedious by today's standards due to its IDL-first approach" [8]. For the reasons above, we have decided to use the keywords of the CORBA-IDL with elements of the ontology and supported with Chichen Itza framework. For example, *ComponentType*, *Interface*, *Method*, *Parameter* and *hasNumParameters* are keywords. Part of the semantic ATM-IDL vocabulary. It is showed below.

```

:Atm          a :ComponentType .
:Bank         a :ComponentType .
:IAtmClient  a :Interface .
:IAtmClient   :hasMethod :deposit .
:IBank       a :Interface .

```

```

C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\cursoXML>pellet query -q atmprecond.sparql atmfullonto.n3

C:\cursoXML>java -Xss4m -Xms30m -Xmx200m -jar C:\cursoXML\bin\Pe
Query Results (1 answers):
Interface1 | Interface2 | Match_Method | Match_Precond
-----
IAtmClient | IBank          | "deposit"    | "amount > 0"

```

Figure 4. Matching the precondition about the amount

```

:IBank          :hasMethod :withdrawal .
:deposit       a :Method .
:withdrawal    a :Method .
:amout         a :Parameter .
:idClient      a :Parameter .
:deposit       :hasNumParameters 2 .
:withdrawal    :hasNumParameters 3 .
:deposit       :hasPrecond :condition1 .
:deposit       :hasPostcond :condition3 .

```

In the code above we would like to emphasize that there are some instances of classes (*Atm* and *Bank*), some classes (*ComponentType*, *Parameter*, *Interface* and *Method*), some object properties (*hasMethod*, *hasPrecond* and *hasPostcond*) and just one data type property (*hasNumParameter*). In particular, the notation *:deposit :hasNumParameters 2* means that the method *deposit* has exactly 2 parameters.

4.4 Using The Pellet Reasoner

Pellet [23] is an open-source Java based OWL DL reasoner. In our verification process we use Pellet for checking the consistency of the ontology. Pellet gives an explanation when an inconsistency is detected. Restrictions can be expressed into an ontology. For instance, the following code states that one component has at least 1 interface.

```

:Component rdfs:subClassOf
  [ a owl:Restriction ;
    owl:onProperty :hasInterface ;
    owl:cardinality 1 ].

```

In contrast with the Logic Programming paradigm, we can check types using ontologies. Besides, in the matching process subtypes can be accepted as parameters. See code below.

```

:Int a owl:Class .
:ShortInt rdfs:subClassOf :Int .

```

The *disjointWith* property allows for verifying restrictions in the input model (Semantic CORBA-IDL file). For example, we could establish that a component made in *.Net* can not run in the *Linux* operating system and the *EJB* component model is not compatible with the *MS COM* model. Defining *disjointWith* properties is also possible [1].

```

:Linux rdfs:subClassOf :OperatingSystem ;
  owl:disjointWith :Windows .
:EJB rdfs:subClassOf :ComponentModel ;
  owl:disjointWith :MS_COM .

```

All properties defined in the Ontology and blank nodes are checked by the reasoner (Pellet) during the consistency verification process.

4.5 Behavioral Contract verification using SPARQL

At this moment the complete verification is not possible using only the reasoner. For more complex checking we can apply another actions such as: production rules [13]. We decided to explore semantic queries in SPARQL [25] instead of production rules. The second step after the reasoner have checked the ontology consistency is to apply a SPARQL query. We defined specific queries that evaluate and verify the contract information of the components. Such queries are completely transparent to the user who only needs to provide the contracts of the components. We have used Jena API [17] and Java language [9] for programming and NetBeans IDE 7.0 [2]. SPARQL is similar to the database SQL but for ontologies. Besides, we can use variables in the queries, constraints, filtering information, logic operators, if statements and more. Lines are linked by variables which begin with a question mark. The same name of variable implies the same value to look for in the query. The *Jena API* allowed us to use SPARQL queries in our framework programmed in Java language. Part of the query which verifies the *Precondition matching* is showed below.

```

C:\Windows\system32\cmd.exe
C:\cursoXML>pellet query -q helpmatching.sparql atmfullonto.n3
C:\cursoXML>java -Xss4m -Xms30m -Xmx200m -jar C:\cursoXML\bin\Pellet-2.3.0\lib\pellet-cli.jar query -q helpmatching.sp
Query Results (8 answers):
Concept | Description | Example
-----|-----|-----
hasInvariant | "Object Property: :<Method> :hasInvariant :<Condition> ." | " :Deposit :hasInvariant :Condition1 ."
hasPostcond | "Object Property: :<Method> :hasPostcond :<Condition> ." | " :Deposit :hasPostcond :Condition1 ."
hasPrecond | "Object Property: :<Method> :hasPostcond :<Condition> ." | " :Deposit :hasPostcond :Condition1 ."
ComponentModel | "Class: Component Model" | "CORBA, EJB, NET, COM, etc."
ComponentType | "Class: Instances of Component Type" | "Instances of Components"
Condition | "Class: Condition used by Pre, Post and Invariant" | "Condition"
Contract | "Class: Contract between 2 software components" | "Contract between two components"
Author | "Class: Author of the Software Component" | "Edgar Castillo"

```

Figure 5. Semantic IDL help using the ontology

```

PREFIX      : <http://www.ejemplo.org/#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

SELECT DISTINCT ?Interface1 ?Interface2
               ?Match_Method ?Match_Precond
{
?Interface1 :typeInterface          :required ;
             :hasMethod              ?Method1 .
?Method1    :hasParameter            ?par1 ;
             :hasMethodName          ?name1 ;
             :hasNumParameters       ?numpar1 .
?par1       :hasIndexOrder           ?pos1 ;
             :hasDataTypeParameter  ?partype1 .
?Method1    :hasPrecond              ?precond1 .
?precond1   :hasVariable             ?var1 .
?precond1   :hasOperator             ?opr1 .
?opr1       :symbolOperator          ?oprname1 .
?precond1   :hasNumber               ?num1 .
             :
} order by ?Match_Method

```

An additional benefit of using ontologies and SPARQL queries has been the extra information (metadata) to offer support for writing the IDL file. See Figure 5.

5 Automated Teller Machine: example

ATM is a machine at a bank branch or other location which enables customers to perform basic banking activities. The component model used for describing the ATM was written using UML 2 notation [7], and is shown in figure 6. The vocabulary of the input model is created by the user who selects classes and relation among concepts and he creates his instances. In this case the input model (semantic IDL file) only has the information of 5 software components and we can create its instances and relations among them using the Chichen Itza's menus.

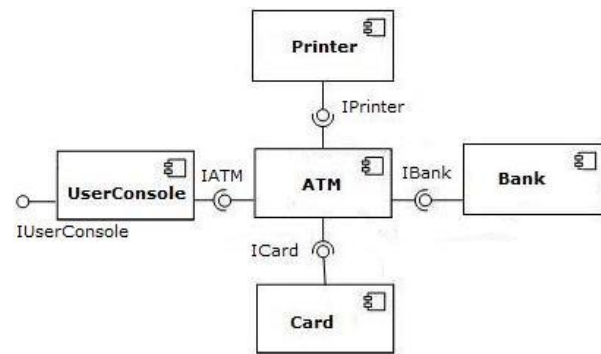


Figure 6. UML ATM Component-based system

6. Conclusions

In this paper we have presented and described a semantic technique for checking the Matching of Software Components. In comparison with other formal methods, this semantic technique, based on logic (ontology), a reasoner and a set of SPARQL queries offers an easy way to check matching among components and verifying the model proposed. This model can be extended and enriched with more concepts that rely on architectural design and non-functional requirements (QoS). The Ontology was expressed in a logic-based language (OWL DL), enabling detailed, sound, meaningful distinctions to be made among the contracts expressed as classes, properties and relations. The OWL DL ontology proposed is checked with the Pellet reasoner. Because it has a finite complexity. The use of a core domain ontology permits us to search for specific component information using intelligent techniques like SPARQL queries. Extending the ontology with no functional properties (Quality of Services attributes), Design Patterns and object properties (*hasInvoke*, *hasResponse*, etc.) for dynamic behaviour are key points for our future work.

7 Acknowledgments

This material is partly based upon work supported by the National Science Foundation under Grant No. OISE-0730065. I would like to express my deepest thanks to Comision Mexico-Estados Unidos para el Intercambio Educativo y Cultural (COMEXUS) through the Fulbright Programme, for gave me a grant to support my research in Miami, Florida without which this work would not have been possible, and Florida International University (FIU), School of Computing and Information Sciences. In particular, I would like to thank to Dr. S. Masoud Sadjadi and his team for their fruitful discussions and support for my research.

References

- [1] F. E. Antoniou Grigoris and V. H. Frank. Introduction to semantic web ontology languages. 2005.
- [2] E. Armstrong, J. Ball, S. Bodoff, D. B. Carson, I. Evans, K. Ganfield, D. Green, K. Haase, E. Jendrock, J. Jullion-ceccarelli, and G. Wielenga. The j2ee™(tm) 1.4 tutorial for netbeans™(tm) ide 4.1 for sun java system application server platform edition 8.1.
- [3] M. Barnett and W. Schulte. Contracts, components, and their runtime verification on the .net platform. *J. Systems and Software, Special Issue on Component-Based Software Engineering*, 2002.
- [4] S. Bechhofer, C. A. Goble, and I. Horrocks. Daml+oil is not enough. In *SWWS*, pages 151–159, 2001.
- [5] T. Berners-Lee, D. Connolly, and S. Hawke. Semantic web tutorial using n3. In *Twelfth International World Wide Web Conference*, 2003.
- [6] A. Beugnard, J. Jézéquel, N. Plouzeau, and D. Watkins. Making components contract aware. *Computer*, 32(7):38–45, 1999.
- [7] M. Bjerkander and C. Kobryn. Architecting systems with uml 2.0. *Software, IEEE*, 20(4):57–61, 2003.
- [8] P. Brada. The cosi component model: Reviving the black-box nature of components. *Component-Based Software Engineering*, pages 318–333, 2008.
- [9] P. J. Clarke, D. Babich, T. M. King, and B. M. G. Kibria. Model checking and abstraction. *ACM Transactions on Programming Languages and Systems*, 16:1512–1542, 1994.
- [10] O. CORBA and I. Specification. Object management group, 1999.
- [11] I. Crnkovic and M. Larsson. Building reliable component-based software systems. Artech House computing library, Norwood, MA, 2002.
- [12] A. del Río, J. Gayo, and J. Lovelle. Verificación y validación mediante un modelo de componentes. In *Actas del Simposio Iberoamericano de Sistemas de Información e Ingeniería del Software en la Sociedad del Conocimiento (SISOFT-2001), Bogotá (Colombia)*, pages 29–31.
- [13] A. C. del Río, J. E. L. Gayo, and J. M. C. Lovelle. A model for integrating knowledge into component-based software development. *KM - SOCO*, pages 26–29, 2001.
- [14] A. Eden and R. Kazman. Architecture, design, implementation. In *proceedings of the 25th International Conference on Software Engineering*, pages 149–159. IEEE Computer Society, 2003.
- [15] T. Gruber. Toward principles for the design of ontologies used for knowledge sharing. pages 907–928. 1995.
- [16] Java.net. Flamingo. <http://java.net/projects/flamingo/>, 2010.
- [17] Jena. Jena a semantic web framework for java. 2000.
- [18] K. Lau and V. Ukis. Deployment contracts for software components. *Preprint*, 36, 2006.
- [19] L. Mariani and M. Pezze. A technique for verifying component-based software3. *Electronic Notes in Theoretical Computer Science*, 116:17–30, 2005.
- [20] X. Nianfang, Y. Xiaohui, and L. Xinke. Software components description based on ontology. In *Proceedings of the 2010 Second International Conference on Computer Modeling and Simulation - Volume 04, ICCMS '10*, pages 423–426, Washington, DC, USA, 2010. IEEE Computer Society.
- [21] C. Pahl. An ontology for software component matching. volume 9, pages 169–178. Springer-Verlag, Berlin, Heidelberg, 2007.
- [22] D. Parnas. Really rethinking 'formal methods'. *Computer*, 43(1):28–34, Jan. 2010.
- [23] B. Parsia and E. Sirin. Pellet: An owl dl reasoner. In *In Proceedings of the International Workshop on Description Logics*, 2004.
- [24] J. P. Paul, P. and J. I. Siddiqui. *Formal Methods State of the Art and New Directions*. Springer, Springer London Dordrecht Heidelberg New York, 2009.
- [25] J. Pérez, M. Arenas, and C. Gutierrez. Semantics and complexity of sparql. *The Semantic Web-ISWC 2006*, pages 30–43, 2006.
- [26] G. Reed. Exploiting formal methods in the real world: a case study of an academic spin-off company. In *Proceedings of Modelling Software System Structures in a fastly moving scenario*, 2000.
- [27] J. Spivey. *Understanding Z: a specification language and its formal semantics*, volume 3. Cambridge Univ Pr, 1988.
- [28] C. Szyperski, D. Gruntz, and S. Murer. *Component software: beyond object-oriented programming*. Addison-Wesley Professional, 2002.
- [29] A. Talevski, P. Wongthongtham, and S. Komchaliaw. Towards a software component ontology. In *Proceedings of the 10th International Conference on Information Integration and Web-based Applications & Services, iiWAS '08*, pages 503–507, New York, NY, USA, 2008. ACM.
- [30] J. Tsai and E. Juan. Compositional approach for modeling and verification of component-based software systems. In *Proceedings of the 2000 Monterey Workshop on Modeling Software System Structures in a Fast Moving Scenario*, pages 13–16. Citeseer.
- [31] J. J. P. Tsai and E. Y. T. Juan. Compositional approach for modeling and verification of component-based software systems. In *Proceedings of Modelling Software System Structures in a fastly moving scenario*, 2000.
- [32] S. Vinoski. Distributed object computing with corba. *C++ Report*, 5(6):32–38, 1993.
- [33] W3C. <http://www.w3.org/consortium/>. 1994.
- [34] W3C. Owl web ontology language, 1994.

SESSION

EMBEDDED SYSTEMS, APPS, ANDROID, AND APPLICATIONS

Chair(s)

TBA

Software Engineering a Family of Complex Systems Simulation Model Apps on Android Tablets

V. Du Preez, B. Pearce, K.A. Hawick and T.H. McMullen

Computer Science, Institute for Information and Mathematical Sciences,

Massey University, North Shore 102-904, Auckland, New Zealand

{ dupreezvictor, brad.pearce.nz }@gmail.com, { k.a.hawick, t.h.mcmullen }@massey.ac.nz

Tel: +64 9 414 0800 Fax: +64 9 441 8181

March 2012

ABSTRACT

Tablet computers are emerging as powerful platforms for educational and demonstration software in areas like computational science and simulation which previously had needed higher performance processing. Developing software as an App or even porting it from other interactive platforms still requires non-trivial software engineering effort however. We describe how a family of complex systems simulation models were developed as a domain related family of Apps and discuss the software engineering issues we encountered in generalising the data structures, and simulation code patterns to run as Android Apps. We also discuss performance achieved on various models with a range of modern tablet computers and other devices with similar processors. We speculate on how domain-specific software engineering methods could automate such future simulation model App development.

KEY WORDS

software architecture; tablet; Apps; Android; ARM.

1 Introduction

Engineering software for tablet computers is an exciting area of development with emerging framework software proving capabilities to exploit the mobile nature and touch screen interaction capabilities of such devices. Despite a somewhat tumultuous history [1, 2], these devices have already found uses in mobile gaming [3–5] and other computationally intensive applications beyond simple data entry [6, 7].

These platforms are here to stay and have a strong potential [8]. In this paper we describe our interests in developing software for computational interactive simulations [9] and our efforts in engineering simulation Apps for tablet computers using the Android operating system [10–12] and associated code frameworks.

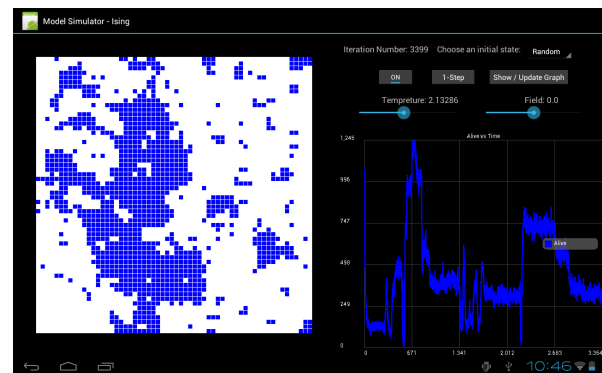


Figure 1: Screen-dump showing our App running the Ising Model

Tablets are powerful platforms for simulation demonstration models, due to their highly interactive graphical display capabilities. However coding for this interactivity can be demanding due to the still maturing touch and multi touch interfaces. Tablet architecture is becoming more and more advanced with the development of dual core and recently quad core systems. This allows for models to become more interactive and become more graphically intense.

Mobile operating systems, such as Apple's IOS and Google's Android have provided developers a solid platform to implement classical models on to test both the devices and the models as interaction with the model is made easy. Stumbling blocks are present however when moving a model from a PC to a mobile operating system. Considerations such as power management, memory management and conservation, and limited computational power all need to be considered when implementing a chosen model.

The Android platform allows a somewhat self managed system for its Apps due to the use of the Dalvik virtual machine [13]. This is a type of Java virtual machine which is

much like Sun's JVM but has optimizations in the areas of power management, better memory sharing and other aspects which enable strong performance on mobile devices such as phones and tablets.

Today's tablets and mobile devices have Advanced Risk Machine (ARM) processors [14]. This is as these ARM processors are low powered and cheap to make which allows fast and efficient mobile computing in comparison to more powerful and power hungry PC CPUs. Even though ARM processors are now as powerful as PC processors they are still capable of running the latest mobile operating systems of IOS and Android. Many devices have multiple ARM processors which perform different aspects for the tablet or mobile device.

Multiple ARM processors and other hardware can be formed into an integrated circuit known as a System on Chip (SOC). A SOC can contain all the hardware that is required to operate a tablet or mobile device, hardware such as multiple ARM processors for applications, GPS, blue-tooth, WiFi and RF communications.

The SOC also contains many busses that link all the processors and many other bits of hardware for peripheral devices for the tablets. There is a limited number of companies that manufacture SOC's which are known for slight optimizations in certain areas of the various chips. SOC's however are not limited to tablets and mobile devices, there are various organisations that have used SOC's in various other development boards such as Panda [15] and Raspberry Pi [16].

As both ARM processors and SOC's become more complex and powerful this allows for tablets and other mobile devices to take advantage of portable and interactive computing power which allows for interesting research in App development. Tablets have been optimized with version of an ARM architecture processor for longer battery life vs battery weight.

ARM gained popularity with more power and cost efficient way to build tablet or mobile computers, running both Android and IOS. The touch display capability supports direct model manipulation of complex systems such as the: Ising model of a magnet [17–20]; Game of Life [21, 22]; Game of Death [23, 24]; Kawasaki exchange model [25, 26]; and Sznajd opinion model [27, 28] which we discuss in Section 2. Figure 1 shows a screen-shot from our App running the Ising model simulation.

In Section 3 we describe various aspects of Android's App architecture that we exploited to engineer our Apps the platform works. In Section 4 we explain our results from the various models and we discuss this further in our discussion in Section 5. We offer some conclusions and directions for further work in Section 6.

2 Model Family

The driving motivation for our building a family of simulation model Apps was our desire to demonstrate these to students learning about the models [29], coupled with the realisation that many of these models have a great deal in common. In software engineering terminology they constitute a family of domain models that are closely related. We found that it was feasible and desirable to construct our App software to exploit this observation.

The models themselves have been described extensively elsewhere - both by ourselves and by other authors. However we give a brief summary of the key features here for completeness and to explain the decisions and results discussed in this present article.

The Game of Life and Game of Death are cellular automaton models. Each cell is initialised to a state or live, dead (or zombie) and a deterministic rule is applied to change the state according to its own state and that of its immediate neighbours. These models are surprisingly complex systems whereby rich and unexpectedly structured spatial patterns of cells emerge from the very simple microscopic rules applied deterministically to the individual cells.

The Ising model is almost as simple a model. In it, a heat-bath algorithm is used to emulate thermal effects on atoms in a magnetic material arranged in a crystalline lattice. The Ising system consists of a micro crystalline array of single bit magnetic moments or "spins" which interacts with its nearest neighbours. At each time step of the simulation each spin is considered in turn and the energy and thermal probability of it "flipping" – reversing its direction are considered. The probability of flipping is different, depending upon the applied temperature.

We find that spins align with their neighbours when the system is cold, but thermally randomize when it is hot. The interesting feature about the Ising system in 2 (or 3) dimensions is that there is a definite Curie temperature that can be measured. In real magnets the Curie temperature is the temperature above which the material stops being a magnet, or an alternative viewpoint is that materials like iron spontaneously become magnetic below their Curie temperature. This is known as a phase transition and is very difficult to explain simply without a model to demonstrate. The value of an interactive simulation is that the temperature parameter can be directly varied by the user and the effects seen in real time. Thus, it is possible for a user of our App to cool or quench a simulated Ising magnet down and see it undergo its phase transition and all the spins start to spontaneously align with one another to produce complex patterns.

The Kawasaki exchange model is constructed in a similar manner to the Ising system. In this case however we

preserve a fixed ratio of the two microscopic species since instead of flipping or changing species, in the Kawasaki system we only allow them to swap positions with one of their (randomly chosen) neighbours. In this respect the Kawasaki system models diffusion and phase separation or “unmixing” of the two species. The rate and manner of unmixing is like the separation of two atomic species in a binary alloy. This sort of dynamical behaviour is of great importance in real materials. Without some separated granules an alloy typically lacks strength and other physical properties but if too much separation occurs it can break apart and cause catastrophic failure in for example fuel rods in a reactor.

Models of opinion formation and propagation are now recognised as important ways to understand crowd behaviours, social and political phenomena. The Sznajd opinion model is also formulated in a manner similar to the Ising or Kawasaki models, but the update rules are simpler. Each species represents an opinion, and at each step if two cells of the same opinion are found next to one another we allow the possibility of them “persuading” all their neighbouring sites to the same opinion. In this sense opinions get pushed out to the neighbours via the Sznajd update rule.

We encounter similar complex spatial patterns formed by all these models. They are all modelled as an array of sites on for example a square lattice. Each cell can take on a discrete and small number of different states – live/dead/zombie or spin-up/spin-down or alloy species x/y , or political opinions $A/B/C$ and so forth. An array of bytes is suited to this model. The update rules are only slightly different for each model so some commonality in code structure is possible.

For convenience in applied the update rules symmetrically we impose periodic boundary conditions - so that a cell at the edge of the array still sees the same number of neighbouring cells as one in the middle. This does not affect the physics or complexity of the model and also supports visualisation actions such as panning and zooming around in a wrap-around display of the simulated system.

The key criteria for engineering a family of Apps for these models are the performance capabilities and managing the code complexity to implement suitable model system sizes on current generation tablet computers using the code development frameworks available for them.

3 Android Java Apps Architecture

The Android operating system has emerged as a powerful and popular platform for developing Apps to run on mobile platforms such as tablet computers and smart phones. Android is a Unix/Linux-like operating system with many well established, tested and widely used and understood

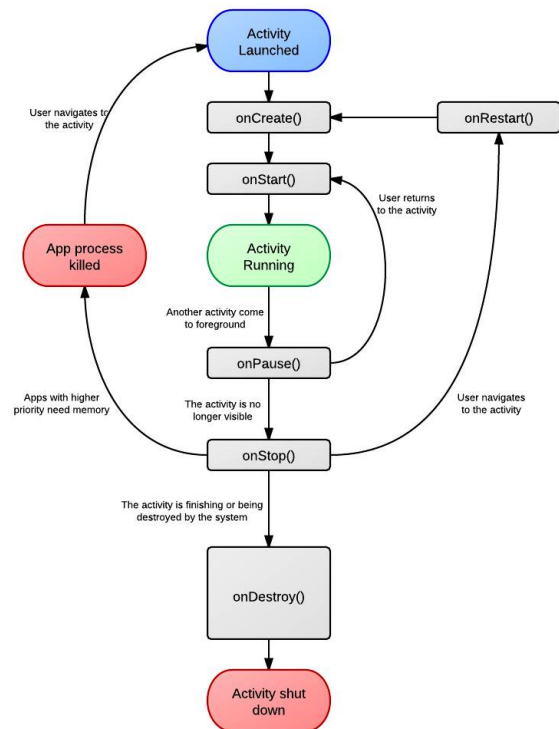


Figure 2: Android Activity Life-cycle

internal software models. Its relative openness makes it an attractive development platform for mobile applications.

Figure 2 shows the Android activity life-cycle. It shows the different calls an Android activity makes on its creation, halt and restart. Our application used many of these activity call-backs to set up and run our simulations. We mainly used the `onCreate()` function to initialize data and classes and the `onPause()` function to halt the simulations when the activity was not longer in view by the user using the home button or the back button. We also used the `onRestart()` call-back to restart our App if the user navigated back to the application. This life-cycle call-back functions of the activity are extremely useful in application as they are mainly event driven. However as can be seen the system does the handling of properly destroying the activity which means from a programming point of view you need to be careful about how you exit and clean up in your application.

The code listed in Figure 3 is an example of a running model in the main loop. This is pseudo-code for creating new thread controlling the main loop and handling the invalidation for the GUI. The code stipulates that when the activity is active update the model in a continuous loop otherwise a one-step mode can be performed which updates each step individually with button control.

Figure 4 and Figure 5 show pseudo code for our rule im-

```

runSimulation function{
new Thread {
  While (active){
    if (redraw){
      calluiupdate ()
    }else if (Running){
      choose rule set
      calluiupdate ()
    }else if (one step){
      choose rule set
      calluiupdate ()
    }
  }
}
}

```

Figure 3: Pseudo code for main loop

```

golconwayrules (){
get number of cells in grid;
  for(int i=0;i<totalcells;i++){
    count num of live neighbours;
    if ((state == 0) && (count == 3)){
      set alive;
    }else if ((state == 1) &&
      ((count == 2) || (count == 3)))
      set state to alive;
    }else{
      state dies;
    }
  }
}

```

Figure 4: Pseudo code for Game of life computation

plementation of the Game of life and the Game of Death. For the game of life we loop through all the cells. For each one we get the sum of the number of live cells surrounding the current cell and then apply the rules to the cell bases on this count. The rules are implemented in the if statements. After every cell has been updated this is considered one step or iteration in the game of life.

For the Game of Death in Figure 5 the implementation of the rules was slightly more complex due to the addition state of zombie but it is very similar to the game of life. Again we loop through all the cells, for each we need to know its current state and how many are alive around it. We can then apply the rules in the if statements to the cell to produce the next state. When all cells have been updates we again consider that one iteration.

The App architecture can derived from Figure 2 showing the App runtime environment. Android supports what is know as a multiple activities model and the screen area is managed using various "activities." We found it useful creating multiple activities to navigate through the application selecting and initializing each model. Once activity is cre-

```

public static void godrules(){
  get number of cells in grid;
  for(int i=0;i<totalcells;i++){
    count num of live neighbours;
    if state alive and
      count is 3 or count is 2{
      then set alive state;
    }else if state is alive{
      set to dead state;
    }
    if state is dead and count is 3{
      set to alive state;
    }else if state is 0{
      set state to zombie;
    }
    if state is zombie and count is 2{
      set state to alive;
    }else if state is zombie{
      set state to zombie;
    }
  }
}

```

Figure 5: Pseudo code for Game of life computation

ated it can send Intents to the Android system which starts other activities. This allows us to combine loosely couple components to perform certain tasks.

Graphical Apps creates a single thread of execution by the Android OS which is called the main thread. UI, invalidation and canvas drawing are performed on this thread and we implemented this with the code on Figure 3, creating a new thread for our drawing to the canvas. Developers can not access the main thread directly, however Async-Task can be used for thread management. Manipulating thread and handlers are avoided when the UI thread controls background operations and result.

These models can be defined in a arbitrary dimension most commonly described as "hyper bricks" of mesh points. This allows us to have a block of memory to index any point in the brick using a single integer which is known as the "k-index" [30]. In our implementation both a k-index and x coordinate system was used due to X,Y location coordinates being precomputed and stored in an array rather than converting to X and Y from a k-index. This was to reduce the amount of modulus operations in the code which has been known reduce performance in computational models. The negative aspect of precomputing the X and Y coordinates is that it requires additional storage for X,Y coordinates on top of storage of the state of each cell.

4 Performance Results

We have measured the simulation and graphical update performance of our model Apps on various specific tablet

Device	FPS	Time(s)
Motorola Xoom	26	38.22s
Samsung Galaxy Tab	34.8	28.95
Asus Transformer	28.2	35.58s
Panda Developer board	17	59.69s
Android Emulator	4.6	213.57s
Animaux (PC Build)	24.7	40.5s

Table 1: Averages for 1000 iterations for Game of Life

Device	FPS	Time(s)
Motorola Xoom	15.2	69.16s
Samsung Galaxy Tab	18.8	55.58s
Asus Transformer	17	58.63s
Panda Developer board	11	95.16s
Android Emulator	4.6	213.57s
Animaux (PC Build)	24.7	40.5s

Table 2: Averages for 1000 iterations for Game of Death

and other compute platforms.

Table 1, Table 2, Table 3 and Table 4 all show interesting results when compared to each other. The tested tablets, Motorola Xoom, Samsung Galaxy Tab and the Asus Transformer all showed similar performance. This was expected as they all contain the same hardware (See Table 5). This was apparent when comparing the Xoom and the Transformer tablet devices. The Galaxy Tablet however performed much better than expected in comparison to the other two tablets. We can only speculate as to why this occurred. The Panda Developer board performed slower than the tablet devices which was surprising as it is very similar in hardware.

The Android emulator that is provided with the Android SDK performed the worst of all tested devices. We suspect this to be as all the hardware of the emulator is emulated in software unlike the other devices that run on physical hardware. The Animaux program for PC performed the same across the all the models. This was expected as the program is programmed in a totally different way with different classes and underlying classes and data structures. Therefore it make it hard for us to draw any relevant con-

Device	FPS	Time(s)
Motorola Xoom	24.2	41.30s
Samsung Galaxy Tab	31	32.41s
Asus Transformer	26.2	38.40s
Panda Developer board	18.2	58.04s
Android Emulator	4.4	274.15s
Animaux (PC Build)	24.7	40.5s

Table 3: Averages for 1000 iterations for Ising

Device	FPS	Time(s)
Motorola Xoom	87.8	11.50s
Samsung Galaxy Tab	128.4	7.77s
Asus Transformer	78.8	12.73s
Panda Developer board	88.6	11.40s
Android Emulator	6.8	153.87s
Animaux (PC Build)	24.7	40.5s

Table 4: Averages for 1000 iterations for Sznjad

clusions as to how powerful PC simulations are compared to those on tablet based devices.

Table 5 shows each device specification that these complex models was computed on. Android tablets and the Panda Developer board running the Dual Core Cortex A-9 chip from the ARM family, this was somewhat interesting since similar results are expected between Android devices. These Android tablets are identical and the only difference to similar devices is the Graphics Processing Units (GPU) in the Panda Development board. Android tablets use a 8 core 333 MHz GPU whereas the Panda boards use 4 core 384 MHz both supporting OpenGL ES 2.0.

Results shown the Samsung device having much faster refresh rates and lower time steps compared to other tablet devices. Since all the models had the Android Ice Cream Sandwich 4.0.3 running as their main operating system, we concluded differences in driver, memory and resource management between the different vendors. Android emulator showed slow results due to the ARM architecture that Android is based on. Personal computers are based on the x86 architecture and thus the emulator need to be executed on a virtual machine which was irrelevant to the test we wanted to contrive on Android OS.

5 Discussion

Our test environments which were all dual core architecture (Tegra 2). This allows creation of a genuinely concurrent new thread so that calculations of the rules and Android's output on the display can be separated. This provided significant performance increases. (see performance comparison graph/grid). This is as Android/Java's system, garbage collection and user display processes are on one core and the other core is left to do all the calculations on the rule-set. When the newer quad core architecture is available (Tegra 3), the App may then be able to run the various models a lot faster as it splits the work load across the four cores the cpu provides.

Models showed that similarities such computation time and refresh rate could be closely correlated, however the Game of Death showed these variables to be slower in all aspects. Having three states was display intensive

Device	CPU	Memory	GPU	L2/L3 Cache
Motorola Xoom	NVIDIA Tegra 2 T20	1 GB - DDR2	ULP GeForce	1MB - L2
Samsung Galaxy Tab	NVIDIA Tegra 2 T20	1 GB - DDR2	ULP GeForce	1MB - L2
Asus Transformer	NVIDIA Tegra 2 T20	1 GB - DDR2	ULP GeForce	1MB - L2
Panda Developer board	Dual-core Cortex-A9	1 GB - DDR2	SGX540 graphics core	1MB - L2
Android Emulator	NA	256Mb - VM	NA	NA
Animaux (PC Build)	3.2 GHz Intel Core i3	4 GB DDR3	ATI Radeon HD 5670 512 MB	4MB - L3

Table 5: Specifications for the computer platforms used.

on the model with garbage collection(GC) initially running at 40%, this was optimized using the Android Debug Bridge(adb). This was a crucial tool developing the App showing thread usage and garbage collection.

The various models across different devices and platforms gave us many interesting and unexpected results. Firstly the various models, Game of life, Game of Death, Ising, and Sznajd all produced interesting results across the three tablet devices. As can be seen from table one through four all the tablets showed slight performance differences. This was interesting as all the tablets are based on the same dual core, Tegra 2 architecture. It was surprising to see the rather large variation in the amount of time it took to do 1000 steps in the various models across the Xoom, Transformer and Galaxy Tablet. The only thing we can put these variations are down to the specific hardware differences of the three tablet devices and the device drivers associated. This could also be due to different ARM processor versions, cache, clock speeds, stand alone GPU processors in devices.

The FPS of the various devices varies in direct relation to the time taken to perform the 1000 iterations of the model. As FPS rises the time taken decreases. This was due to the multi-threaded nature of the application and how Android platform separates its processes across its various available cores. The Panda development board fared somewhat worse than expected in testing. This may be due to its unoptimized nature to the Android platform as the panda board is also designed to run various other operating systems. The Android emulator that comes with the Android SDK performed slowest of all across all the models. This we suspect is due to the emulator running via software rather than hardware like on the tablets. This may be causing a bottleneck in the emulator in graphic output.

The PC based Animaux model performed uniformly across all models in terms of iteration times and FPS. This performed well and it was obvious that the PC implementation had a lot more possible power that could be squeezed out of it if optimized to the models such as the Android App had. Even though the Animaux model was also written in Java, underlying classes and grid structure were used. This

means differences in neighbour discovery, state recovery. Differences in 1000 iteration times in the four models was due to what neighbours were needed by the various models and the requirements or computing them. As these models are loop intensive resource reuse was something we needed to be wary of as garbage collection by the Dalvik virtual machine can be time intensive. The Sznajd model requires the least amount of neighbours to be retrieved hence it performs the fastest.

6 Conclusions

We have described how we architected and developed a family of Apps for tablet computers using Android and associated Java frameworks. We have shown that modern tablet devices produce quite credible performance even when compared to a desk top implementation - for interactive simulation demonstrations. We have discussed software and implementation commonalities across our initial family of application simulation models - including simple automata such as the Game of Life and Game of Death through more sophisticated automata-like models such as the Sznajd opinion models to models like the Ising and Kawasaki models which require random number generation and mathematical function evaluations.

The Android code development framework is quite well suited to such a family of Apps and we have been able to share a lot of code and data structures across this family of models. The graphical interface code and associated idiomatic structures are rather different for a touch sensitive screen found on the tablet to the more common mouse environment and widget collection used on desktops but it is feasible to find ways for a user to exploit either.

The Android activities model and the associated life-cycle is a good framework for the interactive nature of Android applications. It lends itself well to the event driven flow and style of direct model manipulation in computational models. The activity design with call-back functions enables users to prioritize and handle views and the underlying computations in a clean and theoretical way. It also allows easy instantiation and destruction of activities and associated objects that the activity needs before and after

the running of the activity. This aspect of the Android system allowed us to handle and optimize our activities and the flow of our activities to enable our App to be as intuitive as possible.

An interesting area for future development will be to find ways to combine these model generation ideas so that code can be generated for either a tablet or a desktop platform. We have made some inroads into exploring the commonalities across our target simulation models and algorithms, and have identified ways for them to share data structures. There is still considerable work to be done however to identify a common software model architecture for the user interaction and parameters control framework that would be needed to fully automatically generate Desktop and App codes.

References

- [1] Yarow, J.: Tablet computing: A history of failure. *Business Insider Online* (2010) 1–3
- [2] Norman, D.A.: Inside risks: Yet another technology cusp: Confusion, vendor wars, and opportunities. *Communications of the ACM* **55** (2012) 30–32
- [3] Cheng, K.W.: *Casual gaming*. VU Amsterdam (2011)
- [4] Kemp, R., Palmer, N., Kielmann, T., Bal, H.: Opportunistic communication for multiplayer mobile gaming: Lessons learned from photoshoot. In: *Proc. Second Int. Workshop on Mobile Opportunistic Networking (MobiOpp'10)*, Pisa, Italy (2010) 182–184
- [5] Feijoo, C., Ramos, S., Gomez-Barroso, J.L.: An analysis of mobile gaming development - the role of the software platforms. In: *Proc. Business Models for Mobile Platforms(BMMP10)*, Berlin, Germany (2010)
- [6] Buchanan, N.: An examination of electronic tablet based menus for the restaurant industry. Master's thesis, University of Delaware (2011)
- [7] Castellucci, S.J., MacKenzie, I.S.: Gathering text entry metrics on android devices. In: *Proc. Computer Human Interactions (CHI2011)*, Vancouver, BC, Canada (2011) 1507–1512
- [8] Coulter, R.: Tablet computing is here to stay, and will force changes in laptops and phones. *Mansueto Ventures* (2011)
- [9] Huynh, D.B.P., Knezevic, D.J., Peterson, J.W., Patera, A.T.: High-fidelity real-time simulation on deployed platforms. *Computers & Fluids* **43** (2011) 74–81
- [10] Conti, J.P.: The androids are coming. *Engineering and Technology Magazine* **May - June** (2008) 72–75
- [11] Johnson, M.J., K. A. H.: Porting the google android mobile operating system to legacy hardware. In: *Proc. IASTED Int. Conf. on Portable Lifestyle Devices (PLD 2010)*, Marina Del Rey, USA (2010) 620–625
- [12] Kim, S.: Logical user interface modeling for multimedia embedded systems. In: *Proc. Int. Conf on Multimedia, Computer Graphics and Broadcasting (MulGrab 2011)*, Jeju Island, Korea (2011)
- [13] Ehringer, D.: The dalvik virtual machine architecture. Technical report, Google (2010)
- [14] Sloss, A.N.: *ARM System Developer's Guide: Designing and Optimizing System Software*. Elsevier (2010) ISBN 978-0080-490-496.
- [15] Panda: Panda board development resources (2012)
- [16] Pi, R.: Raspberry pi - an arm gnu/linux box (2012)
- [17] Ising, E.: Beitrag zur Theorie des Ferromagnetismus. *Zeitschrift fuer Physik* **31** (1925) 253?258
- [18] Hawick, K., Leist, A., Playne, D.: Cluster and fast update lattice simulations using graphical processing units. Technical Report CSTN-104, Computer Science, Massey University (2009)
- [19] Leist, A., Playne, D., Hawick, K.: Interactive visualisation of spins and clusters in regular and small-world Ising models with CUDA on GPUs. *Journal of Computational Science* **1** (2010) 33–40
- [20] Hawick, K.A.: *Domain Growth in Alloys*. PhD thesis, Edinburgh University (1991)
- [21] Gardner, M.: Mathematical Games: The fantastic combinations of John Conway's new solitaire game "Life". *Scientific American* **223** (1970) 120–123
- [22] Hawick, K.: Cycles, diversity and competition in rock-paper-scissors-lizard-spock spatial game simulations. In: *Proc. International Conference on Artificial Intelligence (ICAI'11)*, Las Vegas, USA (2011)
- [23] Resnick, M., Silverman, B.: Exploring emergence: The brain rules. <http://llk.media.mit.edu/projects/emergence/mutants.html> (1996) MIT Media, Laboratory, Lifelong Kindergarten Group.
- [24] Hawick, K., Scogings, C.: Cycles, transients, and complexity in the game of death spatial automaton. In: *Proc. International Conference on Scientific Computing (CSC'11)*. Number CSC4040, Las Vegas, USA (2011)
- [25] Kawasaki, K.: Diffusion constants near the critical point for time dependent Ising model I. *Phys. Rev.* **145** (1966) 224–230
- [26] Hawick, K.: Visualising multi-phase lattice gas fluid layering simulations. In: *Proc. International Conference on Modeling, Simulation and Visualization Methods (MSV'11)*, Las Vegas, USA (2011)
- [27] Sznajd-Weron, K., Sznajd-Weron, J.: Opinion evolution in closed community. *Int. J. Modern Physics C* **11** (2000) 1157–1165
- [28] Hawick, K.: Multi-party and spatial influence effects on opinion formation models. In: *Proc. IASTED International Conference on Modelling and Simulation (MS 2010)*. Number CSTN-032, Calgary, Canada (2010) Paper 696-035.
- [29] Fenwick, J.B., Kurtz, B.L., Hollingworth, J.: Teaching mobile computing and developing software to support computer science education. In: *Proc. 42nd ACM Tech. Symp. on Computer Science Education SIGCSE'11*, Dallas, Texas (2011) 589–594
- [30] Hawick, K.A., Playne, D.P.: Hypercubic Storage Layout and Transforms in Arbitrary Dimensions using GPUs and CUDA. *Concurrency and Computation: Practice and Experience* **23** (2011) 1027–1050

A Model-driven development approach to design software for embedded systems using the Android platform

L.V. Cobaleda¹, J. C. Villa², J. J. Yepes², J.F. Duitama¹, J.E. Aedo².

¹ARTICA, Software Engineering Research Group

²ARTICA, Microelectronic and Control Research Group

Universidad de Antioquia

Medellin, Colombia

Abstract— the complexity in embedded systems applications at the present time has been increasing considerably. This paper proposes an iterative and incremental approach to design software applications for embedded systems based on Model-Driven Software Development (MDS). MDS helps to develop software by enhancing its reusability and maintainability. The proposed approach is tested by using it to develop a self-monitoring home healthcare system, which gathers and processes information from a generic Wireless Body Sensor Network (WBSN) using Bluetooth technology. The prototype was developed using both the Android software development kit (SDK) and the development platform Eclipse, integrated with IBM Rational Rhapsody plugin.

Index Terms— Android, wireless sensor network, embedded system, methodology, model-driven development.

1. INTRODUCTION

In recent years, Wireless Sensor Networks (WSN¹) have gained worldwide interest, due to their potential for developing applications in a wide range of areas: health, environmental monitoring, control and monitoring processes. In consequence, it is reasonable to expect that many programmers will engage in the development of the next generation of software applications for embedded systems, mobile devices and WSN. Integrating a WSN with a cell phone through protocols is a common design practice in some applications due to the high capacity network connectivity and computational performance of the mobile phone [1]. Various companies have developed mobile devices with different computational capabilities according to people's needs. In some cases, companies open their application programming interfaces (APIs), allowing anyone to develop their own applications for the company's devices [2].

Software development for mobile devices is supported by several methodologies and some of these are specifically for embedded systems development [3]. This paper proposes a methodology to software development for embedded systems

with the android mobile platform that considers a communication channel between Wireless Body Sensor Networks (WBSN) and mobile phone with Android.

This paper is organized as follows: Section II describes related work. Section III describes the application development for the case study. Section IV describes the proposed methodology. Section V shows platforms and the toolkit used to build an application following the proposed methodology. Section VI presents conclusions and future work.

2. RELATED WORK

The development of embedded systems that integrate devices and mobile solutions is gaining acceptance, especially in telemedicine. In [4] the authors study the possibility of implementing a logging application on a smart-phone to help diabetic patients with their daily lives. These patients have to keep track of measured blood glucose levels, as well as daily routines that affect their condition. Incorporating this functionality into the user's cell phone gives easy access, mobility and communication capabilities. The intention was to provide tools that enable the patients to understand aspects about their condition and to simplify data entry. The results show that it is possible to create an alternative solution to conventional logging that includes features missing in today's systems.

The Clinication system [5] is a web- and cell phone-based Patient Adherence Management System. This system can send reminders via e-mail, or send text messages to a patient's mobile. One of the modules of the system is called CellPly, and patients can use this module on a mobile device to control and monitor aspects of their treatment. The patient will receive messages to remind them of their treatment activities, informational messages about the therapy and questionnaires to monitor vital signs. CellPly can be used in addition to the Clinication portal or independently of it.

In the context of therapeutic adherence, another work [6] suggests some elements for working collaboratively to improve the levels of adherence, such as a monitoring system with wireless sensors to capture relevant patient variables like blood pressure, oxygen saturation, heart rate and respiratory

¹ WSN: The Wireless Sensor Networks are distributed devices, self-employed, which use sensors to cooperate in a common task.

rate; a smart pillbox that stores medicines and communicates messages through a wireless channel when it is opened; a call center where medical staff receive alerts indicating low adherence; and finally, an application for managing medical records.

Another work [7] proposes model-driven development of mobile personal healthcare applications. The authors developed an approach to modeling care plans for chronic disease using two domain-specific visual languages (DSVLs). The first allows healthcare providers to model complex care plans, health activities, performance measurements and sub-care plans. The second DSVL describes a mobile device interface for the care plan. A code generator synthesizes mobile device implementation of this care plan application. Unlike these works, in our application communicates with a wireless body sensor network which is managed from the communication and processing of physiological variables. All was developed with MDSO approach.

3. CASE STUDY

This section describes a software application for a cell phone designed for Android operating system that employs a communication channel with a WBSN to obtain and process the information gathered by nodes in a self-monitoring home healthcare context. Moreover, this case study is subsequently used to detail the methodology proposed in this paper.

This work was developed in the context of tele-assistance. It considers a home care scenario for pneumological patients (patients with respiratory disease) who regularly require the measurement of oxygen saturation and heart rate (among other variables) to allow severity assessment of a possible asthma attack. In this case, it would be useful if, after a measurement taken by the pulse oximeter, the personal mobile device not only shows the patient's measurements, but also sends an alert to the health care provider if the measurements are considerably outside of the accepted range.

This application was developed to facilitate the monitoring of patients with respiratory diseases who are being treated at home by medical personnel. The patient regularly takes a measurement of oxygen saturation, heart rate and cardiac activity with a pulse oximeter node and an electrocardiogram (EKG) node, both connected point to point via Bluetooth to a mobile phone. In the mobile device the information received is analyzed according to the ranges of acceptable values for the patient and, if there is a risk, the mobile device sends a warning message to the health care provider. This scenario is shown in Figure 1.

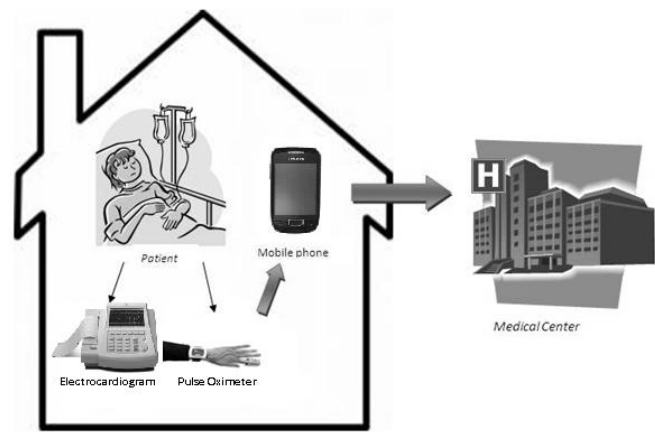


Figure 1 Case study scenario

4. METHODOLOGY

The methodological approach proposed in this paper is based on MDSO. This section describes the proposed method and presents an example design of the case study: a mobile application for monitoring home healthcare patients.

A. Model-Driven Software Development approach

MDSO approach attempts to find domain-specific abstractions and make them accessible through formal modeling. It creates a great potential for automation of software production, leading to increased productivity, quality, maintainability, reusability, and manageability of software complexity. This approach is intended to promote the use of formal and abstract models which can be easily understood by knowledge domain experts.

MDSO separates the specification of the structure and essential system functionality from its implementation using platform-specific models [8] [9]. Thus, it is possible to use concepts that are much less tied to the underlying technology and closer to the problem domain. The domain specification is defined in Platform-Independent Models (PIM), and is completely separated from target platform implementation. The Platform-Specific Models (PSM) are created from the PIM and through model transformations. Finally, the application code is generated.

B. Methodological approach

The methodological model-driven approach presented in this work is iterative and incremental; the key issue is to define a number of short-time iterations, each of them having a group of similar steps; the goal is, after finishing one iteration, to refine the system specification and development, resulting in the final complete product. The aim of the first phase is to select and develop some suitable requirements to produce a stable increment. In the second place, it is to adjust the high-level and independent-platform models, in order to achieve a high reusability and maintainability of the software components, and finally the last phase consists of generating the code for a specific platform.

Therefore, the first step is to identify the fundamental functional (FR) and non-functional (NFR) requirements, which are used to establish the base architecture for the following development. The base architecture is set down in formal high-level platform-independent models. After that, the semi-automatic transformation to platform specific code is

done and the first version of code is generated and tested. In the next steps, additional requirements are specified and detailed, the architecture is refined and the code generation and tests are completed. These activities are presented in detail in the following section.

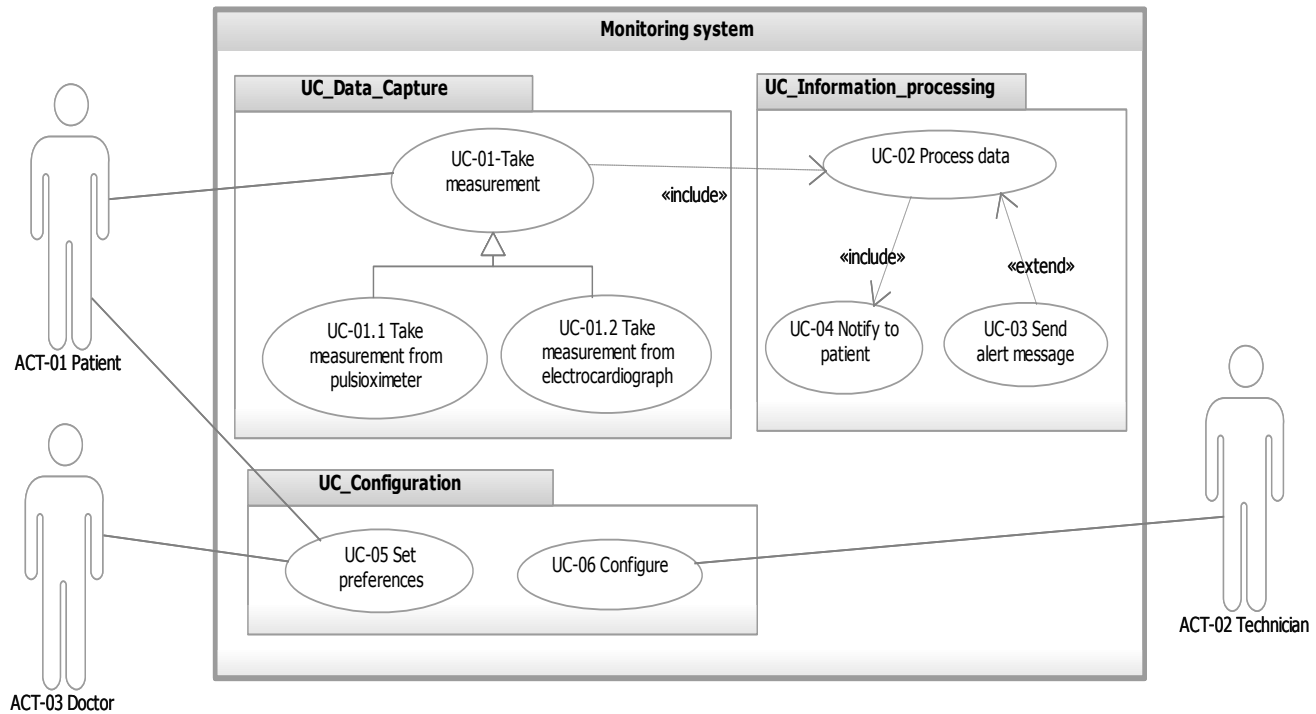


Figure 2 Use Case Diagram

C. Development of requirements

In this phase it is important to select the suitable requirements that will guide the subsequent phases; i.e. the actors, their responsibilities, some functional features and constraints.

In the case study, several requirements were identified in the first iteration, but only those essential to establish the base architecture were selected. These are related to the communication with two WBNS nodes. Figure 2 shows the use case model in which the principal requirements are specified.

Although the system requires communication with two medical devices, this application could require more devices, so the design is projected to be scalable to more medical devices.

D. Application Model

The second phase is intended to create the PIM. The application packages are organized according to the main responsibilities identified in the software system. Focusing on reusability and maintainability is a key issue in this approach. Another activity in this phase is to define the detailed design of classes that conserves the logical distribution made before. Here the design patterns are applied, and the sequence diagrams and state charts are created.

The logical approach in this work is based on the Five-Layer architecture Pattern [10] but applied in a simple way. Specifically, the Abstract Hardware and Operative Systems Layers are not clearly separated because the current requirements are not demanding this partition. Thus, this application is divided into three layers, shown in Figure 3, where the responsibilities concerning the display of information, data processing and technical communication are clearly grouped. This arrangement makes reusability and debugging control easier.

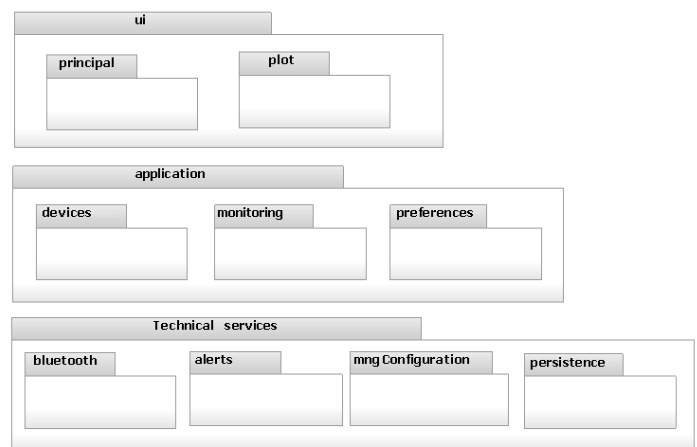


Figure 3 Application Packages

User Interface (UI) layer is responsible for showing the medical measurements to users and for capturing the configuration values. This layer contains specialized view classes corresponding to each medical device. In consequence, it has one view for the pulse oximeter and one for the electrocardiogram. There is also a sub package in this layer, called *plot*, which is responsible for drawing the electrocardiogram signals.

Application (App) layer is responsible for embracing the specific domain classes. The *Devices* package is the most important in this layer. It has a class structure where the extensibility of new functionality is easily allowable through the addition of subclasses. This is shown in Figure 4.

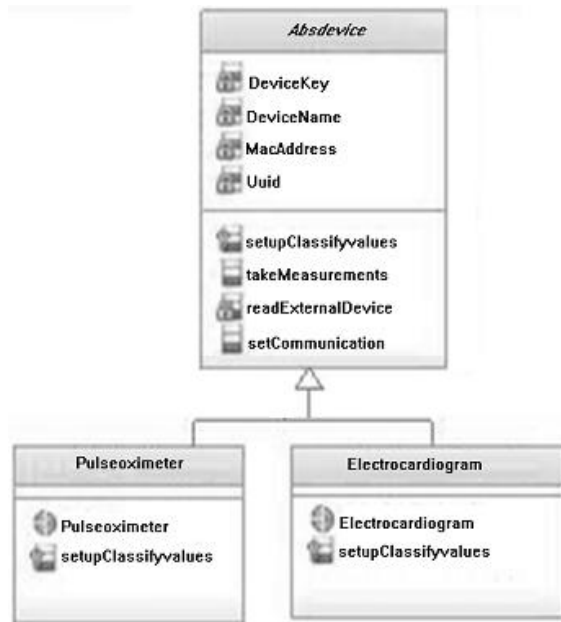


Figure 4 Extract of class diagram for *Devices* package

Classes inside of Technical services (TS) layer implement technical features of communication. In particular, the Bluetooth Serial Port Profile (SPP) was implemented at application level to establish a communication channel between the cell phone and WBSN.

Some of the design patterns adopted in this work are the *Observer pattern* to notify the Bluetooth discovery process; the *Front Controller pattern* to draw the electrocardiogram signs; and the *Channel architecture pattern* to transform the data from medical devices [10]. The sequence diagrams were created and the functionalities were assigned to the classes according to the logical distribution of packages. An extract of this diagram is shown in Figure 5.

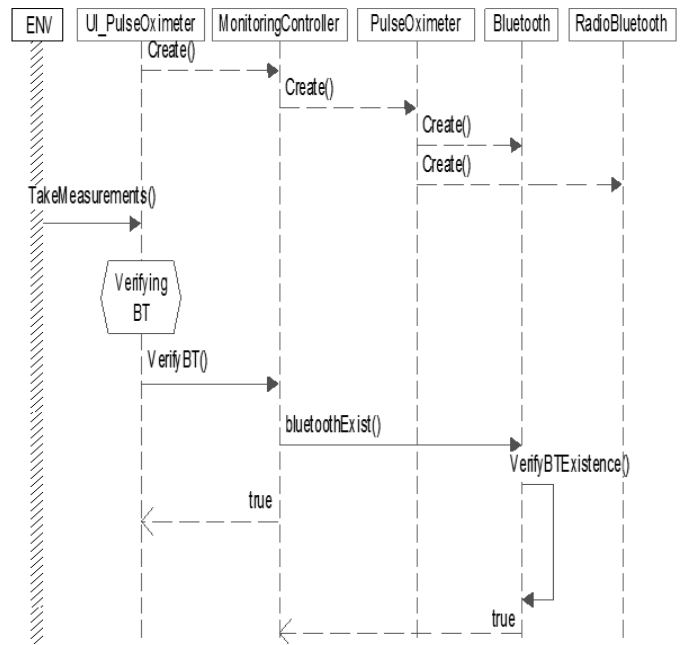


Figure 5 Sequence diagram for verifying Bluetooth

E. Code generation

In this phase, the process of transforming the high-level platform independent models into code for a specific platform is done in a semi-automated way. Subsequently, the particular modifications are introduced in the code directly. The code generation for android 2.2 is done with Rational Rhapsody plug-in for the Eclipse software development kit.

The self-monitoring mobile application runs on Android 2.2 or higher in a context of WBSN. The application establishes Bluetooth communication with a WBSN which captures corporal signs such as oxygen saturation, heart rate and cardiac activity. If the patient measurements are outside of an acceptable range, the mobile sends a message to a health care provider.

The Java code is generated with the Rational Rhapsody tool where some parts were written manually. It is important to emphasize that it is possible to accomplish the iterative development process due to the Rational Rhapsody tool capability for keeping the consistency between models and generated code. Figure 6 shows the work with Rhapsody plug-in tool for Eclipse.

This proposed architecture has also been used in the development of an application for an embedded system with Android, named *System Integration and Interoperability Medical monitoring for Tele-care* (SIMMIT), which is developed by the Microelectronic and Control Research Group. The packages, the class structure and the patterns used were similar. However, the implementation of the channel architecture pattern was different due to the means of communication with external devices, which was achieved

through a USB²-serial interface. In this way, this architecture evidences a valuable reference framework for developing new embedded applications in this domain.

```
import android.content.Context;
import android.util.Log;

public class Pulseoximeter extends Absdevice{

    private String hearthRate = null;
    private String oxygenSaturation = null;

    public Pulseoximeter(Context context) {
        super(context);
        setKeyDevice(Setup.PULSEOXIMETER_KEY);

        setDeviceName(Setup.getInstance(contexto).getDeviceName(
        Configuracion.PULSEOXIMETER_KEY));
    }
}
```

Figure 6 Extract of code

F. Testing

In the deployment process, two WBSN nodes gather data about some vital signs using the cell phone. The pulse oximeter node takes heart rate and oxygen saturation using Bluetooth communication with no difficulty. This is shown in Figure 7.

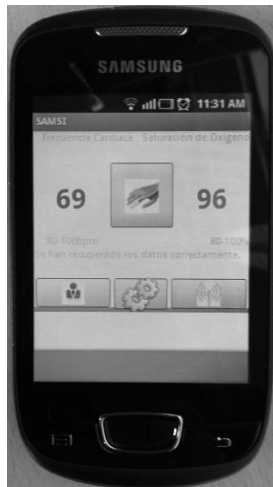


Figure 7 Application Running

5. PLATFORMS AND TOOLS DESCRIPTION

In this section, a set consisting of a toolkit and platforms utilized to build the application are described.

This work was developed with the Android software development kit (SDK) and the development platform Eclipse, integrated with IBM Rational Rhapsody. These platforms and tools were useful for modeling use case diagrams, class diagrams, sequence diagrams and state charts; also for generating code for Android 2.2 (Froyo) platform, in order to interchange information with a generic WBSN node.

A. Eclipse

Eclipse is a software development kit (SDK) [11] consisting of the Eclipse Platform, Java development tools and the Plug-in Development Environment [12]. This Platform is a multi-language software environment composed of an integrated development environment (IDE) and an extensible plug-in system; it can be used to develop applications in various programming languages including Ada, C, C++, Java, Perl, PHP, Python, Ruby (including Ruby on Rails framework) and Scheme. Development environments include the Eclipse Java development tools (JDT) for Java, Eclipse CDT for C/C++, and Eclipse PDT for PHP, among others.

B. Android Software Development Kit (SDK)

The SDK for Android includes a set of development tools [13]. This SDK includes a debugger, libraries, a phone simulator, documentation, sample code and tutorials. The integrated development environment (IDE) is officially supported by Eclipse Plug-ADT (Android Development Tools plugin) and also it can control Android devices that are connected.

C. IBM Rational Rhapsody

Rational Rhapsody is a model-driven development environment for embedded systems based on UML [14]. It was primarily designed to accelerate development, manage complexity, enhance testability, reduce costs and improve quality by leveraging the Object Management Group's (OMG's) Systems Modeling Language (SysML) and Unified Modeling Language (UML) standards. Throughout the development process, Rational Rhapsody was built to manage complexity through visualization, and helps maintain consistency across the development life cycle to facilitate agility in response to ever changing requirements.

D. Samsung GT-S5570

The Samsung GT-S5570 (also known as Samsung Galaxy Mini) is a low-cost smartphone manufactured by Samsung [15]. It was released with the operating system Android 2.2 (Froyo) and Android 2.3(Gingerbread). The Galaxy Mini is a 3.5G smartphone that offers quad-band GSM at 7.2 Mbit/s and also counts with GPS receiver with A-GPS, Bluetooth 1,0 and WIFI 802.11 b/g connections. The display is a 3.14-inch (80 mm) Thin Film Transistor-Liquid Crystal Display(TFT LCD), capacitive touchscreen of vertical QVGA (240x320) resolution. It also has an ARMv6 600 MHz processor and 384 MB RAM with 279 MB available.

E. Pulse Oximeter and Electrocardiogram Nodes

Pulse Oximeter and Electrocardiogram nodes are generic small nodes for health monitoring developed by ARTICA [16]; these are small WBSN systems designed for health monitoring applications. They are composed by MSP430 processor, for the ATMega128RFA1 radio transceiver chip, and the RN-42 Bluetooth chip. Each node communicates with

² USB: Universal Serial Bus

the other via 802.15.4 protocol. In this wireless sensor network, the sink node is the pulse oximeter, which is in charge of gathering information, and sensing the oxygen saturation and heart rate. The Electrocardiogram (EKG) node senses cardiac activity. Figure 8 shows pulse oximeter node and Figure 9 shows EKG node.

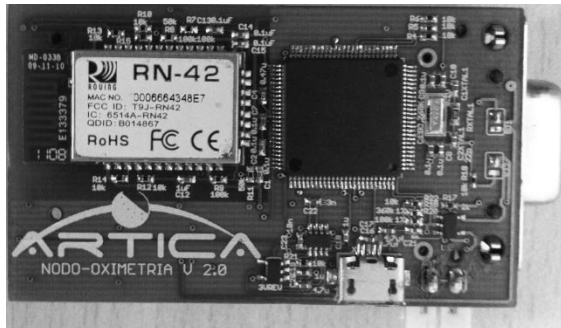


Figure 8 Pulse Oximeter Node

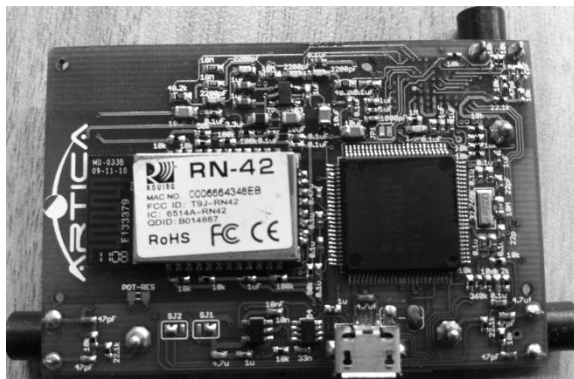


Figure 9 Electrocardiogram Node

6. CONCLUSION AND FUTURE WORK

This paper describes an iterative, incremental and model-driven methodological approach for embedded systems with the android mobile platform that considers a communication channel between WBSN and the embedded system. This approach is demonstrated through a case study of the design of a self-monitoring home healthcare software system that uses a communication channel with a WBSN. It enhances software reusability, extensibility and maintainability: a layered architecture is proposed to improve the reusability and the maintainability, a polymorphic structure is proposed in the application layer to support the extensibility, and the use of some design patterns are proposed to improve the reutilization. The architecture proposed in this work has been tested in other scenarios in the healthcare context and its usefulness evidenced as reference architecture for the development of applications for embedded systems. Future work will include improving the robustness and reliability in the methodological approach and integration with new Android versions.

ACKNOWLEDGMENT

We would like to express our thanks to the Excellence research Center, ARTICA and to the members of

Microelectronic research group and Software Engineering Research Group from Antioquia University.

7. REFERENCES

- [1] G. D. Abowd, L. Iftode, and H. Mitchell. Guest editors' introduction: The smart phone—a first platform for pervasive computing. *IEEE Pervasive Computing*, 4(2):18–19, 2005.
- [2] Beaton, J.; Sae Young Jeong; Yingyu Xie; Stylos, J.; Myers, B.A.; , "Usability challenges for enterprise service-oriented architecture APIs," *Visual Languages and Human-Centric Computing*, 2008. VL/HCC 2008. IEEE Symposium on , vol., no., pp.193-196, 15-19 Sept. 2008.
- [3] Bruce Powel Douglass. *Real-Time Agility: The Harmony/ESW Method for Real-Time and Embedded Systems Development*. Addison-Wesley Professional. 2009.
- [4] A Widen. *Diabetes care on smart phones running the Android platform Design and implementation of a system to help self- monitoring and managing*. Master's Thesis Chalmers University Sweden.
- [5] «Clinication Home Page». [Online]. Available: <http://clinication.com/>. [Accessed: march-23-2012].
- [6] Duitama, F., Gaviria Gómez, N. and Aedo Cobo, J.E. Technical report Macroproyecto Telesalud (ARTICA), Universidad de Antioquia, 2008.
- [7] A. Khambati, J. Grundy, J. Warren, J. Hosking. 2008. Model-Driven Development of Mobile Personal Health Care Applications Found in: *Automated Software Engineering*, International Conference. Issue Date: September 2008 pp. 467-470
- [8] DSDM: Desarrollo de Software Dirigido por Modelos. MDA y Aplicaciones. Universidad de Málaga. [Online]. Available: <http://www.lcc.uma.es/~av/MDD-MDA/>
- [9] Stahl, T. and Völter, M. 2006. *Model – Driven Software Development*. John Wiley & Sons, Ltd.
- [10] B. Powel Douglas. *Real-Time Design Patterns: Robust Scalable Architecture for Real-Time Systems*.
- [11] "What are Eclipse projects and technologies?" [Online]. Available: http://wiki.eclipse.org/FAQ_What_are_Eclipse_projects_and_technologies%3F
- [12] "Eclipse Build Drop". Eclipse Foundation. [Online]. Available: <http://download.eclipse.org/eclipse/downloads/drops/R-3.7.1-201109091335/details.php>
- [13] «Tools Overview». Android Developers [Online]. Available: <http://developer.android.com/guide/developing/tools/index.html>
- [14] Telelogic Rational Rhapsody Getting Started Guide. pg 1. 2007
- [15] "Samsung Galaxy Mini Homepage". [Online]. Available: <http://www.samsung.com/latin/consumer/mobile-phones/mobile-phones/smartphone/GT-S5570EGATCL>
- [16] Artica. [Online]. Available: <http://www.articadct.com/>

Recognizing recurrent development behaviors corresponding to Android OS release life-cycle

Pavel Senin

Collaborative Software Development Laboratory
Information and Computer Sciences Department
University of Hawaii at Manoa
Honolulu, Hawaii, 96822
senin@hawaii.edu

Abstract—Within the field of software repository mining (MSR) researchers deal with a problem of discovery of interesting and actionable information about software projects. It is a common practice to perform analyzes on the various levels of abstraction of change events, for example by aggregating change-events into time-series. Following this, I investigate the applicability of SAX-based approximation and indexing of time-series with *tf*idf* weights in order to discover recurrent behaviors within development process. The proposed workflow starts by extracting and aggregating of revision control data and followed by reduction and transformation of aggregated data into symbolic space with PAA and SAX. Resulting SAX words then grouped into dictionaries associated with software process constraints known to influence behaviors, such as time, location, employment, etc. These, in turn, are investigated with the use of *tf*idf* statistics as a dissimilarity measure in order to discover behavioral patterns.

As a proof of the concept I have applied this technique to software process artifact trails corresponding to Android OS¹ development, where it was able to discover recurrent behaviors in the “new code lines dynamics” before and after release. By building a classifier upon these behaviors, I was able to successfully recognize pre- and post-release behaviors within the same and similar sub-projects of Android OS.

Keywords: software process, recurrent behaviors, data-mining

I. INTRODUCTION

By the large body of previous research it has been shown, that software process artifact trail (change events and associated metadata) is a rich source of process and developers’ information and characteristics. The ability to discover recurrent behaviors with Fourier Analysis of change events is explained in [1], while another work [2] connects recurrent behaviors and software product quality. Thus, potentially, it is possible to relate recurrent behaviors to software product quality and to software process efficiency. The main part of a toolkit aiding such research is not only an efficient mechanism of recurrent behaviors discovery, but a mechanism of recognition of social and project-related constraints modulating these behaviors. This paper presents my exploratory study resulted in a universal framework

for temporal partitioning and mining of software change artifacts. As an evaluation example, it presents a recurrent behaviors discovery from the data extracted from Android SCM (software configuration management) system.

The rest of the paper is organized as follows. In Section 2, I discuss the motivation, results of previous work in MSR and present the research questions. In Section 3, I consider the workflow, data selection, collection, partitioning, and describe algorithms and methods. Section 4 presents results and the contribution. Finally, in Section 5, I discuss limitations and possible extension of this work.

II. MOTIVATION

Software development is a human activity resulting in a software product. The software process is a structure imposed on the software development. This structure identifies a set and an order of activities performed to design, develop and maintain software systems. Examples of such activities include design methods; requirements collection and creation of UML diagrams; requirements testing; performance analysis, and others. The intent behind a software process is to structure and coordinate human activities in order to achieve the goal - deliver a software system successfully. Many processes and process methodologies exist today, and it has been found, that the amount of time and effort needed to complete a software project, and the quality of the final product, are heavily affected by the software process choice [3]. Thus, studying software processes is one of the important areas of software engineering.

Traditionally, the software process study is built from top to bottom: it requires the researcher to guess a whole process, or to notice a recurrent pattern of behavior upfront, and to study it in a variety of settings later. These empirical studies usually involve two expensive and limited in scale techniques: interviewing and monitoring of the developers. Furthermore, these techniques are virtually impossible to apply within open-source project settings where a diverse development community scattered over the globe. Fortunately, current advances in software configuration management (SCM) technologies enable researchers to study

¹<http://source.android.com>

software process by mining software artifact trails [4], such as change logs, bug and issue tracking systems and mailing lists archives.

Mining of large software repositories demands advanced techniques allowing to tame with the complexities of data extraction and its analysis. These challenges are not new to the data-mining community and an enormous wealth of methods, algorithms and data structures have been developed to address these issues. While some of these approaches were already implemented within the field, such as finding of trends, periodicity and recurrent behaviors through the linear predictive coding and cepstrum coefficients [5], Fourier Transform [1] and coding [6], many are yet to be tried.

In this paper, I investigate the application of Symbolic Aggregate Approximation [7] and the term frequency-inverse document frequency weight statistics ($tf*idf$) [8] to the problem of discovering recurrent behaviors from software process artifacts. The motivation behind this choice is coming from the demonstration of outstanding performance by SAX in time-series mining, and from the wide range of successful applications of $tf*idf$ statistics, which is focusing on measuring the degree of dissimilarity as the opposite to convenient similarity metrics. Implementation of this approach I validate on Android SCM data.

A. Research question

In this exploratory work I am investigating the applicability of Lin&Keogh symbolic approximation technique combined with $tf*idf$ statistics to the discovery of recurrent behaviors from SCM trails of Android OS. The research questions I am addressing are:

- Which kinds of SCM data need to be collected for such analyzes?
- What is the optimal approach to data representation and a data storage configuration?
- Which partitioning (slicing) is appropriate, and which set of parameters should one use for SAX approximation?
- What is the general mining workflow?

III. EXPERIMENTAL SETUP AND METHODS

In this section I explain the steps of the recurrent behaviors discovery workflow along with their theoretical background.

A. Data collection and organization

As with many other large open-source projects, Android OS has been in the development for many years. It is “an open-source software stack for mobile phones and other devices”, which is based on the Linux 2.6 monolithic kernel. Development of Android was begun by Android Inc., the small startup company. In 2005, the company was acquired by Google which formed the Open Handset Alliance - a consortium of 84 companies which announced the availability of the Android Software Development Kit (SDK) in November

2007. The Android OS code is open and released under the Apache License.

Google platform is used for hosting, issue and bug tracking systems, whether Git is used as the distributed version control system for Android. The source code is organized into more than 200 of sub-projects by function (kernel, UI, mailing system, etc.) and underlying hardware (CPU type, bluetooth communication chip, etc.). There are about two million change records registered in the Android SCM by more than eleven thousands of contributors within an eight year span. The richness of this data makes Android SCM very interesting repository for exploring.

By using provided Google Data API for bugs and issues data retrieval, and custom coded Git repository data collection engine, I have collected information about bugs and issues, the revision tree, authors and committers, change messages, and affected targets. In addition to that data, by creating a local mirror and by iterating over changes, I was able to recover the auxiliary data for the most of the change records. This auxiliary data provides quantitative summary of added, modified, and deleted targets, as well as the summary about LOC changes: added, modified or deleted lines. All this information was stored in the relational database. Main tables of this database correspond to change and issue events; these accompanied with change target tables, issue details, comments, and tables for contributor authentication. Overall, the database was normalized and optimized for the fast retrieval of change and issue information using SQL language.

The collected data constitute almost full set of collectible artifacts. The only lacking information is the precise information for source-code line changes, which I intentionally omitted in this step due to the storage space and collection time constraints. Despite of being collected, bugs and issues data has not been included into recurrent behaviors discovery experiments in this work mostly due to the complexity of change-issue relations. However, as was shown by previous research, this data is a valuable source of information for recurrent behaviors discovery [2].

B. Temporal data partitioning

By following the previous research targeting social characteristics of committers [2], as well as the release pattern discovery [6], I have partitioned and organized the collected change trails by the time of the day using time windows of

- Full day, 12AM - 12AM
- Late night, 12AM - 04AM
- Early morning, 04AM - 08AM
- Day, 08AM - 05PM
- Night, 05PM - 12AM

For every of these windows, I then aggregated values for commits, added, edited, or deleted targets and lines, producing equidistant time-series abstraction of software development activity.

One of the effects of this data transformation is an instant increase of the number of change data entities by the factor of 5 and production of very sparse equidistant time-series. In order to reduce the sparseness and the complexity (dimensionality) of data, two additional procedures were applied within the post-collection data treatment step: PAA and SAX.

C. Piecewise Aggregate Approximation (PAA)

PAA performs a time-series feature extraction based on segmented means [9]. Given a time-series X of length n , application of PAA transforms it into vector $\bar{X} = (\bar{x}_1, \dots, \bar{x}_M)$ of any arbitrary length $M \leq n$ where each of \bar{x}_i is calculated by the following formula:

$$\bar{x}_i = \frac{M}{n} \sum_{j=n/M(i-1)+1}^{(n/M)i} x_j \quad (1)$$

This simply means that in order to reduce the dimensionality from n to M , we first divide the original time-series into M equally sized frames and secondly compute the mean values for each frame. The sequence assembled from the mean values is the PAA transform of the original time-series.

It worth noting, that PAA reduction of original data satisfies to a bounding condition, and guarantees no false dismissals in upstream analyzes as shown by Keogh et al. [10] by introducing the distance function:

$$D_{PAA}(\bar{X}, \bar{Y}) \equiv \sqrt{\frac{n}{M}} \sqrt{\sum_{i=1}^M (\bar{x}_i - \bar{y}_i)^2} \quad (2)$$

and showing that $D_{PAA}(\bar{X}, \bar{Y}) \leq D(X, Y)$.

D. Symbolic Aggregate approximation (SAX)

Symbolic Aggregate approximation extends the PAA-based approach, inheriting algorithmic simplicity and low

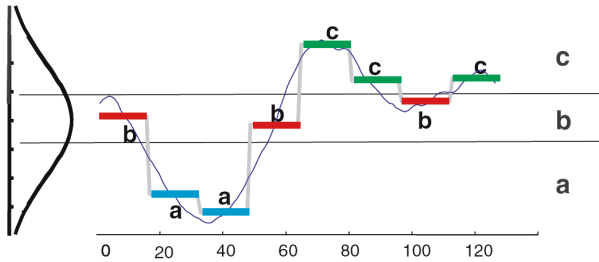


Figure 1: The illustration of the SAX approach taken from [7] depicts two pre-determined breakpoints for the three-symbols alphabet and the conversion of the time-series of length $n = 128$ into PAA representation followed by mapping of the PAA coefficients into SAX symbols with $w = 8$ and $a = 3$ resulting in the string **baabcbcb**.

computational complexity, while providing satisfactory sensitivity and selectivity [7].

SAX transforms a time-series X of length n into a string of arbitrary length w , where $w \ll n$ typically, using an alphabet A of size $a \geq 2$. The SAX algorithm consist of two steps: during the first step it transforms the original time-series into a PAA representation and this intermediate representation gets converted into a string during the second step. Use of PAA at the first step brings the advantage of a simple and efficient dimensionality reduction while providing the important lower bounding property. The second step, actual conversion of PAA coefficients into letters, is also computationally efficient and the contractive property of symbolic distance was proven by Lin et al. in [11].

Discretization of the PAA representation of a time-series into SAX is implemented in a way which produces symbols corresponding to the time-series features with equal probability. The extensive and rigorous analysis of various time-series datasets available to the authors has shown that normalized by the zero mean and unit of energy time-series follow the Normal distribution law. By using Gaussian distribution properties, it's easy to pick a equal-sized areas under the Normal curve using look-up tables [12] for the cut lines coordinates, slicing the under-the-Gaussian-curve area. The x coordinates of these lines called "breakpoints" in the SAX algorithm context. The list of breakpoints $B = \beta_1, \beta_2, \dots, \beta_{a-1}$ such that $\beta_{i-1} < \beta_i$ and $\beta_0 = -\infty$, $\beta_a = \infty$ divides the area under $N(0, 1)$ into a equal areas. By assigning a corresponding alphabet symbol $alpha_j$ to each interval $[\beta_{j-1}, \beta_j)$, the conversion of the vector of PAA coefficients \bar{C} into the string \hat{C} implemented as follows:

$$\hat{c}_i = alpha_j, \text{ if } \bar{c}_i \in [\beta_{j-1}, \beta_j) \quad (3)$$

SAX introduces new metrics for measuring distance between strings by extending Euclidean and PAA (2) distances. The function returning the minimal distance between two string representations of original time series \hat{Q} and \hat{C} is defined as

$$MINDIST(\hat{Q}, \hat{C}) \equiv \sqrt{\frac{n}{w}} \sqrt{\sum_{i=1}^w (dist(\hat{q}_i, \hat{c}_i))^2} \quad (4)$$

where the $dist$ function is implemented by using the look-up table for the particular set of the breakpoints (alphabet size) as shown in Table I, and where the singular value for each cell (r, c) is computed as

$$cell_{(r,c)} = \begin{cases} 0, & \text{if } |r - c| \leq 1 \\ \beta_{\max(r,c)-1} - \beta_{\min(r,c)-1}, & \text{otherwise} \end{cases} \quad (5)$$

As shown by Lin et al., this SAX distance metrics lower-bounds the PAA distance, i.e.

$$\sum_{i=1}^n (q_i - c_i)^2 \geq n(\bar{Q} - \bar{C})^2 \geq n(dist(\hat{Q}, \hat{C}))^2 \quad (6)$$

	a	b	c	d
a	0	0	0.67	1.34
b	0	0	0	0.67
c	0.67	0	0	0
d	1.34	0.67	0	0

Table I: A look-up table used by the MINDIST function for the $a = 4$

It worth noting, that SAX lower bound was examined by Ding et al. [13] in great detail and found to be superior in precision to the spectral decomposition methods on bursty (non-periodic) data sets.

E. Symbolic approximation and indexing

As explained above, application of SAX to the single time-series results in its symbolic representation which is much shorter (reduced in the dimensionality) and easier to manipulate.

By following a sliding window sub-series extraction and SAX indexing technique described in detail by Lin et al. in [7] and Keogh et al. in [11], I have built a number of symbolic indexes for every time-series generated at partitioning step (III-B) combining following parameters for SAX transformation:

- three sizes for sliding window reflecting natural intervals of a week (7 days), two weeks (14 days) and a month (30 days);
- 4 PAA steps for a weekly window, 6 PAA steps for a bi-weekly window, and 10 PAA steps for a monthly window;
- 3 letters alphabet for weekly window, 5 letters for bi-weekly, and a 7 letters alphabet for monthly window.

These indexes were stored in the same relational database, organized and indexed in order to allow the fast retrieval of SAX words and their frequencies for a specific project, a contributor, a time-interval, a SAX parameters set, or any combination of these fields.

F. Behavioral portrait

Here I define a term of “behavioral portrait” of a contributor c as the set of all observed SAX words in her software artifact trail(s):

$$BP_c = \{(w_1, f_1), (w_2, f_2), \dots, (w_n, f_n)\} \quad (7)$$

where each pair (w_1, f_1) corresponds to the observed SAX word and its frequency. This portrait can be further specified by project, time-interval and SAX parameters set. Also it can be easily extended from the individual contributor to a team, whose “behavioral portrait” is a union set of “behavioral portraits” of team members.

G. Token-based distance metrics application to behavioral portraits

In my previous experiments I have measured the performance of three similarity metrics when applied to the behavioral portraits.

The first metrics I have tried is weighted by SAX Euclidean similarity distance defined for common to two behavioral portraits words:

$$D(S, T) = \sqrt{\sum_{S \cap T} (\text{MINDIST}(s_i, t_i) * \|F_{s_i} - F_{t_i}\|)^2} \quad (8)$$

where S and T are two behavioral portraits whose words are ordered by frequency.

The second metrics I have tried is the Jaccard similarity coefficient between two behavioral portraits S and T which is simply

$$J_\delta(S, T) = \frac{|S \cup T| - |S \cap T|}{|S \cup T|} \quad (9)$$

The third metrics I have tried is the *tf*idf* similarity which defined as a dot product

$$TFIDF(S, T) = \sum_{\omega \in S \cap T} V(\omega, S) \cdot V(\omega, T) \quad (10)$$

where

$$V(\omega, S) = \frac{V'(\omega, S)}{\sqrt{\sum_{\omega'} V'(\omega, S)^2}} \quad (11)$$

is a normalization of *tf*idf* (product of token frequency and inverse document frequency):

$$V'(\omega, S) = \log(TF_{\omega, S} + 1) \cdot \log(IDF_\omega) \quad (12)$$

where $TF_{\omega, S}$ is a normalized token frequency

$$TF_{\omega, S} = \frac{|\omega|}{|S|} \quad (13)$$

and IDF_ω is a measure of the general importance of the pattern among all users

$$IDF_\omega = \frac{|D|}{DF(\omega)} \quad (14)$$

where $|D|$ is cardinality of D - the total number of users, and $DF(\omega)$ is the number of users having ω pattern in their activity set.

While first two metrics demonstrated very poor performance in the clustering tests (discussed in the section III-H), the *tf*idf* similarity statistics performed very well and is presented in this work.

H. Clustering

As a universal tool for the exploration of derived behavioral portraits through their partitioning, and for assessment of the metrics' performance, I used hierarchical clustering. The k-means clustering was used in the validation of the class assignment and for general assessment of the validity of the approach.

Table II: Patterns observed within pre- and post-release behavioral portraits, their $tf*idf$ weights and sample, **not normalized** curves. (here $pre-x.x$ and $post-x.x$ rows of the upper table correspond to pre-release and post-releases of Android OS version $x.x$; columns of the table correspond to non-trivial patterns observed in all behavioral portraits; cells of the table contain $tf*idf$ weights computed for a particular SAX word in a particular behavioral portrait)

release	"bbac"	"abca"	"babc"	"bbba"	"bcaa"	"bcbb"	"ccaa"	"cbaa"	"bbcb"	"bbbb"	"bbbc"
post-2.0	0.63	0	0.63	0	0	0	0	0.39	0.24	0.06	0
post-1.0	0	0.93	0	0	0	0	0	0	0	0.09	0.36
post-1.5	0	0	0	0	0	0	0	0	0.79	0.61	0
pre-1.5	0	0	0	0.23	0.23	0.91	0	0.14	0.18	0	0.09
pre-2.0	0	0	0	0	0	0	0	0	0	1	0
pre-1.0	0	0	0	0	0	0	0.79	0	0	0.08	0.61
unnormalized sample curves corresponding to patterns											

Android kernel-OMAP hierarchical clustering stream ADDED_LINES, user mask ``*@google.com``

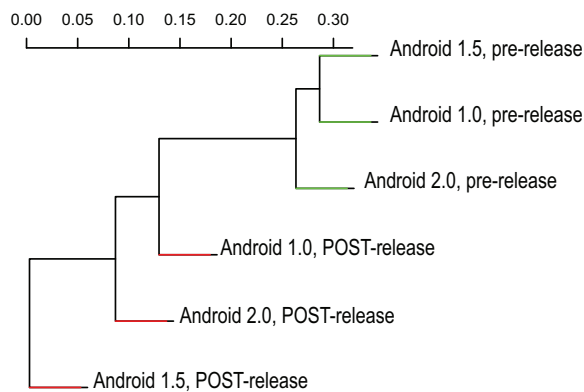


Figure 2: Hierarchical clustering of pre- and post-release behavioral portraits corresponding to the new code lines dynamics of google.com affiliated contributors.

IV. RESULTS

For the experiments related to this work I have tried a number of contributors partitioning schemes, variety of time-intervals and sub-projects selections observing satisfactory performance of investigated approach. However, due to the space constraint of this paper, I present only single validation experiment in this section as the proof of concept.

A. Kernel-OMAP life cycle patterns discovery

I have arbitrary selected the Android kernel-OMAP project as one of the large sub-projects in Android OS. It is the Android kernel implementation for OMAP-based (a proprietary system on chips based on ARM architecture processor by Texas Instruments) devices.

As a “training set” for discovery of behavioral portraits of pre- and post-release patterns, I chose three Android releases: *Android 1.0*, *Android 1.5 “Cupcake”* and *Android 2.0 “Eclair”*. For each of these I generated behavioral

portraits corresponding to four weeks before the release - *pre-release behavioral portrait*, and to four weeks after release - *post-release behavioral portrait* having in place an additional constraint on contributors and the artifacts trail. I have selected contributors affiliated with *google.com* e-mail domain only, expecting that paid developers will have much more consistent behavior [2]. By selecting the *added_lines* artifacts stream only, I additionally limited the scope of the analyzes and the complexity of captured behaviors to the “new code lines dynamics” only. The almost perfect clustering picture (Figure 2) obtained with hierarchical clustering and $tf*idf$ statistics as the distance function indicates, that there are significant differences in the pre- and post-release weekly behaviors of contributors in selected time-windows.

While hierarchical clustering is a good sanity test for the data exploration, the performance of K-means clustering is much more valuable [14]. I performed k-means on the symbolic representation of data using $tf*idf$ statistics and Euclidean distance. The algorithm converged after two iterations separating pre- and post-release dictionaries with a single mismatch for the Android 2.0 pre-release.

By using centroids of two resulting clusters as a basis for pre- and post-release patterns I tested the classifier on the rest of Android kernel-OMAP releases. The classifier was able to successfully classify more than 81% of pre- and post-release behaviors (Table III). When applied to the similar project - kernel-TEGRA - it demonstrated the error rate less than 15%.

The classifier demonstrated a weak, almost random performance on other sub-projects, such as user-interface related projects and e-mail client. However, when re-trained on the platform-external-bluetooth-blueZ project, its performance on other bluetooth-related sub-projects, such as platform-external-bluetooth-glib, platform-external-bluetooth-hcidump, and platform-system-bluetooth, recovered to 20% miss-classification.

Table III: Pre- and post-release development patterns classification results for kernel-OMAP.

Release	Classification	Release	Classification
1.6-pre	misclassified	beta -pre	OK
1.6-post	OK	beta-post	OK
2.2-pre	OK	2.0.1-pre	OK
2.2-post	OK	2.0.1-post	misclassified
1.1-pre	OK	2.1-pre	OK
1.1-post	OK	2.1-post	OK
2.3-pre	OK	2.2.1-pre	OK
2.3-post	OK	2.2.1-post	misclassified

B. Contribution

To the best of my knowledge, this work is the first attempt to study the applicability of symbolic aggregate approximation and term frequency-inverse document frequency weight statistics to the mining of software process artifacts. This methodology has a number of advantages. First of all, SAX facilitates significant reduction of the large complexity (dimensionality and noise) of temporal artifacts and opens the door to application of a plethora of string search and text-mining algorithms. In addition, the *tf*idf* statistics provides an efficient mechanism for discrimination of the signal by ranking symbolic data while focusing on dissimilarity. Finally, the third component I have used - the relational database - facilitates efficient data slicing, indexing, and retrieval.

As an example of a possible data-mining workflow demonstrating the resolving power and correctness of the approach, I presented a case study of building a classifier for pre- and post-release recurrent behaviors. Whereas this classifier demonstrates a good performance within the project it was trained on with less than 20% miss-classification, it has less than 15% miss-classification rate in similar Android OS kernel sub-projects.

V. DISCUSSION

The presented approach and workflow employs two novel techniques in order to discover and rank recurrent behaviors from software process artifact trails. While the approach demonstrates satisfactory performance, the interpretation of the captured behaviors requires more work. The discovered behavioral patterns are organized in Table II by their occurrence: the first three columns belong to the post-release time-window, the four next columns belong to pre-release time-window, while the rest are the behavioral patterns observed in both. The bottom row of the table contains plots visualizing examples of the raw-data streams corresponding to symbolic behavioral patterns. By the visual examination of these examples, it appears that during pre-release most of the added lines within a week fall on the Monday and Tuesday, whereas during post-release time, most of the lines are added during the end of the week and the week-end. While an explanation of these findings requires an additional study to be made, one of the interpretations of such behavior

could be based on the contributors employment profile. For example, if the coding activity of developers paid to work on Android (thus mostly commit during working days) has fallen below the activity of developers working on their own volition (who commit mostly off business hours); which, in turn, could be a consequence of removing of a pre-release code-freeze, or that the paid developers switched in post release period to design, documentation, or bug-fixing activities.

VI. ACKNOWLEDGMENT

I thank to Philip Johnson for his time, useful discussions, and comments.

REFERENCES

- [1] A. Hindle, M. W. Godfrey, and R. C. Holt, "Mining recurrent activities: Fourier analysis of change events," in *Software Engineering - Companion Volume, 2009. ICSE-Companion 2009. 31st International Conference on*. IEEE, May 2009, pp. 295–298. [Online]. Available: <http://dx.doi.org/10.1109/ICSE-COMPANION.2009.5071005>
- [2] J. Eyolfson, L. Tan, and P. Lam, "Do time of day and developer experience affect commit bugginess?" in *Proceedings of the 8th Working Conference on Mining Software Repositories*, ser. MSR '11. New York, NY, USA: ACM, 2011, pp. 153–162. [Online]. Available: <http://dx.doi.org/10.1145/1985441.1985464>
- [3] L. McLeod and S. G. MacDonell, "Factors that affect software systems development project outcomes: A survey of research," *ACM Comput. Surv.*, vol. 43, no. 4, Oct. 2011. [Online]. Available: <http://dx.doi.org/10.1145/1978802.1978803>
- [4] A. E. Hassan, "The road ahead for mining software repositories," in *Frontiers of Software Maintenance, 2008. FoSM 2008*. IEEE, Sep. 2008, pp. 48–57. [Online]. Available: <http://dx.doi.org/10.1109/FOSM.2008.4659248>
- [5] G. Antoniol, V. F. Rollo, and G. Venturi, "Linear predictive coding and cepstrum coefficients for mining time variant information from software repositories," in *Proceedings of the 2005 international workshop on Mining software repositories*, ser. MSR '05, vol. 30, no. 4. New York, NY, USA: ACM, 2005, pp. 1–5. [Online]. Available: <http://dx.doi.org/10.1145/1082983.1083156>
- [6] A. Hindle, M. W. Godfrey, and R. C. Holt, "Release Pattern Discovery via Partitioning: Methodology and Case Study," in *Proceedings of the 29th International Conference on Software Engineering Workshops*. Washington, DC, USA: IEEE Computer Society, 2007. [Online]. Available: <http://dx.doi.org/10.1109/ICSEW.2007.181>
- [7] J. Lin, E. Keogh, L. Wei, and S. Lonardi, "Experiencing SAX: a novel symbolic representation of time series," *Data Mining and Knowledge Discovery*, vol. 15, no. 2, pp. 107–144, Oct. 2007. [Online]. Available: <http://dx.doi.org/10.1007/s10618-007-0064-z>
- [8] T. Roelleke and J. Wang, "TF-IDF uncovered: a study of theories and probabilities," in *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, ser. SIGIR '08. New York, NY, USA: ACM, 2008, pp. 435–442. [Online]. Available: <http://dx.doi.org/10.1145/1390334.1390409>
- [9] B. K. Yi and C. Faloutsos, "Fast Time Sequence Indexing for Arbitrary Lp Norms," in *VLDB '00: Proceedings of the 26th International Conference on Very Large Data Bases*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2000, pp. 385–394. [Online]. Available: <http://portal.acm.org/citation.cfm?id=645926.671689>
- [10] E. Keogh, K. Chakrabarti, M. Pazzani, and S. Mehrotra, "Dimensionality Reduction for Fast Similarity Search in Large Time Series Databases," *Knowledge and Information Systems*, vol. 3, no. 3, pp. 263–286, Aug. 2001. [Online]. Available: <http://dx.doi.org/10.1007/PL00011669>

- [11] J. Lin, E. Keogh, S. Lonardi, and B. Chiu, "A symbolic representation of time series, with implications for streaming algorithms," in *Proceedings of the 8th ACM SIGMOD workshop on Research issues in data mining and knowledge discovery*, ser. DMKD '03. New York, NY, USA: ACM, 2003, pp. 2–11. [Online]. Available: <http://dx.doi.org/10.1145/882082.882086>
- [12] R. J. Larsen and M. L. Marx, *An Introduction to Mathematical Statistics and Its Applications (3rd Edition)*, 3rd ed. Prentice Hall, Jan. 2000. [Online]. Available: <http://www.worldcat.org/isbn/0139223037>
- [13] H. Ding, G. Trajcevski, P. Scheuermann, X. Wang, and E. Keogh, "Querying and mining of time series data: experimental comparison of representations and distance measures," *Proc. VLDB Endow.*, vol. 1, no. 2, pp. 1542–1552, Aug. 2008. [Online]. Available: <http://dx.doi.org/10.1145/1454159.1454226>
- [14] *Initialization of Iterative Refinement Clustering Algorithms*, 1998. [Online]. Available: <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.54.3469>

***GiftScroll* – An Online Gift Registry for the Android Mobile Phone**

Devon Simmonds, Michael Nipper
University of North Carolina Wilmington
601 S College Rd, Wilmington, Tel. (910) 962-3000
{ simmondsd, mjn4406}@uncw.edu

Abstract

Smart-phone ownership rates are on the rise and Android is one of the fastest growing mobile-phone operating systems. Developers can now take advantage of creating mobile applications that provide utilities that were never before possible. Various libraries available for the Android phone have been created, and provide a starting point for creating useful applications. One such application is ZebraCrossing, a barcode scanning application that returns the barcode type and barcode value to a developer. This information can then be sent to Google to receive an XML file containing information about the product, such as its name and description. Various functions built-in to Android can also help a developer create useful applications quickly, such as Android's location-aware technology. This paper reports on the development of an online gift registry for the Android phone using Zebra-Crossing and Android technologies. Results and lessons learned are presented.

Keywords: android, smart phone, online gift registry, software engineering

1. Introduction

Smart-phone ownership rates have sky-rocketed in the last few years, leaving an opening for developers to create applications which were never possible in the past [1, 2, 3]. The Android operating system, currently developed by Google, is the fastest-growing mobile operating system, and is expected to be a dominant player in the smart-phone operating system market for years to come [3, 4]. Many imaginative applications have already been created for Android, including applications which use the ZXing library created by Google to scan barcodes with the unit's camera [5, 6]. The Android phone's barcode scanning capabilities offer developers the ability to easily create useful applications for shopping [7].

One potential use for a barcode scanner on an Android phone is a gift registry program. A gift registry is essentially a "wish list" created by someone and distributed to their friends and family. They are often used for events such as weddings or baby showers [8]. Currently, to create a registry spanning several different stores is burdensome and difficult to communicate effectively [9]. Creators of gift registries also often miss the opportunity to include items from smaller, specialty stores which may not have the technology to create in-store gift registries.

In order to solve these problems, we created an application for the Android 2.* mobile phone operating system. The application uses the camera on the Android mobile phone for scanning barcodes, and stores the information in a database which is accessible both from a web-site and our Android application. Currently, there are not any identical projects in the Android marketplace. However, there are several applications which make use of the ZXing library, including ShopSavvy, Barcode Scanner, and Amazon's Android application [10]. Many of these applications, while useful, have limited capabilities. Several other novel applications which are more sophisticated are currently being developed using the ZXing technology, such as a system which scans different over the counter medications and warns consumers of potential allergies and adverse drug reactions [5].

2. Software Design and Development

2.1 Scope and Objectives

Our main objective was to create an application which is both intuitive and reliable. Therefore, our primary concerns were an easy-to-use user interface and data concurrency control. The user interface for the Android application should be intuitive enough to quickly learn, with a minimalist design principle with an emphasis on reducing clutter. The highly competitive Android application marketplace has no place for a cluttered design which is difficult to use. The website which has access to the data should also be designed

with simplicity in mind. Gift registries are often sent to a wide range of users with varying technical experience. Therefore, the website should be easily navigable to even the most elementary user.

Data concurrency control is also an important aspect of the project. When an item is added to the registry, there should be a check to make sure it is the correct item. When an item is purchased from the registry, it is also essential that it is appropriately marked both on the website and in the Android application. An error in concurrency control could result in a fatal flaw in the system, and each time an item is added or removed from the registry there should be a prompt, minimizing user-error.

The android application should be able to scan a barcode using the camera on mobile phone. The application should then be able to extract information about that product based on the barcode. The location of the item should be deduced using Android's location-aware technology. The information should then be stored in an online database, which will be accessible from both the Android application and a website. The creator of the registry should be provided with a means of sending out the registry, such as e-mail.

2.2 Application Design

Our design process began with a specification of the basic software features using a use case diagram (see Figure 1) and an activity diagram (see Figure 2) to outline the basic behavioral features of the software. These include adding and removing items to and from the registry respectively, along with the ability to edit and view the registry and purchase items.

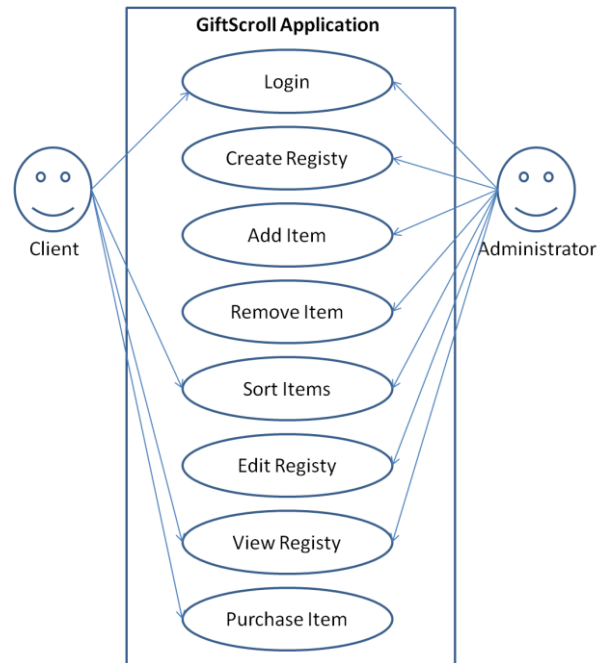


Figure 1: Use Case Diagram

ARCHITECTURAL DESIGN

The two Architectural Designs used for this project are the three-tiered diagram and the model view controller diagram (MVC). The MVC diagram is shown in Figure 3. The three-tiered diagram expresses the organized flow of the project by representing three layers—the user level, logic level, and data level. The MVC diagram shows the flow of the project displaying the interaction between the client and program, representing the data, functionality, and user interface. The Model View Controller Diagram (MVC) is used to represent the interaction between the user or client and the program, GiftScroll, or the server. The advantage of using MVC is to represent the relation between the data, functionality, and user interface of the program. Altogether, MVC shows the flow the program when being used by user.

Each subsystem focuses on a major role of the program. The client subsystem is responsible for creating and/or finding a users' login username and accepting the users' password. The user database subsystem is where the program's users' login usernames and passwords are stored. The registry subsystem contains the users' list, where they can create, edit, and export their registry list. The item subsystem represents the name, price, and description of an item previously added to a list. And finally the viewer subsystem is for the friends and family of the user, where lists can be viewed and items can be purchased and marked purchased.

Following architectural design, the subsystems were individually designed. This part of the design process centers around the development of class and sequence diagrams. The class diagram for the application is shown in Figure 4. The figure shows the classes and relationships organized by subsystems. For example, the Model subsystem is made up of the Registry and DBGateway classes.

by touch gestures, which does make the program non-operational if the user is not familiar with a touch screen environment or if the touch screen is non-responsive. Also, the user must have data access available to use the features of scanning items, the scanner finding the item from the database, and the item being listed in the registry. Being limited with time, exporting the registry will be done by email and text, but not social networks.

2.3 Design Constraints

Being an android application, GiftScroll will only run on the android platform. The program is operated

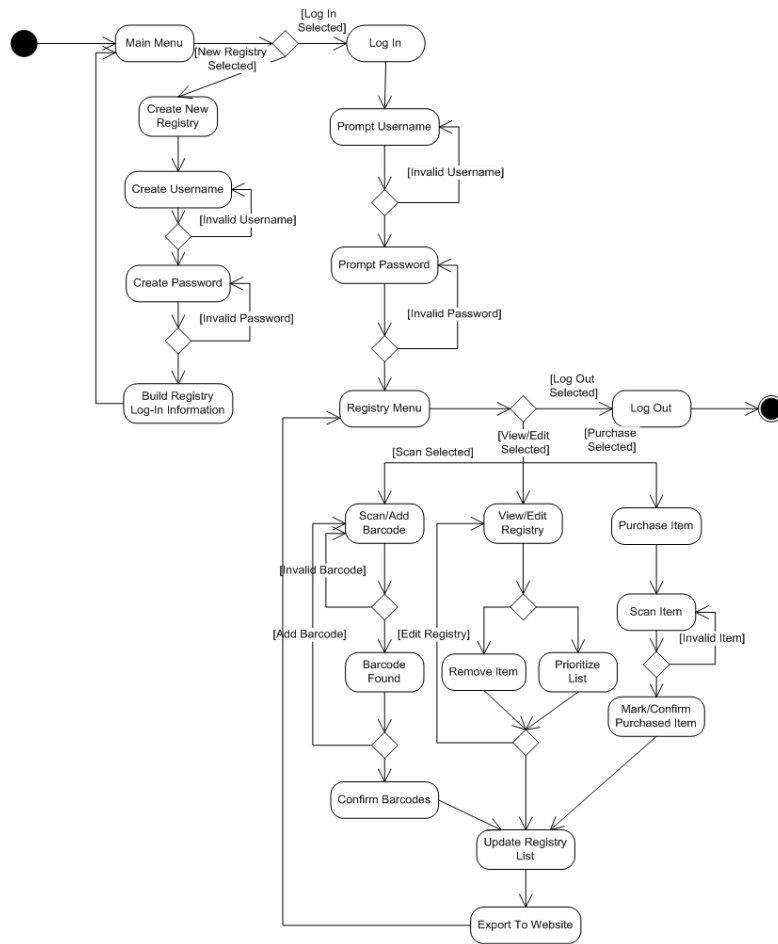


Figure 2: GiftScroll Activity Diagram

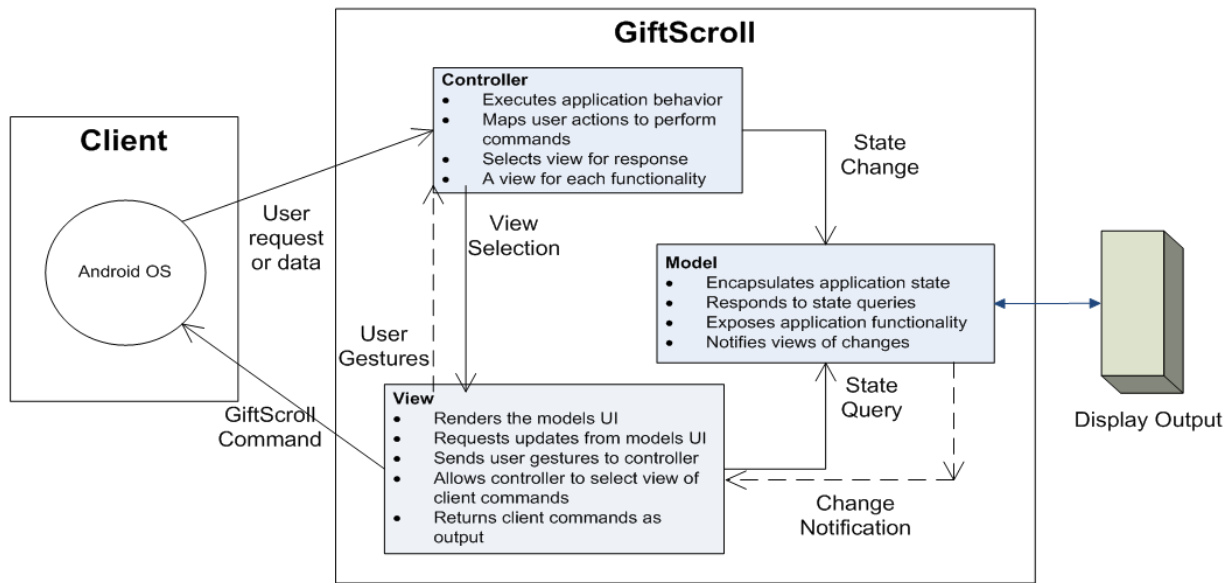


Figure 3: MVC Architecture

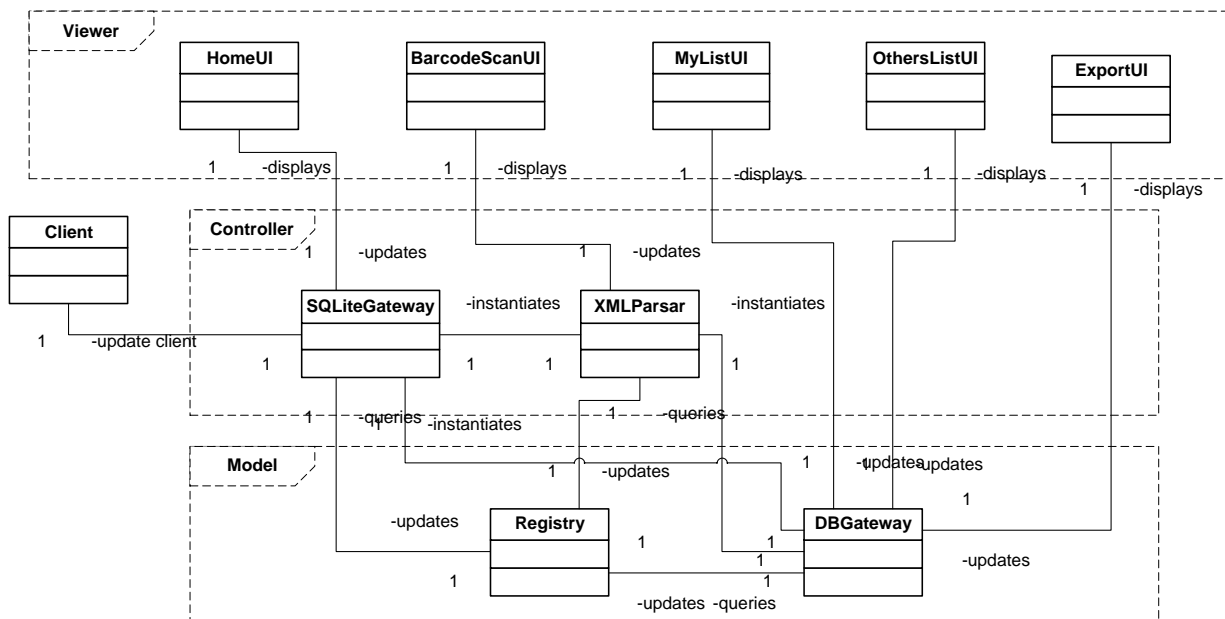


Figure 4: GiftScroll Class Diagram

3. Implementation and Testing

The application was developed and tested using a predefined test suite. Both the Android application and a corresponding web page were developed. A snapshot of the web page is shown in Figure 6. The main menu for the GiftScroll Android application is shown in Figure 5. Users have the options to create, view and edit

etting their preferences and using the barcode scanner when managing items.

The log in screen allows a registered user to log in to Giftscroll by entering their user name in the text field called user and then entering their password in the password field. Once a valid user name and password have been entered, press the log in button. For users that are not registered with gift scroll, the log in screen also provides the ability to register.

In order to register with Giftscroll, click the register button on the log in screen. New users will be asked to provide their First and Last name, their email address, as well as a username and password that are each between 6 to 10 characters long. Once all fields are completed click the register button at the bottom. A message should appear confirming that the registration is complete and asking you to please log in.

To add a new item to the list press the add item button. Once the add item is pressed GiftScroll will go into bar scanning mode, simply hold your Phone over the bar code of the item you wish to scan until scanner finish scanning. This is signified by the red line in the middle of the scanner turning green and the phone plays a sound. Once the bar code is scanned gift scroll will fill out the information for the newly added item. The user will be shown a screen with the information and asked to confirm that this is what they want to add to the list. Before confirming they user will be able change any of the information gift scroll has displayed.

To edit an item on a list press the edit button and select the item. Doing this should bring up the edit item screen, at this screen the user will be able to edit the name of the item, the items location, the items price, the quantity of the item that they want, and the items description. The user will also be able to mark if the item is purchased or not as well as delete the item from the list. To edit simple change the information and any of the text fields and click save, to delete simple click the delete button.

To send a list, click the send button under the edit button. Clicking the send button will bring up the send list menu. At this menu the user will be able to enter a message and send it to their friends by either email or text messaging. To send by text message simple click the text message button, select the contact phone numbers that you wish to send the message to, fill in the message, and click send. To send by email click the email button, select the emails that you wish to send the message to, fill out the message, and click send.

4. Discussion and Lessons Learned

There are ethical issues to take into account during this project. Unauthorized use of information, such as the distribution of user e-mail addresses and other personal information is not only unethical but is also against the law. Under no circumstances will personal information about users be sold or given to a third party without prior consent. There also will be significant effort to secure our database and web server space from being exploited by a malicious third party.



Figure 5: Main Menu

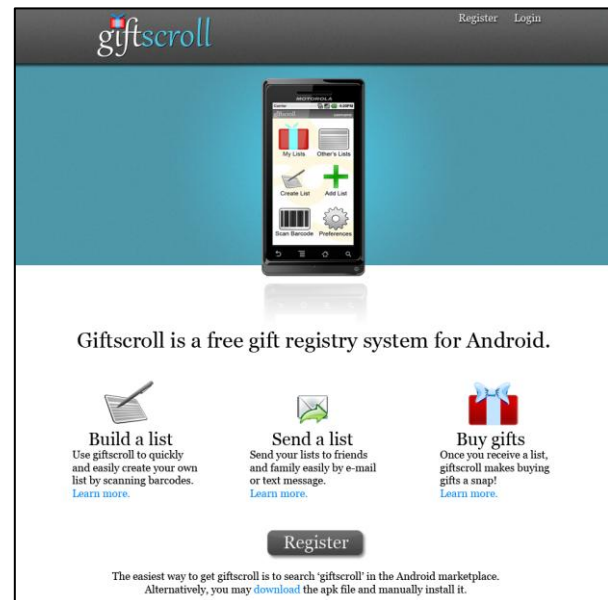


Figure 6 Index of the website.

One security measure taken includes using php scripts to execute MySQL queries and updates in the Android application instead of allowing direct access to

the MySQL database. Allowing direct access from the application would allow the potential for exploitation by a third party. In addition, all administrative areas of the website, as well as the FTP password, will be securely protected with strong passwords. Since the website is hosted on a shared server space, it is imperative that our passwords are secure both to protect our users' information and to protect the other information on the shared server.

Since our website will have the ability to update a MySQL database, significant effort will be put into preventing SQL injection attacks on the client-side before the information even reaches the database on the sever. Doing so is imperative to protecting our website users from executing unwanted scripts from malicious parties.

The only function that was not implemented was the location-aware technology. The code was written and worked correctly on most occasions, however on some occasions a glitch would happen when the application was not able to determine the location. We felt it was better to just leave this feature out until it could be properly addressed.

The primary performance issue expected will be that multiple users will be accessing a registry database at the same time. Another issue may be the limited computing power on most Android mobile phones. The performance of the Android application in conjunction with the Android OS must constantly be evaluated. Since this system does involve a web application, bandwidth is another issue that may hinder performance. The issue of bandwidth consumption must be in the foreground during the website design task.

REFERENCES

- [1] Brown M (2009) The NetGens 2.0: Clouds on the Horizon. *EDUCAUSE Review*, vol. 44, no. 1 (January/February 2009): 66–67.
- [2] Oliver, Earl. ACM SIGMOBILE Mobile Computing and Communications Review (2009). Vol 12 Issue 4: 56-63.
- [3] Xun Li, Pablo J. Ortiz, Jeffrey Browne, Diana Franklin, John Y. Oliver, Roland Geyer, Yuanyuan Zhou, Frederic T. Chong. A case for smartphone reuse to augment elementary school education. *GREENCOMP '10: Proceedings of the International Conference on Green Computing*.
- [4] Gadhavi, Bimal and Khushbu Shah (2010) Analysis Of The Emerging Android Market. San Jose State University: May 2010.
- [5] Jara, A.J et al. "A Pharmaceutical Intelligent Information System to detect allergies and Adverse Drugs Reactions based on internet of things." *Pervasive Computing and Communications Workshops (PERCOM Workshops)*, 2010 8th IEEE International Conference: 809 - 812. May 2010.
- [6] Kulyukin, Vladimir and Aliasgar Kutiyawala (2010): From ShopTalk to ShopMobile: Vision-Based Barcode Scanning with Mobile Phones for Independent Blind Grocery Shopping. *RESNA Annual Conference*. University of Utah: June 2010.
- [7] Fuchs M., Reichl P., and M. Baldauf. *Mobile Augmented Barcodes: Experiences with a Novel Mobile Barcode Scanner in the Wild*. Computational Intelligence: 2010.
- [8] Gillenson Mark L., Sherrell Daniel L., and Lei-da Chen. *Communications of the AIS: Vol 2. Issue 3* (1999). Information technology as the enabler of one-to-one marketing.
- [9] Robertson, Steven C. "System and Method For Providing Electronic Multi-Merchant Gift Registry Services Over A Distributed Network." Patent #: US 2005/0033650 A1. Feb 2005.
- [10] Zebra Crossing Project Wiki.
<http://code.google.com/p/zxing/wiki>
- [11] Kemerer, Chris F. (1993). Reliability of function points measurement: a field experiment. *Commun. ACM* 36, 2 (February 1993), 85-97. DOI=10.1145/151220.151230 <http://0-doi.acm.org.uncclc.coast.uncwil.edu/10.1145/151220.151230>
- [12] Lavazza, L. and Garavaglia, C. 2009. Using function points to measure and estimate real-time and embedded software: Experiences and guidelines. In *Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement (ESEM '09)*. IEEE Computer Society, Washington, DC, USA, 100-110. DOI=10.1109/ESEM.2009.5316018 <http://0-dx.doi.org.uncclc.coast.uncwil.edu/10.1109/ESEM.2009.5316018>
- [13] Harris, A. (2009). Publishing in JISE. *Journal of Information Systems Educators*, 7(1), 12-15.

Extending Java for Android Programming

Yoonsik Cheon

Department of Computer Science
The University of Texas at El Paso
El Paso, Texas, U.S.A.
ycheon@utep.edu

Abstract—*Android is one of the most popular platforms for developing mobile applications. However, its framework relies on programming conventions and styles to implement framework-specific concepts like activities and intents, causing problems such as reliability, readability, understandability, and maintainability. We propose to extend Java to support Android framework concepts explicitly as built-in language features. Our extension called Android Java will allow Android programmers to express these concepts in a more reliable, natural, and succinct way.*

Keywords: Android, domain specific language, framework, Java

1. Introduction

Android is an open-source and rapidly growing platform for developing applications running on mobile devices such as smartphones and tablet computers [1]. It consists of a Linux-based kernel, libraries, and a Java-compatible application framework. The application framework provides a semi-complete application that can be specialized to produce a custom application quickly [2]. It defines conventions for extending classes provided by the framework so that newly added, application-specific classes can interact correctly with the framework classes as well as themselves.

However, the Android framework has a steep learning curve; it takes a long time to learn and be able to use the framework effectively. Reliability is a more serious issue. The framework relies on conventions to implement framework-specific concepts like activities and intents (see Section 2). If these conventions are violated, an application may not work correctly. But, there is no automatic way of detecting such violations or enforcing the framework conventions.

In this position paper we propose a solution to the above problem. The key idea of our solution is to extend the Java programming language to support Android framework concepts like activities and intents as built-in language features by introducing a few new language constructs. The extension will allow one to not only express these concepts in a succinct and natural way but also check them automatically.

2. The Android Framework

Two fundamental concepts of the Android application framework are activities and intents. *Activities* are building blocks of Android programming in that an Android application consists of one or more activities. An activity is a single

```
public class MainActivity extends Activity {
    public void onCreate(Bundle savedInstanceState) {
        ...
        Intent i = new Intent("edu.utep.cs.GRADE");
        Bundle extras = new Bundle();
        extras.putString("name", "Joe");
        i.putExtras(extras);
        startActivityForResult(i, 0);
    }

    public void onActivityResult(int id, int o, Intent r) {
        if (id == 0 && o == RESULT_OK) {
            Bundle extras = r.getExtras();
            ... extras.getString("grade") ...
        } }
}

public class GradeActivity extends Activity {
    public void onCreate(Bundle savedInstanceState) {
        ...
        Bundle extras = getIntent().getExtras();
        if (extras != null) {
            String name = extras.getString("name");
            String grade = findGrade(name);
            Intent r = new Intent();
            Bundle extras = new Bundle();
            extras.putString("grade", grade);
            r.putExtras(extras);
            setResult(RESULT_OK, r);
            finish();
        } }
}
```

Fig. 1: Sample Android code

screen in an application with supporting code. At runtime, there is a stack of activities, each created for one unique screen of the user interface. An activity may invoke another activity. The invoked activity is pushed onto the top of the activity stack and becomes visible. It is popped from the stack when its execution is finished, making the previous activity to resume its execution. Android runs each activity in a separate process each of which hosts a separate virtual machine. This is to provide a sandbox model of application execution to protect the system and other applications from badly-behaved code. One consequence of this decision is that an activity cannot directly invoke another activity.

Android introduces another concept called an intent to combine and glue activities. An *intent* is a message to the Android system asking for performing a certain action on certain data. Upon receiving an intent, the system locates and starts an activity that can perform the requested action on the requested data. If an action requires additional data or returns results, they are piggy-backed on intents. In short, activities are invoked indirectly using intents, and intents are the core of the Android message system.

Figure 1 shows sample code illustrating the use of activities

```

public activity MainActivity {
    public void onCreate(Bundle savedInstanceState) {
        ...
        calls("edu.utep.cs.GRADE", "Joe")
            receiving(String grade) {
                ... grade ...
            };
    }
}

public activity GradeActivity {
    receives String name;
    provides String grade;

    public void onCreate(Bundle savedInstanceState) {
        ...
        String grade = findGrade(name);
        returns(grade);
    }
}

```

Fig. 2: The sample code rewritten in Android Java

and intents. It defines two activities, subclasses of the Activity class, and the first activity invokes the second. The framework method, `startActivityForResult`, invokes an activity by taking two arguments, an intent and a request code. The intent specifies the activity to be invoked along with optional arguments bundled as key-value pairs. The request code is an integer identifying a particular invocation. The main activity also defines a callback method, `onActivityResult`, to be invoked upon completion of the execution of an invoked activity. Because a single callback method handles all activity invocations, the request code—identifying the invocation—is provided as the first argument. As shown in the definition of the second activity class, results are returned by calling two framework methods, `setResult`, and `finish`. The `finish` method makes the control to return to the invoking activity and thus its callback method to get executed.

3. The Problem

The Android framework relies on conventions and programming styles to support the concepts of activities and intents, and this causes several problems.

- **Reliability.** Several factors contribute to this problem, including no parameter validation and no checking for framework conventions, e.g., overriding callback methods like `onActivityResult` and calling framework methods like `setResult` and `finish`. Manually bundling activity arguments is error prone, and missing definitions or statements may cause subtle errors that are often hard to detect and diagnose.
- **Readability.** The source code is not only verbose but also less readable, understandable, and maintainable. For example, the location where an activity is invoked and that of the results become available and used (`onActivityResult` method) are different. Note also that a single callback method handles the results of all invocations, resulting in error-prone case analysis code.
- **Learning curve.** Learning framework classes and their protocols—expected ways of using them, e.g., method overriding and calling—takes a time.

4. Our Approach—Android Java

The key to our approach is to extend Java to support Android framework concepts as built-in language features. For this, we introduce a few new language constructs for activity declarations and invocations. Figure 2 shows the sample code rewritten in our extended Java, called *Android Java*. Activities are now built-in language concepts like classes as indicated with the use of the keyword **activity**. As shown in the `GradeActivity` activity, an activity declaration may include optional parameter declarations, **receives** and **provides** statements declaring input and output parameters. The activity parameter declarations specify the signature of an activity—input arguments and return values along with their types. The **returns** statement is used to return from an activity with optional results. An activity is invoked using the **calls** statement that specifies the name of the activity to be invoked, along with activity arguments; an optional **receiving** clause specifies the code to handle return values.

By mapping framework concepts to programming language constructs, Android Java addresses all the problems described previously. An explicit declaration of activity parameters will enable us to perform parameter validation, either statically or dynamically. An activity invocation and return is expressed in a more concise, natural, and readable way. One only needs to learn a few new language constructs that explicitly support the concepts of activities and intents.

5. Discussion

Android Java may be implemented in several ways including preprocessing, compiling, and annotations. Preprocessing is the easiest and quickest way to implement Android Java. Android Java code can be translated to plain Java code by essentially converting (a) the **calls** statement to a `startActivityForResult` method call wrapped with an appropriate check for parameter validation and (b) the **receiving** clause to the `onActivityResult` method with dispatching code. An Android Java compiler may be built to produce virtual machine code directly, by extending an open-source Java compiler like OpenJDK and Eclipse. Yet another possibility is to translate or express Android Java constructs in Java annotations and write an annotation preprocessor; however, this requires support for statement-level annotations.

Although we considered only activities and intents, there are many other concepts and features of the Android framework that could also be explicitly supported in Android Java, and the general problem is to map these features—currently supported in framework conventions and styles—to built-in language constructs in Java-based, domain specific programming languages.

References

- [1] Google, “Android website,” <http://www.android.com/>.
- [2] R. E. Johnson, “Frameworks = (components + patterns),” *Communications of the ACM*, vol. 40, no. 10, pp. 39–42, Oct. 1997.

SESSION
COST ESTIMATION METHODS

Chair(s)

TBA

A Lightweight Test-Driven Approach to Test Indie Software Products

Yong Lai, and Hassan Reza
School of Aerospace Sciences
Department of Computer Science
University of North Dakota
Grand Forks, ND 58201, USA
reza@aero.und.edu

Abstract

In this work, we discuss the application of testing method to a typical independent (or indie) software development. Toward this goal, an experimental indie software ShoppingList is developed and tested using the proposed techniques. Some testing frameworks and tools are also utilized to facilitate the process. Based on our results we concluded that the test-driven development does have a positive effect on reducing the cost of indie software development..

Key Words: Test-driven Development, Extreme Programming (XP), Lightweight testing, JUnit, Jemmy, Indie Software Development and Testing.

1. Introduction

Comparing to well-established software testing method (e.g., white-box, black-box, etc.) used in the traditional software development methodologies (e.g., waterfall model, spiral model, etc.) indie software testing is a new concept and it is at its primitive stage. Informally, the indie software development refers to the independent development of software systems by very small team of talented but market savvy software developers having limited human and financial resources. Examples of indie software developers include independent game and mobile applications (or mobile apps) developers.

The front-end and back-end developments of these types of software require agilities and speed due to market short windows of opportunities. Traditional heavy weighted and plan driven software development process such as waterfall model are very expensive and hence are ill-suited for development of these types of systems for the following reasons: demands complete, consistent, and correct requirements, testing process takes place only after the software has been written, lack of systematic test case design and testing plan, combined debugging and testing approach, etc [4].

In this work, we propose a lightweight testing method and guidelines for indie software developers aiming to improve the software quality, simplify the testing process, and reducing the cost of testing. Toward this goal, we are applying our proposed method to a typical indie software application to examine its feasibility.

In what follows, we first describe the test-driven development method [1]. This discussion will assist us to effectively incorporate the methodology into our proposed software testing approach. Next, our lightweight testing management process is discussed. We then discuss our decision on programming language, testing framework and tool support. A model-view-controller [3] testing model is developed to depict the feasibility of our proposed testing process. More specifically, the feasibility of our method is examined using a simple indie software application known as *ShoppingList* that manages a list of groceries the user wants to shop.

2. Background and Related Works

Test-driven development (TDD) refers to a testing method that requires testing a requirement (or code) first and implementing that requirement (or code) when the test fails [1]. More specifically, TDD is an evolutionary and specification development method in which writing code and testing the corresponding code are carried out in interleaved fashion [1]. The approach originally introduced as part of agile development such as Extreme Programming [4] that often works with automated software testing such as Junit [10]. Test-driven software development process heavily relies on the repetition of a very short development cycle: first developer writes a failing test case describing the new functionality, then s/he writes the code to pass that test, and finally, s/he refactors (or improve) the new code according to acceptable standards [1,4]. Testing-driven development (TDD) is especially efficient in solving the prevailing problem by indie software developers consisting of small number of talented programmers with limited resources.

A test-driven development cycle includes the following steps [4]:

- 1) Identify new functionality and generating the corresponding test cases.
- 2) Write the code: Initially, the test case will fail and corresponding code must be written and implemented to pass subsequent test case.
- 3) Refactor the code: After all test cases passed successfully, we need to clean up the code along with the test cases as necessary.
- 4) Repeat: The previous three steps are repeated to push forward the whole development process until the whole program is written.

Figure 1 shows a typical test-driven development cycle.

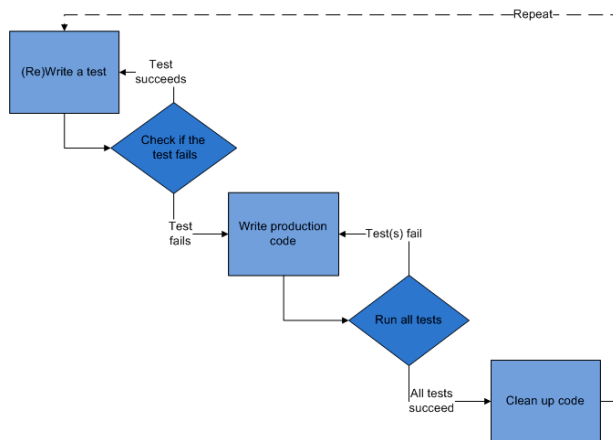


Figure 1. Test-driven development cycle.

3. A light-weight testing method

The well managed testing processes such as waterfall models have been proven to be very effective but very expensive in the development of commercial software product because the approach demands complete, consistent, and correct requirements before the validation and verification (V&V) begin [4]. The testing (static and dynamic) activities usually are normally performed by independent testing teams aiming at showing both correct and incorrect behaviors of the systems under test. Because of the cost and human resource attributed to these approaches, it is impractical to directly apply them in the indie software development. Therefore, it makes sense to adopt a lightweight testing process that shall be cheaper and yet effective for indie software developers with limited resources.

A lightweight methodology shall be a flexible development process using a handful of rules to manage the unstable requirements and environments. Examples of light-weight approach include adaptive software development [5] and extreme programming (XP) [6], feature driven development [7], etc.

In general, the testing management process includes

the following steps: 1) test planning, 2) test design, 3) test setup, 4) test execution, 5) test result analysis, and 6) test reporting. In our experiment, we simplified the task of each stage to create a light-weight testing management process. To further simplify the testing process, we have integrated the elements of lightweight testing process with the steps of test-driven development (TDD) as follows:

- 1) Test planning and test design. These two stages are integrated in the first step of test-driven development. Different from the test planning and test design stages in traditional testing management process, which usually requires the requirements to be complete into consideration. In addition, test planning and design at each cycle of test-driven development focuses on a small chunk of requirements. Since the test planning and design only consider a small part of the functionality of the software, a detailed test schedule is minimized or becomes unnecessary. Also, because of the small scale of indie software, other steps such as deciding test strategy, test requirements and items can be skipped in order to keep the testing management process simple and agile.
- 2) Test setup and test execution. For the test-driven indie software development, test setup and execution are related to writing product code to pass the newly added test case. Again, because of the small scale of indie software, test setup is trivial. After the product code is written, the new test cases and old test cases all need to be executed to ensure the new code doesn't introduce errors (regression testing). With the advancement of the development, test execution will be a tedious task as the number of test cases keeps growing. Normally, automated testing tools [10] are used to ease the task of test execution. The test entrance criterion is easy to define: when the new test cases are written. So as the test exit criterion: the new product code passes both the new test cases and old test cases.
- 3) Test result analysis and reporting. The task of this stage is to write test report, analyze defects, make changes, and prevent defects from reoccurring. In test-driven indie software development, this stage should happen before or after the third step. The most important task for indie developers at this stage is to document the test case design and test results including the effectiveness of test cases, whether new product code passes the old test cases, the choice of test data, etc. The output of this stage is very important as well since it provides insightful guidance for future software development.

A modified test-driven development incorporating a light-weight testing management process is shown in figure 2.

4. Testing tools and techniques

Besides testing methodology, testing tools also play an important role in the indie software development. Appropriate choice of testing tools can greatly enhance the productivity of indie developers because they can minimize the test cost by providing sophisticated testing frameworks and saving programmers' effort in creating, organizing, and executing test cases. For different programming languages and platforms, there are many different testing tools available for indie developers to choose. Some of them are open source, while others are commercial software. The factors influencing the choice of testing tools include but not limited to individual's styles, budget, tools, appropriateness to the project, etc.

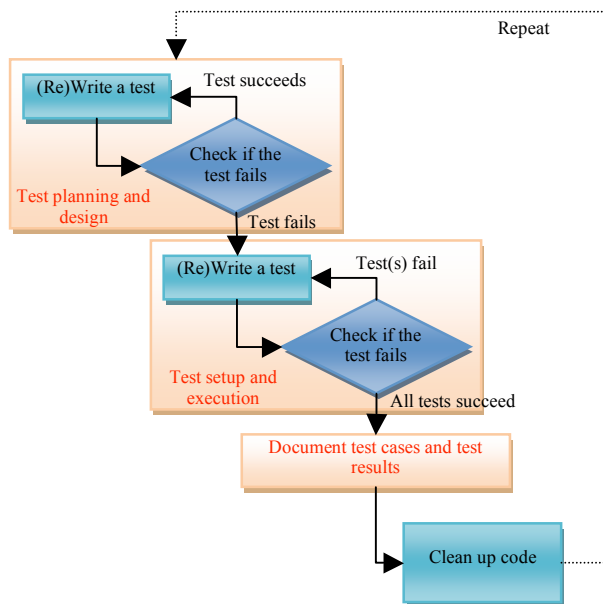


Figure 2. An Improved test-driving development cycle using light-weight

4.1 Programming language

We choose Java to implement our experimental project because as a programming language, Java is comparable with any other high level programming languages. Java has the advantage of being used as a prototype language. Also, as a mature programming language, Java enjoys the availability of many well-established frameworks and tools support used in test-driven development and test automation. More importantly, Java is representative among the programming language adopted by indie developers especially in the current trend of increasing

popularity of Android platform development and mobile apps.

4.2 Unit testing framework

We choose JUnit [10] as our testing framework. JUnit is a unit and regression testing framework; it is commonly considered as *de facto* tool for test-driven development that works with java language. The key justifications for using Junit include providing a complete assertion tools to validate actual result against expected results; generating Java test cases by API; and reporting and aggregation supports.

A high-level class diagram of JUnit framework is shown in figure 3.

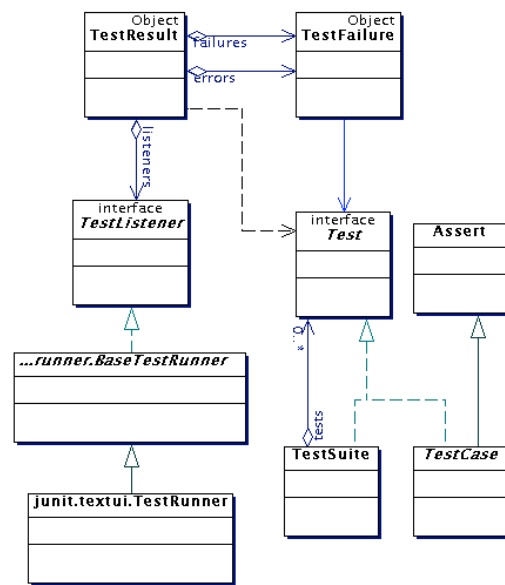


Figure 3. High-level class diagram of JUnit framework.

The core classes of JUnit framework include the following:

- **Test:** Test is the interface that all test cases must implement. In JUnit framework, two classes TestCase and TestSuite implemented the Test interface.
- **TestCase:** TestCase is the simplest type of Test. Most test case written by indie developers are going to extend this class. A concrete TestCase class contains methods (with signature testXXX) implementing individual tests as well as optional setup (for setting up test environment) and teardown (for cleaning up test environment) methods.
- **TestSuite:** Like TestCase, TestSuite also implements the Test interface. Like a container, TestSuite is used to collect together tests: TestCases, other TestSuites, or any combination of the two. The purpose of

TestSuite is organizing logically related testCases into individual groups.

- **Assert:** Assert is the superclass of TestCase that provides all kinds of assertion methods, which are used to decide if a test passes, or fails.
- **TestFailure:** TestFailures encapsulates an error or failure that occurs during a test run. It keeps track of the test cases that failed and the exception that was responsible for the error or failure.
- **TestResult:** It accumulates the results of running the tests and is responsible for informing the start and end of a test.
- **TestListener:** Any class that wishes to track the progress of a test run can implement this interface. It has methods for notification of the start and end of each test, as well as occurrence of errors and failures.
- **TestRunner:** TestRunner is used to run specified test cases. It expects the name of a TestCase class as argument. If this class defines a static suite method it will be invoked and the returned test is run. Otherwise all the methods starting with “test” having no arguments are run.

JUnit is not an automated testing tool: developers still have to write the test cases themselves. But by providing testing support library and runtime framework, JUnit does enable developers' write their test cases more conveniently, execute the test cases in batch, or run the test cases automatically with the help of Java Another Neat Tool (ANT) build script. At each test run, JUnit will report how many tests are executed, how many of the test methods in a test succeed, and how many failed. The detail of each failure will be reported in a form of a print of Java's stack track leading to the location of the failure.

4.3 Mock objects

Test-driven development (TDD) focuses on a small part of the functionality of the software one at a time. The direct result is numerous independent, small, but focused test cases. Some test cases are context sensitive and hence require system to be in the specific state. This makes it very impractical to set the system in a specific state to meet the requirement of each test case since it may greatly increase the number of test cases. The other problem is lack of proper coverage criteria to measure the adequacy of test cases when context is very important.

Mock objects are shown to be very effective when the objects are not currently available (e.g., Database). Mock objects are interpreted in three different ways [2]:

- **Stubs:** a class with methods that do nothing. They are created simply to allow the system to compile and run.
- **Fakes:** a class with methods that return a fixed value or values that can either be hardcoded or set programmatically.

- **Mocks:** a class in which testers can set expectations regarding what methods are calls, with what parameters, how often, return values for various calling situations, etc. It also provides a way to verify whether the expectations are met.

Despite the differences between the semantics of mock objects, they have a common target that is to take the place of real objects for the purpose of testing some functionality that interacts with and is dependent on the real object [2]. There are two major approaches to create mock objects: 1) coding them from scratch and 2) generating those using frameworks or tools.

A mock object written from scratch looks like a traditional class having some methods. It is just that the logic contained in the methods are greatly simplified to return desired value or performing simplified actions that a real object would do in place. Hand-coding all the mock objects can be a very time consuming task when the number of test cases is large.

There are many frameworks and tools have been created to make mock objects creation easier. EasyMock is one of such frameworks which generate mock objects at runtime using Java's proxy mechanism. EasyMock fits very well with test-driven development because the generated mock objects will not be affected by refactoring due to its unique style of recording expectations. Other similar frameworks and tools include MockMaker and MockObjects project.

4.4 Automated testing techniques

For indie software developers, two major automated testing techniques are used more often than others: batch test and automated GUI test.

4.4.1. Batch test. Batch test refers to the ability to execute a lot of test cases together [4]. It is usually supported by a testing framework. In our experiment, the batch test functionality is provided by the JUnit framework.

4.4.2 Automated GUI test. Most indie software involves intense graphical user interface (GUI) interaction with the user. Thus, GUI testing is inevitably an important part of the overall testing process. However, using test-driven development for GUI testing is tricky because automated testing of user interface is not as intuitive as other non-interface related testing. Fortunately, there are many automated GUI testing tools have been developed to help developers make the task easier.

In this work, we choose Jemmy to implement our automated GUI testing. Jemmy is a Java library that is used to create automated tests for Java GUI application. Using Jemmy, each component in the Swing API has a corresponding Operator class in the Jemmy API. An operator is used to wrap a corresponding Swing object on the GUI to be tested. The developer can specify the

behavior of each operator and wrap all these code in a single testXXX method of a TestCase class which can be executed by JUnit framework automatically.

4.5. A model-view-controller testing model

Model-view-controller (MVC) is a classic design pattern often used by indie and GUIs application developer which requires the ability to maintain multiple views of the same data [3][8].

To test the application designed using MVC pattern, we identify three classes of test cases corresponding to the essential elements of the MVC pattern, namely, test cases to test model, test cases to test controller, and test cases to test GUI. This test partitioning, in turn, allows us to organize test cases in manner that traceability between code implementing the requirements and test cases to validate these code can be established. Furthermore, it helps to simplify the testing process because each test group is independent of other test suites and hence reduces the duplicated test. This should increase the effectiveness of test cases to uncover possible software defects.

5. Case Study

A simple mobile application, namely, *ShoppingList* is used to show the feasibility of our proposed method. *ShoppingList* attempts to help the shopper to keep track of the groceries (items) she/he wants to shop. The main features of the program are as follows:

- It keeps a list of groceries items the user needs to buy. New groceries can be added to the list and existing groceries can be removed from the list.
- The name of each grocery in the list is unique. Attempt to add duplicated grocery to the list should fail.
- Each grocery in the list has a priority which helps the user decide which groceries are needed more than others. The priority of a grocery can be changed by the user.
- Groceries belong to different categories such as diary, baked food, vegetable, meat, frozen food, etc. The user can manage the category to accomplish tasks such as adding new categories, delete existing categories, or modifying categories. If a category containing groceries is deleted, those groceries' category property will be automatically changed to "uncategorized".
- The grocery list can be sorted by name, category, or priority. The order can be ascending or descending.

The project "ShoppingList" is developed using the Eclipse java development tool [11] in Java language. The application is designed using the MVC design pattern [8].

Using our proposed method, the first step is to identify small set of functionalities and corresponding test cases according to MVC. Figure 4 shows a screenshot of the project structure and the functionalities to be tested according to MVC.

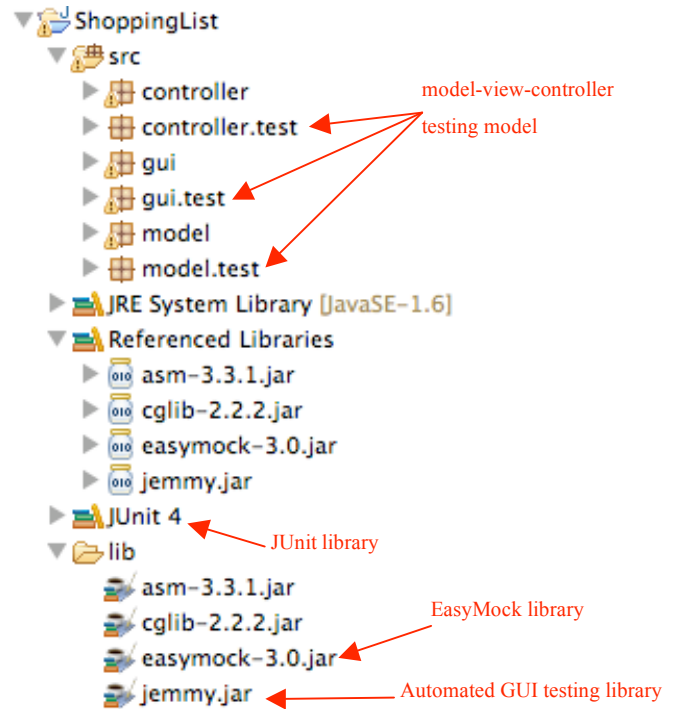


Figure 4. Project "ShoppingList" structure.

Step 2 requires to test setup and test execution environment. To this end, JUnit framework is used which allows multiple test cases be executed at one time (i.e., batch testing). Upon completion of step 2, the final step, test result analysis and reporting, is performed. The testing result will be provided immediately after the testing is done. Using this feature, we can test our test cases in different groups. For example, we can choose to execute all the test cases in the model testing group. After all the test cases in the model testing group passed, we can start executing all the test cases in the GUI testing group and leave the test cases in the controller testing group for the last. In this way, program defects are isolated and the errors in one group won't affect other groups.

Figure 5 shows a JUnit testing report for a successful test run. The listbox shows that there are total 4 test cases: TestEmptyGroceryList contains one test "testSize"; TestGroceryListWithOneGrocery contains one test "testSize"; TestGroceryListWithTwoGroceries contains two tests "testSizeAfterAddingTwo" and "testContents".

Figure 6 shows a JUnit testing report for a failed test run. It shows that the test "testSize" in TestEmptyGroceryList failed.

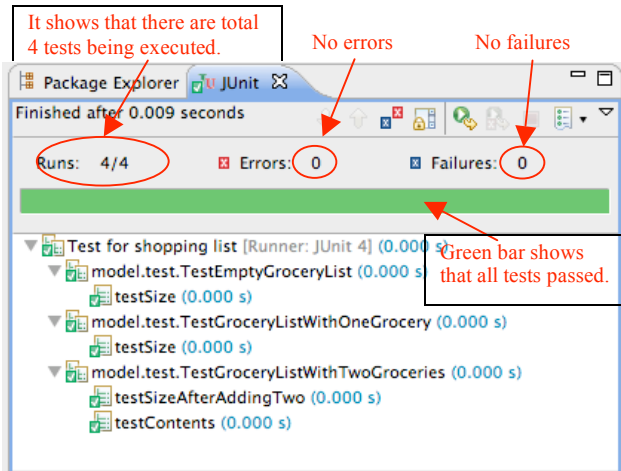


Figure 5. JUnit testing report for a successful test run.

5.2 Automated GUI test

In our experimental project, the automated GUI test is implemented by Jemmy framework [9]. In this work, we are going to test the functional features such as adding/deleting a grocery item by user interface. Figure 7 shows a simplified GUI.

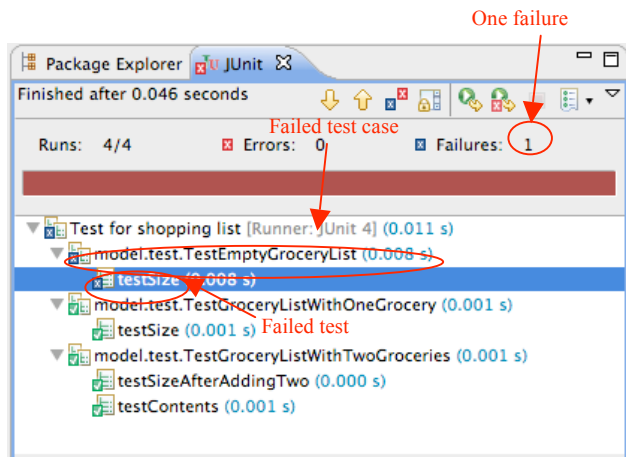


Figure 6. JUnit testing report for a failed test run.

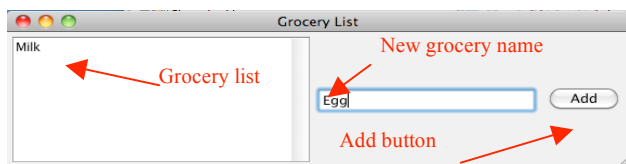


Figure 7. The simplified user interface (GUI)

The automated GUI testing code of adding new groceries using Jemmy is shown in figure 8.

```
public void testAdding()
{
    String LOST_IN_SPACE = "Lost In Space";
    Movie lostInSpace = new Movie(LOST_IN_SPACE);
    movies.add(lostInSpace);

    mainWindow = new org.netbeans.jemmy.operators.JFrameOperator("Movie List");
    MovieListEditor editor = new MovieListEditor(movieList,
        (SwingMovieListEditorView)mainWindow.getWindow());
    org.netbeans.jemmy.operators.JTextFieldOperator newMovieField =
        new org.netbeans.jemmy.operators.JTextFieldOperator(mainWindow);
    newMovieField.enterText(LOST_IN_SPACE);

    org.netbeans.jemmy.operators.JButtonOperator addButton =
        new org.netbeans.jemmy.operators.JButtonOperator(mainWindow, "Add");
    addButton.doClick();

    org.netbeans.jemmy.operators.JListOperator movieList =
        new org.netbeans.jemmy.operators.JListOperator(mainWindow);
    javax.swing.ListModel listModel = movieList.getModel();
    assertEquals("Movie list is the wrong size", movies.size(), listModel.getSize());
    for (int i = 0; i < movies.size(); i++)
    {
        assertEquals("Movie list contains bad movie at index " + i,
            movies.get(i),
            listModel.getElementAt(i));
    }
}
```

Figure 8. The automated GUI testing code using Jemmy.

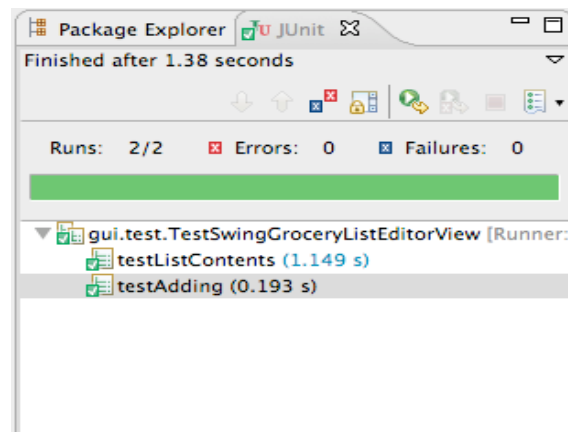


Figure 9. The testing report for adding new groceries.

Figure 9 shows the testing report for the automated GUI test. It shows that there is one test case in the test run. The test case contains two tests “testListContents” and “testAdding”. The green bar shows that all tests passed.

The console output of the testing execution is shown in Figure 10.

```

<terminated> TestSwingGroceryListEditorView [JUnit] /System/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Home/bin/jav
Trace:
Start to wait action "Wait Any javax.swing.JList loaded (ComponentOperator.WaitComponentTimeout)"
Trace:
"Wait Any javax.swing.JList loaded (ComponentOperator.WaitComponentTimeout)" action has been produce
: javax.swing.JList[,0,0,256x128,alignmentX=0.0,alignmentY=0.0,border=,flags=50331944,maximumSiz
Trace:
Start to wait frame "Frame with title "Grocery List"" opened
ComponentOperator wrapper
provided by Jemmy
Trace:
Frame "Frame with title "Grocery List"" has been opened in 0 milliseconds
gui.SwingGroceryListEditorView[frame1,42,22,576x164,invalid,layout=java.awt.BorderLayout,title=G
Trace:
Start to wait action "Wait Any javax.swing.JTextField loaded (ComponentOperator.WaitComponentTimeo
(ComponentOperator.WaitComponentTimeout)
Trace:
"Wait Any javax.swing.JTextField loaded (ComponentOperator.WaitComponentTimeout)" action has been pr
: javax.swing.JTextField[,285,57,206x28,layout=javax.swing.plaf.basic.BasicTextUIUpdateHandler,
Trace:
Start to wait action "Wait AbstractButton with text "Add" loaded (ComponentOperator.WaitComponentTim
Trace:
"Wait AbstractButton with text "Add" loaded (ComponentOperator.WaitComponentTimeout)" action has bee
: javax.swing.JButton[,496,56,75x29,alignmentX=0.0,alignmentY=0.5,border=com.apple.laf.AquaButton
Trace:
Start to wait action "Wait Any javax.swing.JList loaded (ComponentOperator.WaitComponentTimeout)"
Trace:
"Wait Any javax.swing.JList loaded (ComponentOperator.WaitComponentTimeout)" action has been produce
: javax.swing.JList[,0,0,256x128,alignmentX=0.0,alignmentY=0.0,border=,flags=50331944,maximumSiz

```

Figure 10. The console output of test automated GUI testing of adding new groceries.

6. Conclusions and Future works

Through our experiment, we found out that test-driven development does have a positive effect on indie software development. First, the introduction of testing at the early stage of software development helps developers to better understand the requirement specification and hence prevent design errors from being propagated into the later stage of development. This, in turn, decreases the development cost and hopefully increase the productivity.

Also, appropriate choice of testing framework and tools can help developers implement effective test automation and simplify the task of regression test.

In addition, a light-weight testing management process helps to enhance the test-driven development process because it enforces the regulation of test design, documentation, execution, and reporting.

Last but not the least, the development of innovative testing models should always be encouraged since it may help to simplify the testing process.

This work by no means is complete and therefore it warrants additional work and experimentations to validate our process. Therefore, future work includes, among other things, more case studies and experimentations.

References

[1]. K. Beck. Test-Driven Development by Example, Addison Wesley, 2003.

[2] Astels, D. (2003). Test-driven development: Practical Guide. Prentice Hall PTR. (2003).

[3] R Taylor, N. Medvidovic, E. Dashofy. Software Architecture, Foundation, Theory, and Practice. Wiley, 2010.

[4] I. Sommerville. Software Engineering, 9th edition. Addison wisely, 2012.

[5] J. Highsmith. Adaptive Software Development : Collaborative approach to development of complex, 1999. Systems.

[6] K. Beck. *Extreme Programming Explained: Embrace Change. 2nd edition.* Addison-Wesley, 2005.

[7] S. Palmer, S.R., and J. Felsing. *A Practical Guide to Feature-Driven Development.* Prentice Hall, 2002.

[8]. G. Krasner, S. Pope. A cookbook for using the model-view controller user interface isn Smalltalk-80. Journal of Object-Oriented Programming, 1(3):26-49, 1998.

[9]. Jemmy framework . <http://jemmy.java.net/>

[10]. Junit Framework. <http://junit.org/>

[11]. The Eclipse Java Development Tool. <http://www.eclipse.org/jdt/>

Establishing the Optimal Software Cost Equation Using Cost Affecting Factors and Elitist Gene Expression Programming

Divya Kashyap¹ and A. K. Misra²

¹ and ²CSED, Motilal Nehru National Institute of Technology, Allahabad

¹div.kashyap@gmail.com, and ²akm@mnnit.ac.in

Abstract- Accurate estimation of software's cost is one of the most intricate activities during the software development lifecycle. There are many factors that directly or indirectly affect the development cost of any software and these factors range from the size of the software to the experience of project manager. Calculating and interpreting these factors would be much valuable in software cost estimation if an ideal relationship between cost and these factors is found out. In this paper we have tried to identify some of the critical factors that affect the cost and we have also attempted to create a software cost equation that institutes the relationship between these factors and the software cost, using Gene Expression Programming. Our aim is to create an optimized model for cost estimation that includes least number of most acute factors for more accurate predictions. To settle this relationship, we have used Gene Expression Programming, a variant of Genetic Programming, because of its proven potential in evolving mathematical equations.

Keywords: Software Development Cost, Cost Affecting Factors, Genetic Expression Programming, Elitism, Function Points and Cost Estimation.

I. INTRODUCTION

Software cost estimation is one of the widely researched areas related to software engineering and software project management. It is the most onerous task which is perilous for its customers, developers and users because it severely affects the total software project management process, including contract negotiations, scheduling, resource allocation and project planning [1, 2]. Overestimation of cost may result in the failure to win the project contract or over employment of the resources while underestimating it may result in undeveloped functions, low quality and late deliveries. So it is a considerably important activity to envisage the software cost and its component units as early as possible in the development process so as to shape the future development plans [3]. One can also claim that the timely production of quality software also depends on this estimation.

Various cost estimation models have been described in the past and Boehm in [4] classifies these models into six categories: Parametric models, Expertise based, Learning oriented, Dynamic based, Regression based and Composite Bayesian. Out of these prototypes parametric or factor based models are the most prominent and precise. These are also

easy to understand and use [2, 3, 4]. Some of the factor based models are Putnam's SLIM [5], Function Point [6], Checkpoint [7], Price-S [8], Estimacs [9], Seer-Sem [10], SELECT-estimator [11] and COCOMO II [12, 13]. The traditional parametric model is envisioned in figure 1.

Laird & Brennan [15] have shown that notwithstanding the presence of several cost estimation models, only 28% of the total completed projects are delivered on time and each of these delivered projects can average over its budget by 45% or even more. Cost estimation is typical and critical because there are many factors that affect the software cost. Some of these factors are: Function size, duration, implementation language, development technique, developer's skills, project schedules, project manager's discretions and many more [6, 16, 17, 18]. Calculations that connect these parameters are hard because these parameters are ambiguous, uncertain and there is dearth of information about these factors during the initial phases of software development when we have to determine the cost. Also the rapid changes in user requirements make this process even more typical.

To tackle this imprecision and uncertainty we have used Gene Expression Programming (GEP) [19, 20] to successfully find an optimal relationship between the software cost factors. GEP is a special form of Genetic Programming [21, 22] in which each individual is a calculation unit or a mathematical equation. It is an optimization procedure used to optimize a set of mathematical expressions based on the fitness behavior of the expression, determined by its ability to perform a given computational task. In our paper the software cost equation is modeled as a mathematical expression and its accuracy in estimating the cost would be considered as its fitness. To improve the accuracy, we have combined Genetic Expression Programming with Elitist strategy [23] which makes it computationally fast and more accurate. The flow of rest of the paper goes in the following manner: In section two we have described the various factors that affect the software cost in section one. Then in third section we briefly discussed elitist strategy and gene expression programming. Finally, in section four we have framed the problem and tried to approach it using elitist gene expression programming with results and conclusions.

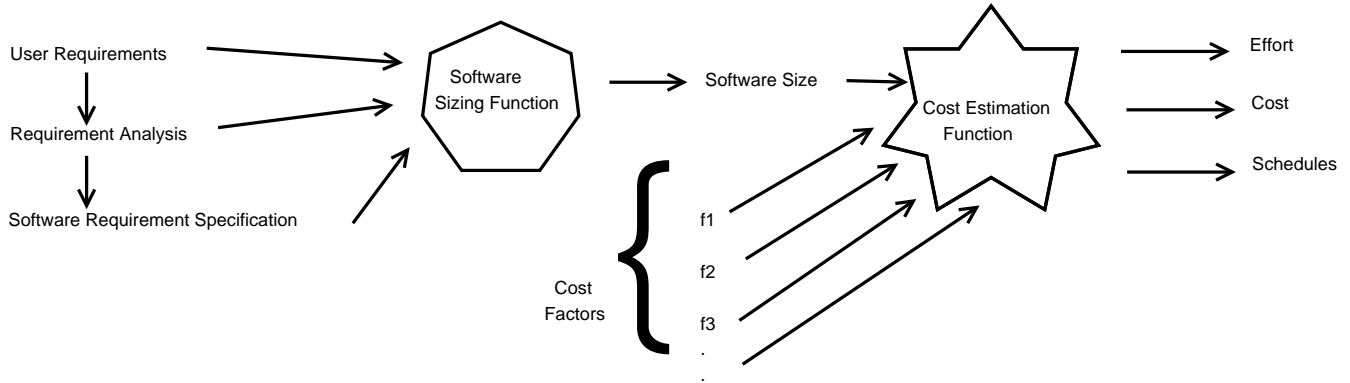


Figure 1: A Typical Parametric Cost Estimation Model

II. FACTORS AFFECTING SOFTWARE DEVELOPMENT COST

Finding the software cost function is the most arduous task in the field of software engineering, since there are many factors that simultaneously and synchronously affect the software development activities during the whole development life cycle. We have analyzed some hundred projects from ISBSG data repository [33] to find out the most prominent factors, so as to quantify the effect of these parameters on the software cost, using Gene Expression Programming. The following factors are the prerequisite for cost estimation:

A. Function Points (real number)

Function Points (FP), founded by Albrecht in [24], is the measurement of the functionality of any program. FP can be calculated from the detailed software requirements. The five categories of functions to be count are: *external inputs, external outputs, external inquiries, external interfaces and internal files*. After the functions of all categories (x_j) have been found out, with their complexity, we calculate the Unadjusted Function Points (UFP) [25], using the following table 1 and equation 1.

TABLE I
FUNCTION COUNT WEIGHTING FACTORS

Function Type	Low	Average	High	Total
External inputs	--*3+	--*4+	--*6=	---
External outputs	--*4+	--*5+	--*7=	---
External inquiries	--*3+	--*4+	--*6=	---
External interfaces	--*5+	--*7+	--*10=	---
Logical internal files	--*7+	--10+	--*15=	---
Total Unadjusted Function Points				---

$$UFP = \sum_{i=1}^n \sum_{j=1}^5 w_{ij} * x_{ij} \quad (1)$$

w_{ij} and x_{ij} are the weighting factor and function count respectively, in the i^{th} row and j^{th} column.

B. Software Quality Factor (real number)

The software quality is the degree to which the software meets the client's expectations. To calculate the Software

QualityFactor (SQF), we have used the following quality attributes,

described by McCall's in[26]: *Correctness, Usability, Efficiency, Reliability, Integrity, Portability, Reusability, Maintainability, Flexibility and Testability*. Although it is very difficult to directly quantify the quality attributes of any system, SQF can be estimated by combining the ratings for the individual quality factor [27] as described in following equation 2.

$$SQF = c_1 m_1 + c_2 m_2 \dots \dots c_n m_n \quad (2)$$

c_n are regression coefficients, m_n are the metrics that affect the quality factor calculated on a grading scheme ranging from 0 (low) to 10 (high).

C. Project Duration (natural number)

It is the total number of workdays for which the project has been developed. Cost is directly proportional to project duration.

D. Team Size (natural number)

Total number of developers who functioned conjointly throughout the entire development process, for the successful completion of the project. Cost increases with the size of team.

E. Number of Consultants (natural number)

It is the number of advisers or experts associated with the project. Project cost increases with the number of counselors.

F. Risk Classification

The overall risks and contingencies associated with the development process as well as the project itself.

G. Explicit Constraints

The explicit constrictions levied by the client or developers, such as, schedule constraints, execution time limitations, resource constraints or storage constraints.

H. Team's Experience and Ability

The assessment of skills, expertise and capabilities of the programmers in handling the project of particular type. Greater

the expertise and ability, the lesser will be the cost. This factor also includes the project team's cohesion.

I. Development Platform

This is the primary platform for the development of the software. Usually, each software project is developed for one of the following platforms: multi-platform, mainframe, midrange or personal computer. The cost decreases from multi-platform to personal computers.

J. Use of Modern Programming Practices

The degree to which modern software tools and programming practices are used in the development of software project. This factor also specifies the type of development language use in the process. Usually using the 4GL language and other development tools such as IDEs increases the implementation efficiency. It may increase or decrease the cost. An increase in the cost would be noticed if development tools are very costly or cost may decrease if the use of tools or particular programming language saves time and effort.

The factors from *H* to *Lare* measured at a scale of 1 to 5, ranging from 1 (very low) to 5 (very high). As one common characteristic of all parametric models is that the quality of outputs rest on the quality of inputs i.e. accuracy of cost estimation directly depends on the preciseness of input parameters and the relationship settled between there discrete parameters and the software cost [14], so care must be taken to be more and more precise while calculating these parameters.

III. GENE EXPRESSION PROGRAMMING AND ELITISM

A. Gene Expression Programming

Gene Expression Programming (GEP) is a variant of Genetic Programming [28]. It is a subset of Evolutionary Computation [29, 30] that automatically evolves and creates, computer ably programmable units. These units can be conventional mathematical expressions or complex polynomial structures [31]. In GEP each computational unit or mathematical expression is encoded as a linear chromosome (genotype) which can be translated into expression trees (phenotypes). There is a one-to-one relationship between the symbols used in the linear chromosome and the nodes of the corresponding expression tree.

B. GEP Chromosome Structure

A GEP chromosome is made up of functions and terminals and has two parts, *head* and *tail*. Head is made of both terminals and functions, while tail contains only terminals. The tail length *t* is calculated through the following formulae:

$$t = h(n - 1) + 1(3)$$

Where *h* is the head length and *n* is the number of distinct terminals. The tail, randomly formed, is a noncoding region which has nothing related to the expression but it plays a crucial role in the evolution. To understand, how GEP can be

effectively utilized to code for mathematical equations and how genetic operators can be applied over them, let us considering the structural organization of GEP using two expressions, *exp₁* and *exp₂*.

$$exp_1: a + ((b - d) * c) (4)$$

$$exp_2: (a * (b^d)) - c (5)$$

In context to *exp₁* and *exp₂*:

Set of functions, $F = \{+, -, *, \wedge\}$,

Set of terminals, $T = \{a, b, c, d\}$,

$$h_1 = h_2 = 7 \text{ and } t_1 = t_2 = (7(4-1) + 1) = 22.$$

$$exp_3: + a * - c b d (6)$$

$$exp_4: - * c a \wedge b d (7)$$

These expressions, *exp₃* and *exp₄* are known as *Karva* notation or the *K-expression* of the corresponding expressions: *exp₂* and *exp₃*. *K-expressions* are straight forwardly formed by reading the expression tree from left to right and top to bottom. The trees corresponding to *exp₁* and *exp₂* are shown in figure 2.

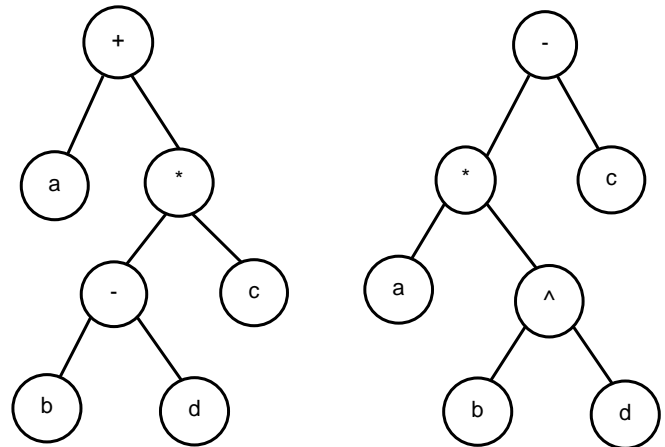


Figure 2: Expression trees corresponding to *exp₁* and *exp₂*

GEP is simple and elegant. The structures of genotypes allow the genetic operators to be applied over it very easily. Secondly they have a perfect and one-to-one mapping from genotype to phenotype which makes it easier and faster to quickly determine the fitness of any chromosome. Thus GEP efficiently completes the prerequisites of any evolutionary system.

C. GEP Genetic Operations

Due to the simplicity of GEP any selection scheme, roulette-wheel selection or tournament selection can be applied on GEP chromosomes. The three kinds of recombination operators for GEP are: single point, two point and gene recombination for any GEP chromosome. Chromosomes can be replicated through mutation also. For more detail on GEP genetic operators we redirect the reader to [19]. Sample recombination and mutation for *exp₁* and *exp₂* are shown in

table II, here exp_3 and exp_4 are the offspring and exp_5 and exp_6 are the mutants formed from exp_1 and exp_2 .

TABLE II
CROSSOVER & MUTATION FOR GEP

exp_1		exp_2		exp_3		exp_4		exp_5		exp_6	
HEAD	+	-	+	-	+	-	+	-	+	-	
	a	*	a	*	a	*	a	*	a	+	
	*	c	*	c	*	c	*	c	*	c	
	-	HEAD	-	HEAD	-	HEAD	-	HEAD	-	HEAD	
	c	^	c	^	c	^	c	^	c	^	
b	b	b	b	b	b	b	b	b	b	b	
d	d	d	d	d	d	d	d	d	d	d	
Crossover at 5 th Position				Random Mutation							
TAIL	a	b	a	b	a	b	a	b	a	b	
	b	c	b	c	b	c	b	c	b	c	
	a	d	a	d	a	d	a	d	a	d	
	b	a	b	a	b	a	b	a	b	a	
	a	a	a	a	a	a	a	a	a	a	
	c	b	c	b	c	b	c	b	c	b	
	d	b	d	b	d	b	d	b	d	b	
	d	a	d	a	d	a	d	a	d	a	
	c	d	c	d	c	d	c	d	c	d	
	a	c	a	c	a	c	a	c	a	c	
b	c	b	c	b	c	b	c	b	c		
c	d	c	d	c	d	c	d	c	d		
c	a	c	a	c	a	c	a	c	a		
b	b	b	b	b	b	b	b	b	b		
a	d	a	d	a	d	a	d	a	d		

D. Elitism

Elitism or elite preservation is a technique to retain the best solutions during subsequent generation of any evolutionary algorithm. There are two general approaches to inculcate elitism in the evolutionary process. One is to combine the parents and offspring of the previous generation before forming the mating pool for the current one. The other way is to maintain an archive to which the promising solutions at each generation can be copied [32]. In our paper also, we have applied the elitist strategy by maintaining an archive. Since the memory resources are limited, for both the variants, we select and hold only a specific number of elites. Elitism makes any evolutionary algorithm fast and efficient because it considerably reduces the loss of good solutions.

IV. PROBLEM DESCRIPTION AND SOLUTION APPROACH

Hitherto, various cost estimation models are present in the software industry that either calculates cost as the function of cost drivers or they simply model the cost estimation as regression problem to construct the expression which best predicts the cost, combining all input parameters. This

approach is known as metric based software cost estimation since we consider various metrics/factors to predict the software cost. The three major disadvantages of using this terminology are:

- 1) Because of inherent dynamism in the cost affecting factors, it is difficult to select the appropriate factors for cost estimation of any particular project. Once we are able to locate them, it is even more difficult to find their values in the early phases of development.
- 2) We have to analyze a lot of test cases to achieve a certain level of accuracy in the constructed expression or otherwise our predictions would be far from the reality.
- 3) The rapid changes in the Information Technology and software development methodology and processes have made it difficult to stabilize a particular estimation model.

Although, we are also succeeding in the same direction (metric based) for cost estimation, we are unlike from the previously proposed models, in our approach. We have tried to assuage these three problems.

First we have found out the least set of most affecting factors. From the available pool of factors we have selected only those factors which are not only the most dominating ones but these factors are easy to calculate also. This set of factors is briefly discussed in section 2 of this paper. Thus we have condensed the problem one.

Secondly, instead of using the conventional regression tools or simple genetic programming, we have used elitist Gene Expression Programming (GEP), described in section 3 of this paper. GEP has a proven potential for setting and optimizing the complex mathematical equations. It is simple and has directly mapped genotypes and phenotypes. Its operators are fast and effective. Also a very little knowledge is required by this heuristic method. Thus we overcome the second problem.

Thirdly, in our factor set we have considered quality factors, explicit constraints, development platform and use of modern programming practices. All these factors will clearly indicate the characteristics of the development methodology. So the third problem is also tapered.

In the proposed methodology, we have considered ten parameters to set up a cost estimating equation. One of the most prominent factor is function point, it itself requires some pre-calculations on the basis of user requirements, so as to be used in the estimation equation directly. The others factors can also be easily calculated from the users requirements and project manager's strategies. To connect these parameters we have used eight simple mathematical operators, namely, addition, subtraction, multiplication, division, exponent, log, square and cube. Now we have to establish the relationship

between the factors using these mathematical operations, so as to form the cost equation and we have used elitist GEP for this purpose. To start the elitist GEP iterations we randomly form a pool of five hundred initial equations. These initial and the consequent equations get processed through thousand iterations. This processing includes binary tournament selection, single point head-tail crossover and random mutation. This procedure eventually results in final expressions or cost equations (elite archive) that minimizes the Mean Magnitude Relative Error (MMRE) [34], described as equation 8. MMRE is used as a benchmark to measure the performance of the model, we have proposed.

$$MMRE = \frac{1}{N} \sum_{i=1}^N \frac{|actual_effort_i - estimated_effort_i|}{actual_effort_i} \quad (8)$$

N is the total number of fitness cases. $actual_effort_i$ and $estimated_effort_i$ represents the value of actual effort and model calculated effort respectively for the i^{th} project. Both $actual_effort$ and $estimated_effort$ are measured in the *person-months*. The cost in INR or dollars is directly proportional to this effort unit, i.e. the more *person-months* the more would be the cost. This whole optimization procedure, based on elitist Gene Expression Programming is briefly described in algorithm 1 of this paper.

VI. RESULTS

We set the experiment table with 30 projects, out of which two third i.e. 20 are used to for learning and the remaining 10 are used for testing. Thus the value of variable N is 20 in equation 8. Algorithm 1, gives a set of 50 cost equation, so to be more precise in our test results, we calculate the resultant estimated effort as the average effort calculated from these 50 equations. We compared our model i.e. Elitist GEP (EGEP) with PSO based COCOMO model (PC), Artificial Neural Network approach (ANN) and simple Genetic Programming (GP). From the results in figure 3, we can conclude that EGEP works well in most of the cases with minimum MMRE. The more interesting and appealing results are related to the time

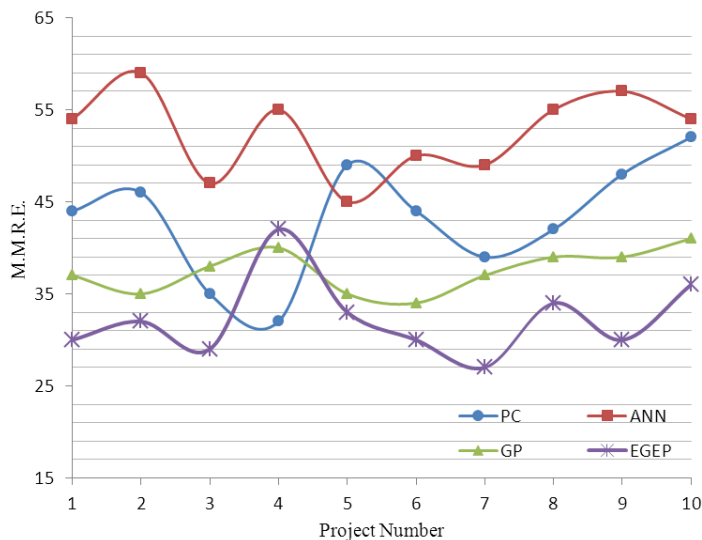


Figure 3: MMRE for different models complexity of the algorithm i.e. EGEP not only competes other algorithms in terms of MMRE but also in terms of total time taken to learn and subsequent predictions. The average time complexity comparisons of EGEP with PC, ANN and GP are shown in figure 4.

Algorithm 1: Elitist Genetic Expression Programming (GEP)

Inputs:

Parameters:	Values:
Terminals:	All factors discussed in section 2.
Functions:	{+, -, *, /, ^, log(x), x^2, x^3}
Fitness function:	Mean Square Error (MSE).
Population:	500.
Size of elite archive (s):	50.
Total generations (m):	1000.
Selection operator:	Binary tournament selection.
Crossover operator:	Single point.
Mutation operator:	Random single point.
Crossover probability:	90%
Mutation probability:	10%

Output:

Set of 50 best cost equations.

Procedure:

- Step 1.** Generate an initial population P_0 and create the empty archive A_0 . Set $t=0$.
- Step 2.** Calculate the fitness of the individuals in P_t and A_t , using genotype (k-expression) to phenotype (expression tree) conversion and equation 8.
- Step 3.** Copy the best s solutions in P_t and A_t to A_{t+1} .
- Step 4.** If $t \geq m$ return A_{t+1} .
- Step 5.** Apply selection operator to P_t to form the mating pool.
- Step 6.** Apply crossover and mutation operator on the mating pool to form P_{t+1} . Set $t = t+1$ and go to Step 2.

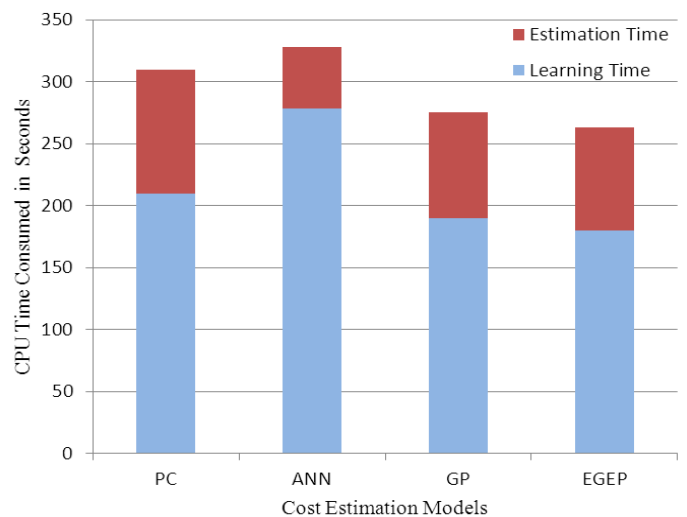


Figure 4: Time complexities for different models

VII. CONCLUSIONS

Cost estimation is a hard-hitting topic related to software industry. Although, many cost estimation models have been proposed till date, there is a continuous requirement of new techniques to predict the cost because of rapid change in the domains of information technology and software development methodology itself. We always necessitate a model that absorbs all the factors associated with the project and fallouts with a precise development cost estimate. In the light of this issue, our research in this paper gives an overview of the state of art of software cost estimation using principles of evolutionary computation.

We have tried to find out the most dominating parameters that potentially affect the cost of any software project and then we have proposed a methodology to set up a cost equation containing these parameters. Gene expression programming, which is a special case genetic programming, is used to establish the software cost equations between the software cost and the factors that affect it. From our experiment we have found that the equations found from Elitist Gene Expression Programming are the most accurate ones when compared to other traditional techniques.

VIII. REFERENCES

- [1] A. Nolan and S. Abraham, "Dealing with Cost Estimation in Software Product Lines: Experiences and Future Directions", *Software Product Lines: Going Beyond*, LNCS, Springer Berlin, pp. 121-135, 2010.
- [2] H. Leung, Z. Fan, "Software Cost Estimation", *Handbook of Software Engineering*, Hong Kong Polytechnic University, 2002.
- [3] R. Gray, S. G. MacDonell, and M. J. Shepperd, "Factors systematically associated with errors in subjective estimates of software development effort: The stability of expert judgment", *IEEE 6th Metrics Symposium*, IEEE Computer Society, Los Alamitos, CA, USA, pp. 216-227, 1999.
- [4] B. Boehm, C. Abts, and S. Chulani, "Software development cost estimation approaches — A survey", *Technical Report 2000-505*, Univ. of California and IBM Research, Los Angeles, USA, 2000.
- [5] L. Putnam and W. Myers, "Measures for Excellence", Yourdon Press Computing Series, 1992.
- [6] A. J. Albrecht, "Measuring Application Development Productivity", *Joint SHARE, GUIDE, and IBM Application Development Symposium*, pp. 83-92, Monterey, CA, 14-17 October 1979.
- [7] C. Jones, "Applied Software Measurement", McGraw Hill, 1997.
- [8] R. Park, "The Central Equations of the PRICE Software Cost Model", 4th COCOMO Users' Group Meeting, November 1988.
- [9] H. Rubin, "ESTIMACS", IEEE, 1983.
- [10] R. Jensen, "An Improved Macrolevel Software Development Resource Estimation Model", *Proceedings of 5th ISPA Conference*, pp. 88-92, April 1983.
- [11] SELECT (1998), "Estimation for Component-based Development Using SELECT Estimator", *SELECT Software Tools*, 1998. Website: <http://www.selectst.com>.
- [12] B. Boehm, "Software Engineering Economics", Prentice Hall, 1981.
- [13] B. Boehm, B. Clark, E. Horowitz, C. Westland, R. Madachy, and R. Selby, "Cost Models for Future Software Life Cycle Processes: Cocomo 2.0", *Annals of Software Engineering, Software Process and Product Measurement*, Science Publishers, Amsterdam, The Netherlands, vol. 1, pp.57-94, 1995.
- [14] Ali A. Malik, B. Boehm, "Quantifying requirements elaboration to improve early software cost estimation", *International Journal of Information Sciences*, vol. 181-13, Elsevier Science Inc., New York, 2009.
- [15] L. Laird and C. Brennan, "Software Measurement and Estimation: A Practical Approach", IEEE Computer Society, John Wiley & Sons, Hoboken, 2006.
- [16] R. Lagerström, Liv von Würtemberg, H. Holm and O. Luczak, "Identifying factors affecting software development cost and productivity", *Software Quality Journal*, Springer Netherlands, pp. 1-23, 2011.
- [17] R. Lagerström, Liv von Würtemberg, H. Holm and O. Luczak, "Identifying factors affecting software development cost", *Fourth International Workshop of Software Quality and Maintainability (SQM)*, Germany, 2010.
- [18] Z. Jiang and P. Naud'e, "An examination of the factors influencing software development effort", *International Journal of Computer Information and Systems Science and Engineering*, vol. 1, no. 3, pp. 182-191, 2007.

- [19] C. Ferreira, “*Gene Expression Programming: A New Adaptive Algorithm for Solving Problems*”, Complex Systems, vol. 13 - 2, pp. 87-129, 2001.
- [20] C. Ferreira, “*Gene Expression Programming: Mathematical Modeling by an Artificial Intelligence*”, 2nd Edition, Springer-Verlag, Germany, 2006.
- [21] J. R. Koza, “*Genetic Programming: On the Programming of Computers by Means of Natural Selection*”, Cambridge, MA: MIT Press, 1992.
- [22] J. R. Koza, “*Genetic programming II: automatic discovery of reusable programs*”, Cambridge, MA: MIT Press, 1994.
- [23] K. Deb, A. Pratap, S. Agarwal, T. Meyarivan, “*A fast and elitist multi-objective genetic algorithm: NSGA-II*”, IEEE Transactions on Evolutionary Computation, vol. 6-2, pp. 182–197, 2002.
- [24] A. J. Albrecht, and J. E. Gaffney, “*Software function, source lines of codes, and development effort prediction: a software science validation*”, IEEE Trans. on Software Eng. SE-9, pp.639-648, 1983.
- [25] Chris F. Kemerer, “*Reliability of Function Points Measurement. A Field Experiment*”, Communications of the ACM, vol.36 -2, pp. 85-97, February 1993.
- [26] J. McCall, P. Richards and G. Walters, “*Factors in Software Quality*”, Technical Report RADC-TR-77-369, US Department of Commerce, 1977.
- [27] Roger S. Pressman, “*Software engineering: a practitioner’s approach*”, 5th ed., McGraw-Hill series in computer science, 2001.
- [28] A. Abraham, N. Nedjah, L. Mourelle, “*Evolutionary Computation: From Genetic Algorithms to Genetic Programming*”, Studies in Computational Intelligence, Springer Berlin, vol. 13, pp. 1-20, 2006.
- [29] T. Back and H. Schwefel, “*An overview of evolutionary algorithms for parameter optimization*”, Journal of Evolutionary Computation, MIT Press Cambridge, vol. 1-1 MA, USA, 1993.
- [30] D. B. Fogel, “*Evolutionary Computation. Toward a New Philosophy of Machine Intelligence*”, IEEE Press, Piscataway, NJ, 1995.
- [31] Online article on Gene Expression Programming, (URL: <http://www.gene-expression-programming.com/>), Feb 27, 2012.
- [32] E. Zitzler, M. Laumanns and S. Bleuler, “*A Tutorial on Evolutionary Multiobjective Optimization*”, Metaheuristics for Multiobjective Optimisation, Lecture Notes in Economics and Mathematical Systems, Springer Berlin, vol. 535, pp. 3-37 , 2004.
- [33] Online data repository, (URL: <http://www.isbsg.org/>) Feb 28, 2012.
- [33] M. Shin and A. L. Goel, “*Empirical data modeling in software engineering using radial basis functions*”, IEEE Trans. on Software Engineering, pp. 567-576, 2000.

Using metadata for automated testing of complex object structure

Jaroslav Zacek

Faculty of Science, 30. dubna 22
University of Ostrava
Ostrava 702 00, Czech Republic
jaroslav.zacek@osu.cz

Frantisek Hunka

Faculty of Science, 30. dubna 22
University of Ostrava
Ostrava 702 00, Czech Republic
frantisek.hunka@osu.cz

Abstract—Ontologies have more than one level of the abstraction view. This ensures to create a precise model of the subject under study. Different level architecture requires using a specific model techniques such as power-type objects to represents object that exists on more than one level. Complex object structure makes hard to implement unit testing. This paper proposes a new approach to implement unit testing by using object metamodel. That makes unit testing fully automated and increase efficiency. The paper also proposes a required extension to JUnit framework and verifies the approach on the real model.

Keywords—Automated testing; reflection; metamodel; power-type; REA ontology

I. INTRODUCTION

Latest researches in modelling are focused on metamodel and working with meta-models. Resource-Event-Agent business ontology is defined as a multi-level ontology to define economic business processes. Main advantage of the REA model is ability to make real-time overlook of the resources in model. That means the management can have relevant information about supply items, cash flow, and orders almost immediately and do not have to wait until financial statements. The economic model contrary to model in any other domain has specific needs. We can define a simple business process for example – car production. Customer buys a new car. He signs the contract and defines parameters. Every car has specific serial number. At the time of signing the contract there is no serial number because the car is not in the product line yet. The model (information system) must carry the information about car type and must be able to define the specific attributes later. Car has many accessories and accessories can be bound to the car serial number (for example the GPS system). Every accessory is bound to contract and specific car; therefore it must be bound to specific car in model with no serial number. When the car is finished the serial number is created and all components in model must be informed about changing the value. These complex model structures can be created with power-types. However automated testing of these special object types is not trivial, consume developer time, and increase complexity of the whole implementation.

II. REA ONTOLOGY

We can find a several concepts in almost every information system. If concepts are determined and defined

correctly a whole design of the information system is much easier and do not violate domain rules. An adaptation of the new requirements is easier as well. These concepts are known as REA (Resources, Events, Agents). Fundamentals of the REA concepts for economic software are economic resource, economic event, economic agent, commitment, and contract. REA also specifies a domain rules to ensure consistency of the whole information system from the business perspective. There are, of course, other approaches to define the modeling entities (archetypes, pleomorphs). These concepts are focused on certain subdomain in specific business domain therefore REA ontology can generally describe all these concepts. REA was originally proposed as a generalized accounting model and published by W.E. McCarthy in 1982. Since then, McCarthy and G. Geers have extended original REA model to a framework for enterprise information architecture and ontology for business systems in 2002. REA became the foundation for several standards such as ebXML, and Open-edi. Fundamentals of the REA concepts are presented in fig. 1.

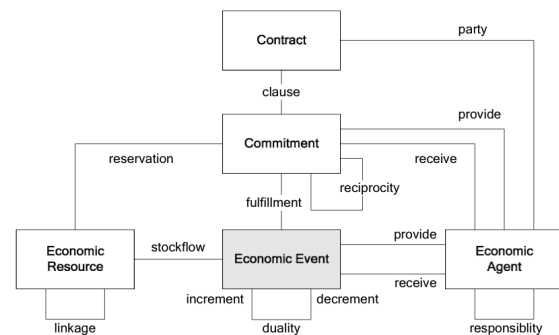


Figure 1: REA concept fundamentals

Economic Resource is a thing that is valuable and unique to other and has utility for economic agents. Usually users want to plan monitor and control that resource which must be also a part of economic information system. Examples of the economic resources are money, material, labor, tools, services and products.

Economy Agent can be an individual or organization capable of having control over economic resources and capable of interaction between other economic agents. Economic agents are usually customers, vendor or other companies. The company itself is a specific economic agent,

because the REA model is created from the company perspective.

Economic event is an act that represents an increment or a decrement specific value of the economic resource. The economic event can occur instantaneously (purchase of merchandise) or over time (rental).

Commitment expresses the obligation created in a long-time economic event. For example, commitment represents a one item on the order.

Contract is a set of increments and decrements commitments. Contract also specifies what happens if the commitments are not fulfilled. Order document is a typical example of contract concept. Order consists of items to order and can specify the penalties if the items have not been received. Additional penalties are specified as another commitments.

REA has two abstraction conceptual levels – operational level and policy level. Operational level defines concepts that describes past events and policy level defines planning by future events.

III. TYPIFICATION IN REA ONTOLOGY

Basic typification is shown in fig. 2. *StudentType* instance represents a different sort of students. *StudentType* has three attributes – *yearOfStudy*, *groupName* and *course*. Student instance is always directly bound to specific *StudentType* and specifies the instance by three attributes – *studentName*, *studentAddress* and *grades*. Student instance can access to *StudentType* attributes. Relationship between *StudentType* and Student is 1:M. That means if one instance of Student class changes inner attribute of the *StudentType*, all instance of Student class bound to that *StudentType* instance reflects that change automatically.

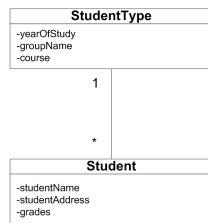


Figure 2: Typification

Another important fact is that *StudentType* is instantiated first and represents a parent of the complex Student model structure. A child objects Students are instantiated afterwards. This structure is convenient if the user must change the values of the parent object periodically and needs a specific grouping, for example year of study in object *StudentType*.

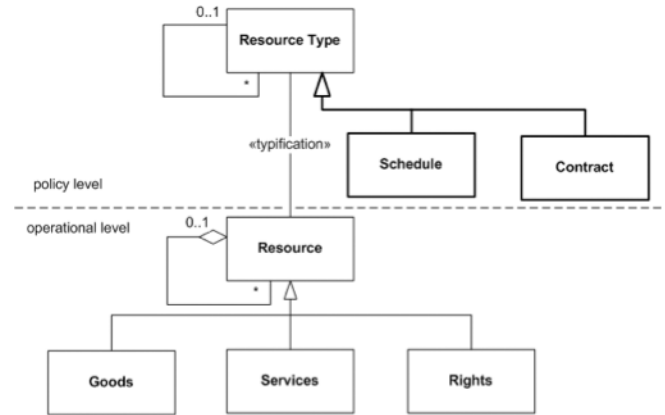


Figure 3: Resource typification in REA ontology

Fig. 3 shows resource model in REA ontology with two abstraction levels. Policy level expresses the future event by *ResourceType*. Object Resource on operational level represents existing resources as Goods, Services, and Rights. Modelling problem between Resource and *ResourceType* must be solved by typification. When user creates an order a type of resources is created. At that time there is no Resource specification such as serial number, colour etc. Specific resource is specified only as a type of Resources not the actual instance of object Resource. *ResourceType* is specific instance that will be specified later during order process by objects Schedule and Contract. Another words *ResourceType* can be and will be changed anytime and every resource connected to *ResourceType* must be aware of that change.

From a developer point of view there are two possibilities to implement model with changing parent objects:

- Object reference as an attribute in parent
- Inner classes

ResourceType class has specific attribute – Resource. This attribute expresses an array of resources instantiated and related to *ResourceType*. All instances of *ResourceType* are stored in hash map and all Resource instances can access *ResourceTypes* attributes by searching with provided ID during Resource instantiation. This construction is not adequate because of complex data structure and complicated implementation. A hash map requirement can be resolved by using reflection in every child class to get reference to parent instance.

Inner classes are much more suitable to model typification. This construction is also called a power-type [6]. Power-type is construction between two or more meta-layers to model complex objects such as policy and operational level in REA ontology. Following testing process can be applied to both constructions however inner classes are more widespread and therefore the paper aims this construction.

IV. TESTING POWER-TYPES

Implementation of the power-type is class with inner classes. From a testing point of view a power-type should be tested on the lowest level. Software testing at the lowest level (programming language) is referred to as unit testing. At this level, individual inner classes and specified methods are tested. More emphasis is placed on verification and black box testing at this level. The main concern is that algorithms works correctly and specified values are verified. Basically a unit test is a test designed to evaluate a piece of code. A suite of test can be used to evaluate an entire solution. Unit testing should be divided into four areas:

- Class scope testing
- Inheritance-level class scope testing
- Implementation-level class scope testing
- Method scope testing

Class scope testing defines the interfaces for power-types and then establishes and maintains their effectiveness thought the development. Power-type is usually not interface itself, but should contain some complex methods using interfaces from inner classes.

Inheritance-level testing aims to check all methods inherited from super classes. Power-type is a specific construction of inner classes therefore there is no inherited method from super class. In fact a power-type construction is a different approach to ensure inheritance.

Implementation-level class scope testing is exactly the opposite to inheritance-level testing. The main concern is to test all implemented methods in actual classes and none of the inherited methods. Overridden methods are regarded as implemented methods.

Method scope testing is an alternative to class-level testing and is focused to specific methods or to the most important method of the specific class. This approach is suitable for specific design classes with few core methods that do a lot of work in static context. Method-level testing is usually used in evaluating robustness.

Automated tools are used widely in the test driven development approach to minimize human work time (developer/tester work), speed up test process and reduce human error. One of the most used frameworks is JUnit. JUnit is a testing framework that was originally developed by Kent Beck [9] and Erich Gamma for testing at class-level scope. Since then, a lot of implementation specifications and testing patterns has been discovered [1]. JUnit has special package to compare object values – Assert (org.junit.Assert). This package provides libraries to check values defined by object types (Integer, Double, String, Long...).

However this package is prepared only to objet values and primitive data types and cannot state of equation on complex structures such as Power-type. An example of typification is presented in fig. 4. Subject under study is university consisting of several faculties. Every faculty has at least one department.

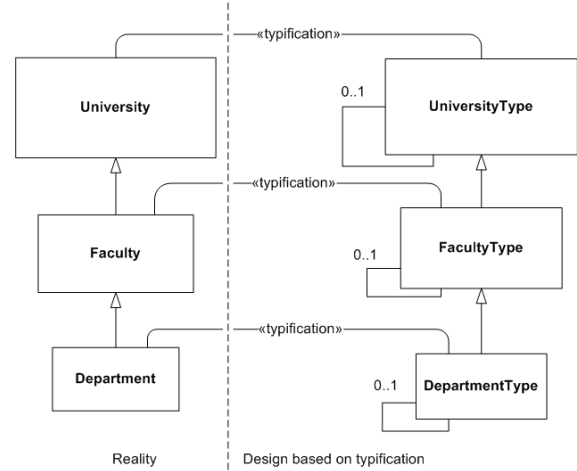


Figure 4: Typification example

At this example the typification is much more appropriate than inheritance. Top level of the model defines a university.

University has specific attributes, such as name, address and town. These attributes have been defined to faculty and department as well. Main difference between generalization and typification is in inner attributes. In generalization the department is defined by inner attributes, attributes inherited from the Faculty and attributes inherited from the University. All attributes are on the Department instance level. That means if university changes address, all instances of classes Faculty and Department must be informed about that change and therefore one change of the attribute generates many new method calls.

A model example of University-Faculty-Department shows fig. 5. This class diagram is based in fig. 4 by applying typification approach. *University* class represents top of the hierarchy with attributes universityName, universityType, rectorName, chancellor, address, and town.

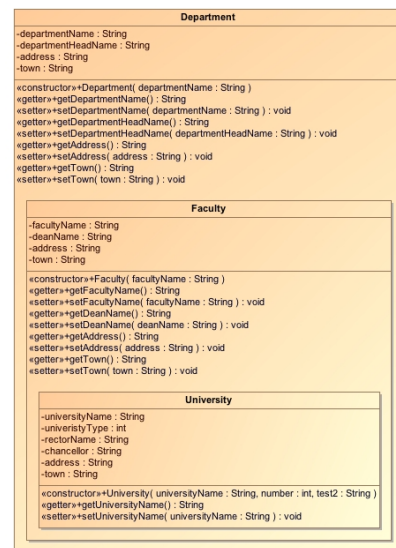


Figure 5: Class diagram of University-Faculty-Department model example

V. DESIGNING A POWER-TYPE TESTING LIBRARY

Power-type construction could be a significant part of the REA model implementations. A lot of models have been analyzed and determined basic requirements for testing library:

- Library must be able to test across the level of typification
- Test power-types automated
- Reduce the time to create test case
- No additional implementation to tested class
- Easily implemented
- No additional dependences to other libraries

The library must be able to reveal a number of inner classes in implemented power-type. Theoretically the number of inner classes is infinite as seen in fig. 6. Practically the number depends on programming language implementation (for Java programming language is maximum Integer.MAX_VALUE). For example model from fig. 5 has three meta-levels: University, Faculty and Department and library must equals all attributes in the right meta-level. Another example has only two levels. The library must determine all outer classes to get their references automatically.

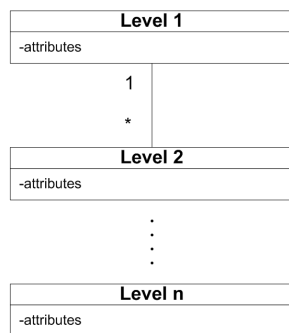


Figure 6: Inner class level

Library must be able to work with automated testing frameworks such as JUnit. Automated testing framework works with assert library and expects Boolean value. A new method, which decides equality of a two power-types, will return always a Boolean value. Method will be marked as static as well according to design pattern Library. These conditions support integration to existing testing frameworks.

Significant lines of code must be implemented to test all inner attributes of the main class from fig. 5. Additional lines must be created in inner classes to access all attributes. Access methods are often implemented just for testing purpose and makes whole code confusing. 44 lines of code must be written to state that two instances of *Department* class are equal.

All testing methods should be separated from tested class [1]. The new library must be able to get all needed parameters from metamodel of the instantiated class (power-type).

The key to the easy implementation and usability is well-arranged code [10] and simplicity of design. New method must be created according to design pattern Library and

provide documented interface to seamless integration to current project.

New library must also use only API of the current programming language (in this case Java API).

VI. POWER-TYPE METAMODEL

Every instance of the power-types class has the own metamodel carries important information about inner structure, number of attributes and method. This metamodel must be used to discover inner attributes and get reference to the concrete value. We used a reflection to allow work with metamodel in our testing library.

A. Reflection and metamodel

Reflection as a term in information science means ability to read and change program structure and behavior during the program running. Considering object-oriented programming approach, reflection means ability to read and change object attributes, read and execute the object methods, passing calling results and instantiate new objects. Generally the reflection is able to read object metamodel during program running without changing any object attributes. Reflection is widely used with Smalltalk programming language and scripting languages. Reflection can be used as a universal tool to make object persistent [2] or to generate project documentation.

Reflection enables creating a new object instance entered by name during program running. Following source codes are in Java programming language, but same function can be done with .NET platform and languages defined under Common Language Specification. Basically there are two requirements to programming languages:

1. Ability to read object metadata and work with them as a metamodel (object self-identification).
2. Some tool to enable object metamodel extension.

Before the source code of metadata model instantiation the metamodel must be discovered.

By using reflection technique most of the requirements defined in section 5 are met. Reflection can obtain a reference to instance of inner and outer class. Reflection can obtain all declared fields in instance. If instance consists of more then one object (typically power-types) last declared field is a reference to upper object and in Java language is marked as „this\$0“. Number after dollar symbol expresses the level nested class. Therefore we must use a technique to discover number of nested classes in power-type as show in fig. 7.

```
private static int getLevel(Object obj) {
    Field[] fields = obj.getClass().getDeclaredFields();
    String[] stringArray = (fields[fields.length - 1]
        .toString()).split("this\\$");
    return new Integer(stringArray[1]);
}
```

Figure 7: Method to determine number of nested classes

Method has only one parameter – obj. First line gets a reference to declared fields of the prototype instance. Declared fields are part of the power-type metamodel.

Second line parses the String value obtained from the array of declared fields.

The whole metamodel of instantiated class can be discovered by method *getClass()*. This universal method is a part of Object metamodel defined by Java programming language and therefore an additional library import is not required. Method retrieves a runtime class of an object. Retrieved object is locked by static synchronized methods of represented class. To reveal a number of levels in power-type instance we do not have to browse all inner classes. Method *getDeclaredFields()* is a part of Object metamodel as well. This method retrieves an array of objects reflecting all the fields declared by the class or interface represented in Class object returns by *getClass()* method. This includes public, protected, and private fields. The method does not include inherited fields or fields from inner classes. Retrieved elements are not sorted and are not in any particular order. If the method returns array with length 0 there is no declared field in specific instance of class or interface. The retrieved array contains specific classes – Field. A Field class provides information about, and dynamic access to, a single field of a class or interface using their metamodel. The reflected field may be a class field or an instance field. Every returned field has method *toString()* returning String object to describe the field. Last string is a reference to outer class. The format is the access modifiers for the field, if any, followed by the field type, followed by a space, followed by the fully-qualified name of the class declaring the field, followed by a period, followed by the name of the field. The modifiers are placed in canonical order as specified by Java language specification. The power-type in fig. 5 includes this string: “final cz.osu.example .University\$Faculty cz.osu.example .University\$Faculty\$Department.this\$1”. In this context the final means that the reference has been created in instantiation process and cannot be change except calling class destructor. Modifier is followed by definition of full outer class name including packages. Inner classes are divided by dollar symbol.



Figure 8: Reflection field references

In this example are inner classes represented as *University\$Faculty*. Next string is definition of current power-type class composed from class University, Faculty, and Department. Last entry of the whole string value is the reference to outer class in notation “.this\$1”. Diagram in fig. 8 shows comprehensive structure of the references for example from fig. 5.

The basic idea of the testing library is to obtain metamodel at the top level of the power-type and iterate through all values. If the values are equal (including complex object data types) the algorithm finds a reference to the next inner class and repeats the comparative process. When the comparative process ends on all level of the two compared power-types the process ends and returns a result. The

algorithm can be speed up by comparing the number of inner classes. The whole process shows flowchart in fig. 9.

One of the fundamental features of the object-oriented programming is encapsulation. Metamodel of the power-type respects this attribute with key word private. Reflection uses special method to retrieve value of the current field and this method returns specific data type wrapped to Object data type. From an algorithm point of view the specific data type is not relevant and we do not have to check or cast retrieved value. All objects in programming language (specifically Java) have method equals implemented by default Object specification and we use this method to compare values. However if the value is marked as private the method of the reflection package throws *IllegalAccessException* as a response to obtain the value. Therefore we must set accessibility mark to every field that will be obtained and compared. This approach has one big advantage. The comparative process runs over all fields of the power-type metamodel on all levels not interfered by encapsulation and therefore we can assert that two power-types are equal or not.

B. Implementation and consequences

We choose a Java programming language because we have a lot of projects implemented on Java technologies and significant count of power-types for automated testing.

First step of the algorithm flowchart diagram instructs to obtain level of classes in power-types. Full implementation is relatively long and because of page limitation the paper shows only significant passages.

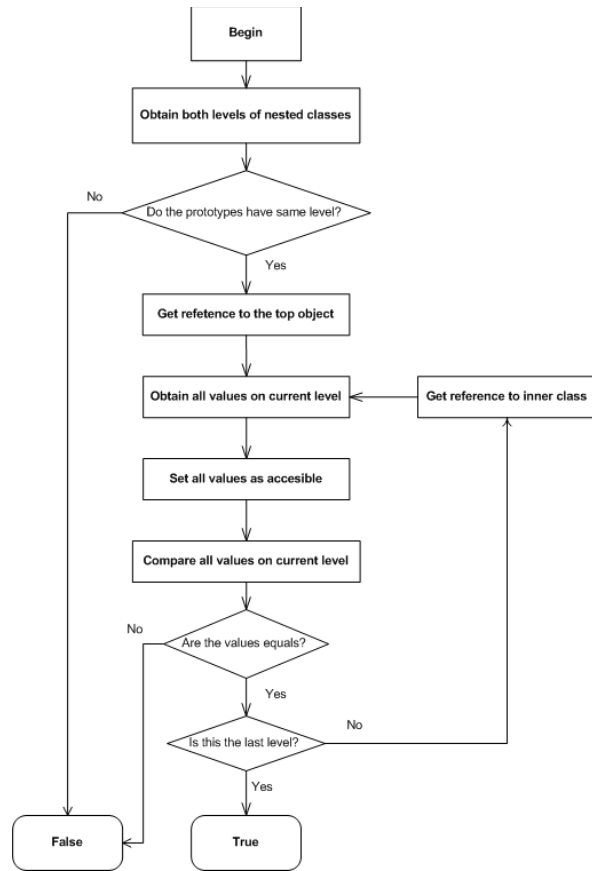


Figure 9: Algorithm flowchart

VII. RESULTS

By implementing new library for testing prototype equality a coding has been dramatically reduced. In the beginning the testing of power-type model from figure 5 required about 3 lines of code per one attribute. That means 42 lines of Java code to test equality on whole model excluding getters and setters methods. These methods are required for testing however these methods should be implemented automatically and therefore we do not count them.

```
System.out.println("Equals? - " +
    Power-typeTesting.equals(real, real2));
```

Figure 10: Power-type testing library example of use

That means that by using a *Power-typeTesting* library the code is reduced three times. Moreover there is no requisition to implement accessing methods. At the beginning of the project we tested the library to simple power-types such as Flight-RealFlight, University-Faculty-Department. Later we used complex economic power-types in REA ontology. We have tested Commitment-Economic Event power-type, Resource-Resource Type power-type, Event-Event Type power-type, and Agent-Agent Type power-type. These power-types have been identified in REA ontology before and by using new testing library the time required for testing was dramatically reduced. New code was well arranged,

automated and fully integrated into JUnit testing framework. The project PowerTest is an open-source and is available at <https://sourceforge.net/projects/powertypetest/>.

VIII. CONCLUSION

The paper is focused to problem of automated testing power-type objects. First section introduces the reader to the problem of power-types testing and summarizes the main challenges in testing complex object structures. Next two sections define key principles and terms of REA ontology and use of power-types in REA ontology designing model. These two sections determines of using power-types as necessary technique to support REA models creation. Section 4 discuss ordinary techniques to automated testing of power-type objects and defines basic approaches to test complex object models. A main requirements for creating a universal testing library has been defined. Based on the requirements a new testing library has been designed and implemented. A new testing library has been successfully tested in many REA ontology models. The results are significant reduction of time to realize unit testing and fully integration to JUnit framework. The new code is well arranged and number of necessary lines has been reduced three-times. Moreover the testing library is not bound only to REA ontology power-types and can be used on any other complex object with inner classes. The implementation is available as open-source at Sourceforge.

Acknowledgments: This paper is supported by IGA no. 6141, Faculty of Science - University of Ostrava

REFERENCES

- [1] J. Thomas, M. Young, K. Brown, A. Glover, "Java Testing Patterns", Wiley Publishing Inc., 2004, ISBN 0-471-44846-X 41-67.
- [2] I. Forman, N. Forman, "Java Reflection In Action", Manning Publications Co., 2005, ISBN 1-932394-18-4.
- [3] C. J. Chang, L. R. Ingraham, "Modeling and Designing Accounting Systems", Wiley Publishing Inc., 2007, ISBN 13 978-0-471-45087-0.
- [4] P. Hruby, "Model-Driven Design Using Business Patterns", Springer, 2006, ISBN 13 978-3-540-30154-7.
- [5] J. Arlow, "Enterprise Patterns and MDA", Addison-Wesley Professional, 2004, ISBN 9 780-321-11230-9.
- [6] C. Gonzales-Perez, "Metamodelling for Software Engineering", Wiley Publishing Inc., 2008, ISBN 13 978-0470-03036-3.
- [7] M. Soden, "Operational Semantics for MOF Metamodels", available on-line at <http://www.metamodels.de/docs.html>, Mar. 2010.
- [8] B. Eckel, "Thinking in Java (4th edition)", Prentice-Hall PTR, 2006, ISBN 13 978-0131872486.
- [9] K. Beck, "Test-Driven Development: By Example", Addison-Wesley Professional, 2002, ISBN 978-0321146533.
- [10] S. McConnell, "Code Complete: A Practical Handbook of Software Construction", 2nd edition, Microsoft Press, 2004, ISBN 978-0735619678.

Empirical Estimation of COCOMO I and COCOMO II Using a Case Study

Muhammad M. Albakri¹

M. Rizwan Jameel Qureshi²

¹⁻²Department of Information Technology, King Abdul-Aziz University, P.O. BOX 80221 Jeddah 21589, Saudi Arabia

Abstract- *There are several software estimation models such as Line of Code, Function Point and CONstructive COst MOdel (COCOMO). The original COCOMO model is one of the most widely practiced and popular among the software development community because of its flexible usage. It is a suite of models i.e., CONstructive COst MOdel I and CONstructive COst MOdel II. In this paper, we are evaluating the both models, to find out the level of efficiency they present and how they can be tailored to the needs of modern software development projects. We are applying COCOMO models on a case study of an E-Commerce application, that is built using HTML and JavaScript. We will also shed light on the different components of each model, and how their Cost Drivers effect on the accuracy of cost estimations for software development projects.*

Keywords- *COCOMO I; COCOMO II; Software Cost Estimation; Software Cost Drivers' Assessment; Trade-off Analysis; Component Composition.*

1 Introduction

The main stimulus for the COCOMO I model is to help people understand the cost consequences of the decisions they will make in developing and supporting a software product. COCOMO II not only offers a cost estimation tool, but also provides a great amount of parameters which explain what the model is estimating, and why it produces the estimates it does. COCOMO I is actually a hierarchy of three sub-models and each sub-model is progressively more detailed than the other. This paper will present our results and findings after applying two of COCOMO's sub-models. The First sub-model is 'Basic COCOMO'. It is a single-valued model and calculates the software development cost and effort of a program by measuring lines of code (LOC). Basic COCOMO itself is divided into three modes based on the nature of the software project. First is 'Organic Basic COCOMO', it is used in small-sized simple software projects developed by small teams with good application

experience. Second is 'Semidetached Basic COCOMO', it is used in medium-size software projects developed by teams with diversified levels of experience. Third is 'Embedded Basic COCOMO', that is used in massive software projects with strict resource constraints developed by multiple teams acquiring immense levels of experience, and sophistication. The second sub-model is 'Intermediate COCOMO'; it is simply 'Basic COCOMO' plus a set of subjective 'Cost Drivers'. Those drivers are used to assess product, computer, personnel, and project attributes of a software project. The evaluator uses a six-level scale to decide where each attribute fall. When an attribute is assessed, it produces what is called an Adjustment Factor. After all adjustment factors are multiplied together, they give an Effort Adjustment Factor (EAF) that is usually equal to a value between 0.9 and 1.4. The EAF is then mathematically applied on all Basic COCOMO's formulas. Third sub-model is Detailed COCOMO, as the name indicates, it produces the most accurate estimation of all three sub-models of COCOMO I. It combines Basic and Intermediate COCOMO together, boosted by an assessment of every Cost Driver's impact on each stage of 'Barry Boehm's software engineering process'. COCOMO II model on the other hand, is divided into four sub-models. Each sub-model is based on different inputs and estimates the effort of different activities of a software project. 'Application Composition' is the first sub-model. It estimates the effort of prototype systems developed using scripts, database programming, etc. And it uses application points as an input. Second sub-model is 'Early Design', it calculates initial effort based on system requirements and design options, and uses function points as an input. Third sub-model is 'Reuse', it estimates the effort of integrating reusable automatically generated components and uses generated line of code as an input. Fourth sub-model is 'Post Architectural', it estimates the development effort of system design specifications and uses lines of source code as an input.

The paper is further organized as: section 2 covers related work. Section 3 defines the research problem. Section 4 describes the brief case study design. Section 5 illustrates the evaluation. Section 6 covers the discussion.

2 Related Work

Boehm et al. [1] proposed evaluation criteria for the validity of the process models and they provided effective results. This article also explained the strengths and weaknesses of various cost estimation techniques for the period of 1965 to 2005 (40 years). COCOMO-II [2] was an excellent model up to 2005 but it did not enfold the new requirement and development styles for the reuseness or estimation of cost. COCOMO-II directed the software experts to create and designed new models such as the Chinese government version of COCOMO (COGOMO) and the Constructive Commercial-off-the-Shelf Cost Model (COCOTS) etc. Different future challenges were discussed for the invention of new model/methods and tools.

The author discussed different software cost estimation techniques and highlighted various hot areas and challenges of research in the field of software cost estimation. In [3], it is emphasized that there should be a need to research more in this field to open the new horizons for novice researchers. Nasir [4] discussed the strengths and weaknesses of various software estimation techniques to provide the basis for the exactness of software cost estimation. Basic Project Estimation Process also presented in a wonderful style. This paper clearly elaborated the different types of models those were derived from COCOMO (I&II).

Reusability of components in Component Based Development (CBD) is illustrated in [5]. The research in [5] also discussed and compared different architectures of CBD. It may be mentioned that a detail explanation of advantages and disadvantages of CBD elaborated very nicely. A comparison, of component based development (CBD) with other traditional software development practices, is also provided.

Succi and Baruchelli [6] highlighted the importance of standardization of components for the software reusability. The discussion of this research paper was to find how total development cost of a software system affected on the basis of component-based software engineering. The main two factors those were affecting the standardization cost of a component have been explained. According to them, the cost of the standardization of component(s) must be included during the cost-benefit analysis of a software system.

Gill [7] highlighted the pertinent issues of software reusability for component based development on the basis of CBSE, considered the important issues of software reusability and high level reusability guidelines. He mentioned that how much reusability resulted to improve product reliability and to reduce overall software development cost.

The problem of crosscutting which is produced during component development is elaborated in [8]. They solved this problem by the extension with Aspect oriented methodology. It was mentioned by an example that how new business rules resulted in the more adaptable and reusable components. Aspect Component Based Software Engineering has been developed with success in the CORBA Component Model domain [9].

Dolado [10] wrote a report for the validation of the component-based method (CBM) for software size estimation by the analysis of 46 projects. Then the complete process of this analysis and different techniques of analysis was mentioned. Relationship of LOC (Line of Code) and NOC (No. of Component) was carried out with suitable examples. Comparison of CBM and a Global Method (Mark II) was also provided [10].

3 Research Problem

A number of discussions are reported in the literature for the effectiveness of COCOMO models. This paper is written to find out the accuracy of cost estimation of both models when applied on a specific project. Further, what is the impact of cost drivers during the system development life cycle phases.

We want to validate the accuracy of the cost estimations of COCOMO models for projects that are built using HTML and JavaScript. Hence, we will not only find out how accurate and reliable they are, but also whether they are suitable for estimating HTML and JavaScript Code.

4 Case Study Design

The case study is for a project completed by author Mr. Albakri in a course called 'Human Computer Interaction'. The objective of the project is to follow the principles of HCI in creating an E-Commerce web application of an online bookstore. The application consists of fourteen webpages written in HTML and JavaScript. All fourteen pages were fully designed to have different content and perform different web tasks. Then, they were coded and connected together according to their design. The pages are a demo experience of how a real user would buy a book online. Details of the pages are mentioned in the next section.

5 Evaluation

The sub section 5.1 covers the COCOMO I whereas COCOMO II is covered in the sub section 5.2 subsequently.

5.1 Applying COCOMO I

Sub-model Used: Basic COCOMO I

Mode Used: Organic

Formulas Used:

$$Effort(\text{Man Month}) = 3.2 \times (KLOC)^{1.05} \quad (1)$$

$$Time = 2.5 \times (Effort)^{0.38} \quad (2)$$

Calculating Total LOC:

Table 1: HTML Pages, 14 pages in total:

Webpage Name	Number of Lines of Code
aboutsUs	101 LOC
bookDetails	119 LOC
categories	376 LOC
congrats	91 LOC
contactUs	144 LOC
feedBack	267 LOC
index	276 LOC
myAccount	114 LOC
payment	245 LOC
searchResults	327 LOC
shoppingCart	157 LOC
signIn	118 LOC
signUp	190 LOC
verification	146 LOC
Total = 2918 LOC, 2.918 KLOC	

Estimating Effort:

$$Effort = 3.2 \times (2.918)^{1.05}$$

$$Effort = 9.851 \text{ MM}$$

Estimating Time:

$$Time = 2.5 \times (9.851)^{0.38}$$

$$Time = 5.963 \text{ Month}$$

Sub-model Used: Intermediate COCOMO I

Mode Used: Organic

Formulas Used:

$$Effort(\text{Man Month}) = EAF \times 3.2 \times (KLOC)^{1.05} \quad (3)$$

$$Time = 2.5 \times (Effort)^{0.38} \quad (4)$$

Cost Drivers:

- Product Attributes:
 1. RELY- Required Software Reliability.
 2. DATA – Database Size.
 3. CPLX – Product Complexity.
- Computer Attributes:
 4. TIME – Execution Time.
 5. STOR – Main Storage.
 6. VIRT – Virtual Machine Volatility.
 7. TURN – Computer Turnaround Time.
- Personal Attributes:
 8. ACAP – Analyst Capability.
 9. AEXP – Applications Experience.

10. PCAP – Programmers Capability.
11. VEXP – Virtual Machine Experience.
12. LEXP – Programming Language Experience.

- Project Attributes:
 13. MODP – Use of Modern Programming Practices.
 14. TOOL – Use of Software Tool.
 15. SCED – Required Development Schedule.

Table 2: Estimating Cost Drivers Values:

	Very Low	Low	Normal	High	Very High
RELY				1.15	
DATA					1.16
CPLX					1.30
TIME		0.85			
STOR				1.21	
VIRT				1.30	
TURN				1.15	
ACAP				0.86	
AEXP		0.80			
DCAP			1.0		
VEXP				0.90	
LEXP				0.95	
MODP			1.0		
TOOL					0.83
SCED		0.85			

Calculating Effort Adjustment Factor(EAF):

Here all assessment values are multiplied together to determine the EAF:

$$EAF = 1.15 \times 1.16 \times 1.30 \times 0.85 \times 1.21 \times 1.30 \times 1.15 \times 0.86 \times 0.80 \times 1.0 \times 0.90 \times 0.95 \times 1.0 \times 0.83 \times 0.85 \quad (5)$$

$$EAF = 1.1$$

The equation further substitutes as follows.

$$Effort(\text{Man Month}) = 1.1 \times 3.2 \times (2.918)^{1.05}$$

$$Effort = 10.836 \text{ MM}$$

$$Time = 2.5 \times (10.836)^{0.38}$$

$$Time = 6.182 \text{ Month}$$

Sub-model: Organic Detailed COCOMO I

This sub-model was not used this model based upon two reasons. First, is that this E-Commerce application does not require to go through the detailed project phases of 'Barry's software engineering process. Second, it is a

small scale project. Though, the findings will be stated in section 6.

5.2 Applying COCOMO II

Sub-model Used: Application Composition

Formulas Used:

$$Object\ Point(OP) = Estimated\ Count(EC) \times Complexity\ Factors(CF) \quad (6)$$

$$New\ Object\ Point(NOP) = OP \times \frac{(100 - \%reuse)}{100} \quad (7)$$

$$Effort(Man\ Month) = \frac{NOP}{Productivity} \quad (8)$$

$$Cost/NOP = \frac{Labor\ Rate}{Productivity} \quad (9)$$

$$Total\ Project\ Cost = \frac{Cost}{NOP} \times NOP\ of\ current\ project \quad (10)$$

Table 3: Project Parameters:

	EC	CF		
		Simple	Average	Complex
Screens	14	1	2	3
Reports	6	2	5	8
Components	10	1	1	10
Environment Maturity	25			

After Substitution:

$$Object\ Point(OP) = (14 \times 2) + (6 \times 5) + (10 \times 1) \quad (11)$$

$$OP = 68\ OP$$

This e-commerce application reuses 90% of previous common e-commerce applications.

$$New\ Object\ Point(NOP) = 68 \times \frac{(100 - 90)}{100}$$

$$NOP = 6.8 \approx 7\ NOP$$

$$Effort(Man\ Month) = \frac{6.8}{25}$$

$$Effort = 0.272 \approx 1\ Man\ Month$$

Assumed, the average labor rate is 1500 USD.

$$Cost/NOP = \frac{1500}{25}$$

$$Cost/NOP = \$60$$

$$Total\ Project\ Cost = 60 \times 6.8$$

$$Total\ Project\ Cost = \$408$$

Sub-model Used: Early Design

$$Formula\ Used: Effort = A \times Size^B \times M \quad (12)$$

Where:

A is 2.94, a coefficient proposed by Boehm,

Size, is in KLOC,

B, reflects the increased effort required as the size of the project increases, ranges from 1.1 to 1.24.

M, is a multiplier which is based on a simplified set of seven project characteristics that influence the estimation.

Project Characteristics:

1. RCPX – Product Reliability and Complexity
2. RUSE – Reuse Required
3. PDIF – Platform Difficulty
4. PERS – Personnel Capability
5. PREX – Personnel Experience
6. SCED – Schedule
7. FCIL – Support Facilities

Table 4: Estimated Project Characteristics:

	Very Low	Low	Normal	High	Very High
RCPX				1.5	
RUSE					1.40
PDIF				1.20	
PERS			1		
PREX			1		
SCED		0.85			
FCIL			1		

Calculating multiplier M:

$$M = 1.5 \times 1.40 \times 1.20 \times 1 \times 1 \times 0.85 \times 1$$

$$M = 1.6$$

B value is medium (equal to 1.15), because the size of this e-commerce application is predicted to require medium expansion effort.

$$Effort(Man\ Month) = 2.94 \times 2.918^{1.15} \times 1.6$$

$$Effort = 16.118\ Man\ Month$$

Sub-model Used: Reuse

$$\text{Formula: } Effort (Man Month)_{Auto} = \frac{ASLOC \times \frac{AT}{100}}{ATPROD} \tag{13}$$

This formula estimates generated code. Where:

- 'Auto', indicates that 'Effort' is of generated code,
- ASLOC, is the number of adaptive LOC of reusable components,
- AT, is the percentage of adapted generated code,
- ATPROD, productivity of engineers integrating the code, usually approximates to 2400 LOC/Month.

Here, it is assumed that the HTML and JavaScript code of reusable components is generated using design models inserted into a code generator.

$$Effort (Man Month)_{Auto} = \frac{291.8 \times \frac{0.1}{100}}{2400}$$

$$Effort_{Auto} = 0.000121 \text{ Man Month}$$

Sub-model Used: Post-Architectural

$$\text{Formula: } Effort = A \times Size^B \times M \tag{14}$$

As the name indicates, this model is used when more project parameters become identified.

Detailed Project Cost Drivers:

1. RELY – Required Reliability.
2. CPLX – Complexity of Modules.
3. DOCU – Extent of Documentation.
4. DATA – Database Size.
5. RUSE – Required percentage of Reusable Components.
6. TIME – Execution Time Constraint.
7. PVOL – Platform Volatility.
8. STOR – Memory Constraints.
9. ACAP – Analysts Capability.
10. PCON – Personal Continuity.
11. PCAP – Programmer Capability.
12. PEXP – Programmer Experience.
13. AEXP - Analyst Experience.
14. LTEX – Language and Tool Experience.
15. TOOL – Use of Software Tools.
16. SCED – Development Schedule Compression.
17. SITE – Extent of Multisite Working and Quality of Inter-Site Communication.

The last cost driver 'SITE' was excluded because the work site is not relevant to the nature of this application.

Table 5: Estimated Cost Drivers:

	Very Low	Low	Normal	High	Very High
RELY				1.15	
CPLX					1.30
DOCU				1.1	
DATA					1.16
RUSE					1.40
TIME		0.85			
PVOL		0.70			
STOR				1.21	
ACAP				0.86	
PCON				0.90	
PCAP			1		
PEXP			1		
AEXP		0.80			
LTEX			1		
TOOL					0.83
SCED	0.85				

$$M = 1.15 \times 1.30 \times 1.1 \times 1.16 \times 1.40 \times 0.85 \times 0.70 \times 1.21 \times 0.86 \times 0.90 \times 1 \times 1 \times 0.80 \times 1 \times 0.83 \times 0.85$$

$$M = 0.84$$

$$Effort (Man Month) = 2.94 \times 2.918^{1.15} \times 0.84$$

$$Effort = 8.462 \text{ Man Month}$$

6 Discussion

As mentioned above in Table 2, the estimated fifteen cost drivers provide more information about different areas of the application that were not obtainable at the beginning of the project to enhance cost and effort estimations. In the basic stage, the effort and time to develop the application was 9.851 MM and 5.963 Months respectively, but when intermediate stage was reached, 0.985 more effort and 0.219 more time was needed to complete the project.

At the beginning of the project, 'Organic Basic COCOMO' envisions the system as a single unit, whereas 'Organic Intermediate COCOMO' divides the system into subsystems or components. The intermediate cost drivers allow estimating particular components not the entire system, therefore enabling the development team to choose the best course of action regarding project's plan.

In 'Detailed COCOMO' the estimator's understanding does not only cover different project parameters, it also considers the project as a sequence of phases and each phase is estimated in a different way. That is the most major difference between it and previous sub-models. The 'Detailed COCOMO' assigns different cost drivers for each

of phase of the project. These phase-dependant cost drivers are the reason behind producing much more accurate estimations. For example if we consider the 'ACAP' cost driver, it is assigned a value of 1.00 for Coding and Unit Testing phase, which has no influence on the multiplication operation, but a value of 1.40 for Requirements phase. This intelligent manipulation of cost drivers can save up analysts' energy and time for phases that need them. They do not have an impact on coding and testing phases, because they are not involved.

COCOMO I only uses number of thousands lines of code (KLOC) as an input, and so it is best used in projects built using structured programming languages.

'Application Composition' is intended for prototype projects, where the project is built by composing components called 'Object Points'. It not only considers cost drivers but also project environment's characteristics like developer's experience and capability, CASE tool's maturity and capability, number of screens, and number of system generated reports. So, each component (object) is customized and then attached to whole body of the project in a different way and with a different level of challenge. As mentioned in previous section, 90% of the project is reused components from existing e-commerce applications. The relation between '%reuse' and 'Effort' is disproportional; the more components are reused (90% out of 100%), the less effort (1 Man Month) is required.

'Early Design' model is similar to 'Organic Intermediate COCOMO'. In each model more information is uncovered as initial stages of the project are concluded and design stages are initiated. However, only a rough system design is required to make early estimations. A very important element in the formula is 'B' as mentioned in section 5.2.2, which has a great influence on the effort estimation. While comparing the value of effort estimated by 'Application Composition' and the value of effort estimated by 'Early Design', the impact of 'B' is definitely obvious. More 15.118 man effort is needed to meet the increasing size of the project.

When 'Reuse' model is used HTML code can be generated using various code generators. It may appear that 'Early Design' model and 'Reuse' model are similar, because they both estimate reusable components, but that is not the case. In fact, the estimator using 'Early Design' needs neither to understand the reusable components nor to modify it. The estimator simply just uses them. Each time the generated code is studied and then refined, the cost of the component will decrease.

'Post-Architectural' model is used once an initial architectural design of the system is available. The model uses the same formula as 'Reuse' model uses. However, the estimation is the most accurate and realistic among others, because ten more cost drivers are uncovered and used in the formula. In section 5.2.2, 16.118 man effort is needed to develop the e-commerce application. This estimation was found to be unrealistic, because conventionally such a small scale application (2918 LOC) does not need that amount of man effort to be completed. Thanks to the detailed cost drivers, effort is reduced to 8.462 Man Month.

Acknowledgements

The author Muhammad Albakri would like to thank Dr. M. Rizwan Jameel Qureshi for his continuous and valuable support. It would never been possible to complete the work without his guidance throughout the making of this paper.

7 References

- [1] Boehm, B. W. and R. Valerdi. Achievements and Challenges in Cocomo-Based Software Resource Estimation published by IEEE Computer Society. 74-83 (2008).
- [2] Boehm, B. W. An Overview of the COCOMO 2.0 Software Cost Model (1999).
- [3] Zaid, A., M. H. Selamat, A. A. A. Ghani, R. Atan and K. T. Wei. Issues in Software Cost Estimation, IJCSNS Int J of Computer Science and Network Security, 8(11): 350-356 (2008).
- [4] Nasir, M. A Survey of Software Estimation Techniques and Project Planning Practices, Proceedings of the Seventh ACIS Int. Conf. on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing (SNPD'06), (2006).
- [5] Qureshi, M. R. J. and S. A. Hussain. A Reusable Software Component-Based Development Process Model Int. J of Advances in Engineering Software, 39(2): 88-94 (2008).
- [6] Succi, G. and F. Baruchelli. The Cost of Standardizing Components for Software Reuse, Standard View 5(2) (1997).
- [7] Gill, N. S. Reusability Issues in Component-Based Development, ACM SIGSOFT Software Engineering Notes, 28(4): 4 – 4 (2003).
- [8] Clemente, P. J. and J. Hernández. Aspect Component Based Software Engineering, University Extremadura. 1-4 Spain (2001).
- [9] Frakes, W. B. and K. Kang. Software Reuse Research: Status and Future, IEEE Transactions on Software Engineering, 31(7): 529-536 (2005).
- [10] Dolado, J. J. A Validation of the Component-Based Method for Software Size Estimation, IEEE Transactions on Software Engineering, 26(10): 1006-1021 (2000).

Comparative Analysis of Software Reliability Estimation Models –State and Path Based

Arashdeep Kaur(Student)¹ and Monika(Assistant Professor)²

¹U.I.E.T, Panjab University, Chandigarh, U.T, India

²U.I.E.T, Panjab University, Chandigarh, U.T, India

Abstract - Software reliability is an important factor that contribute to the quality of software. The objective of this paper is to provide an overview of the research in the area of architecture-based software reliability models considering the system architecture approach, uncertainty factors influencing the model. The aim of this paper is to establish a relationship between various models by identifying the limitations and enhancements of previous models over the new models.

Keywords: component-based software, reliability analysis, markov chains, component dependency graphs.

1 Introduction

Software reliability is often defined as “the probability of failure-free operation of a computer program for a specified time in a specified environment [23]. It refers to the quality of the software. Early quality prediction at the architecture design stage is highly desired by software managers and developers, as it provides a means for making design decisions and thereby facilitates effective development processes [1]. Software architecture analysis aims at investigating how an architecture meets its quality requirements [2] based on the structure and the correlation among the components of the software.

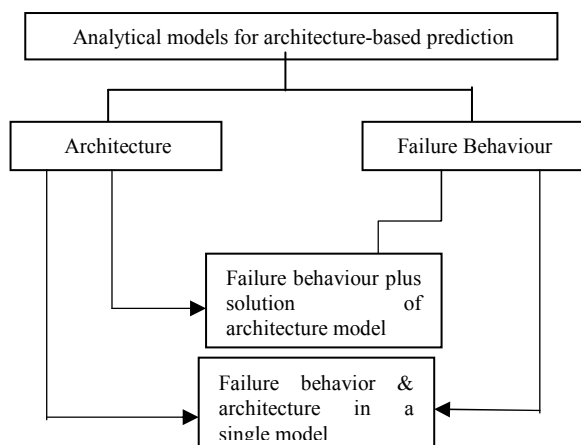


Figure 1. Classification of architecture based models.

In Figure 1 evaluation of analytical models is categorized based on architecture and failure behavior. Hierarchical method and Composite method are the two approaches which can be used as a solution method in architecture based prediction for reliability. Prevalent approaches to software reliability modeling are black-box based [1], i.e., the software system is considered as a whole and only its interactions with the outside world are modeled, without looking into its internal structure. Several critiques of these time-domain approaches have appeared in the literature and some of these include the fact that they are applicable very late in the life-cycle of the software, ignore information about testing and reliabilities of the components of which the software is made, and do not take into consideration the architecture of the software.

White box reliability modeling is another approach that considers the internal structure of the software. The approaches to architecture-based prediction fall broadly into two categories: *state-based approaches* [7], and *path-based approaches* [13]. The path-based approaches to architecture-based software reliability prediction generally assume that the successive executions of the components are independent. This assumption leads to very pessimistic estimates of reliability, and largely impedes the applicability of these approaches.

2 State-based models

State-based models estimate software reliability analytically. They assume that the transfer of control between modules has a Markov property, that is, model software architecture with a discrete time Markov chain (DTMC), continuous time Markov chain (CTMC), or semi Markov process (SMP). The reliability of software application is estimated either by solving the composite model that combines software architecture with failure behavior (composite method), or by superimposing failure behavior on the solution of the architectural model (hierarchical method). If the reliability improves over time, as faults are discovered and corrected, one would expect that the number of failures detected per unit of time would be decreasing and the time between failures would be increasing.

Table 1. Summary of State based models with uncertainty factors.

S. no	State Based Model	System Architecture	Uncertainty factors influencing the Model
1.	Littlewood (1975) [5]	Irreducible CTMC	Operational Profile, Time between Failure, Component Failure Rate
2.	Littlewood (1979) [6]	Irreducible SMP	Operational Profile, Time between Failure, Component Failure Rate
3.	Cheung (1980) [7]	Absorbing DTMC	Component Reliability, Number of Component Execution, Operational Profile
4.	Laprie (1984) [8]	Irreducible CTMC	Mean execution time of Component, Number of Failures, Time between failures.
5.	Kubat (1989) [9]	SMP	Expected number of Component executions, Time spent in each Component
6.	Littlewood (1995)	DTMC	Total workload, Operational Profile, Time between Failure, Component Failure Rate.

7.	Gokhale (1998) [10]	Absorbing DTMC	Module Reliability, number of module executions, Number of failures.
8.	Ledoux (1999) [11]	Irreducible CTMC	Component Failure Rate, Operational Profile, Time between failures.
9.	Reussner (2003) [22]	DTMC	Individual Component Reliability, Number of Component Executions, Operational Profile.

*DTMC(Discrete time markov chain)

**CTMC(Continous time markov chain)

***SMP(Semi-markov process)

Factors that bring uncertainty in reliability estimation based on Path-based models are the operational profile, Number of component execution, Individual Component reliabilities. Table 1 gives an overview of the various state based models together with the system architecture and uncertainty factors influencing the model.

3 Various Models

3.1 State Based Models

1. Littlewood Model [5] - [6] : This was one of the earliest approach to estimate software reliability. It considers software reliability in terms of operational reliability. Firstly a reliability system with system architecture based upon irreducible CTMC was made [5]. An another approach [6] was developed that consist of a modular program in which transfer of control between modules follow a SMP. This Model describes structure via dynamic behavior using Markov assumption. It analyze both the component and interface failures.

Limitations of Littlewood Model [6]:

- This model considers that failure occur if given input value specification of computation to be performed and output values are incorrect or delayed not considering performance requirements.

- Littlewood Verrall is applicable when there are no failures during testing or when failure data are not available.
- This model suffers from the shortcomings of the composite solution approach which considers both the architectural behavior and failure behavior together.

2. Cheung Model [7] :The early approach (for not continuously running applications) by Cheung uses a discrete time Markov chain (DTMC).The user-oriented reliability model developed to measure the reliability of service that a system provides to user. A simple Markov model is formulated to determine the reliability of a software system based on the reliability of each individual module and the measured intermodular transition probabilities as the user profile. It takes into account that infinite number of component executions may occur until termination of the application execution, e.g. in case of loops.

Enhancements in Cheung Model:

- It measures the reliability of software system with respect to user environment.
- various effects of user profile is discussed, which summarizes the characteristics of users of system on software reliability.
- Sensitivity analysis techniques are developed to determine the modules that are most critical to system reliability.

Limitations of this approach:

- The applicability of this model is restricted to two assumptions:
module reliability independence and transfer of control independence.
- This model may be refined to be considered for component failures due to aging in addition to design errors.
- The model shows that the components have to be extremely reliable in order to produce an acceptable results which is not always the case.

3. Laprie Model [8] :This model which considers only the component failures is the special case of littlewood model [6]. It describes the software system made up of n components by a CTMC.

Enhancements in Laprie Model:

- It considers component interconnection, inter component transition probability, component failures and other statistical information. It is assumed that each component fails with constant failure rate.
- Embedded DTMC of this (n+1) state CTMC is equivalent to DTMC that represent Cheung Model [7] with additional transitions from both exit state and failure state to starting state which represent immediate reset/restart of program execution.

Limitations of this Approach:

- Laprie Model [8] assumes that the failure rates are much smaller than the execution rates which leads to asymptotic behavior relative to the execution process. To overcome this limitation Laprie model considers the hierarchical solution method.

4.Kubat : This model was considered as an enhancement over cheung model. The model proposed by Kubat [9] includes the information about execution time of each component, thus resulting in an SMP as a model of software architecture. Transitions between components follow a DTMC with initial state probability vector $q = [q_i]$ and transition probability matrix $P = [p_{ij}]$.The solution method taken in this work is hierarchical. The reliability of component is estimated as the probability that no failure occurs during its execution. This model was made for certain improvements in cheung model [7].

Enhancements in this Approach:

- Kubat [9] takes into account the execution time of each component for measuring the reliability.
- It provides an approach to measure the each component reliability as compared to cheung [7] in which no proper method was defined.

Limitations of Kubat model:

- It concludes that component failure leads to system failure. some measures are still needed to be taken to avoid this problem.
- Kubat is based on the assumption that components are highly reliable and variances of number of times each component is executed are very small. This cannot be assumed for all types of softwares.
Note that once component reliabilities are estimated the solution approach reduces to the hierarchical treatment of the Cheung model [7].

5. Gokhale [10] describes the architecture by an absorbing DTMC and uses a hierarchical solution method. However, it differs in the approach taken to estimate the component reliabilities. Given time-dependent failure intensity $\lambda_i(t)$ and the cumulative expected time $V_i t_i$ spent in the component per execution of the application, the reliability of component is estimated as

$$R_i = e^{-\int_0^{V_i t_i} \lambda_i(t) dt}$$

Enhancements in Gokhale model:

- It provides an enhanced approach for estimating component reliability considering time dependent failure rates and utilization of component through cumulative expected time spent in component per execution.
- It donot assume that component reliability of software architecture is given. It can be obtained experimentally by testing the application.

Limitations of Gokhale model:

- Gokhale concludes that Component failure leads to system failure. This is not valid statement depending upon whether the part containing bug is executed or not. There must be some approach related to this issue that is needed to be considered.
- Solution to issue of dependency among components was not developed. This is the major drawback in all the previous models as well as Gokhale model.

Note that, the special case of this model that assumes constant failure intensities is equivalent to the special case of Kubat model [9] that assumes deterministic execution times.

6. Ledoux [11] : A general model (Op) is presented and is specifically designed for software systems; it allows the evaluation of various dependability metrics, in particular, of availability measures. Op is an attempt to overcome some limitations of the well-known Littlewood [6] reliability model for modular software.

Enhancements in Ledoux model:

- Ledoux considers the way failure processes affect the execution and deals with the delays in recovering an operational state.
- Primary failure and Secondary failures was discussed, which was one of the limitations in Littlewood model .

Limitations of ledoux model:

Analysis of factors affecting the reliability was not considered.

7. Reussner Model [22] use Markov chains to model system architecture. This method uses the idea to express component reliability not as an absolute value but as function of the input profile of the component. Markov chains are constructed in a hierarchical manner and states include calls to component services in addition to usual component executions. Services may invoke different methods, which may be either internal or external for the component. The reliability of the component is calculated by the reliabilities of the methods, which it uses.

3.2 Path-based models

Path-based models are similar to state-based models, but consider only finite number of component executions traces. It usually correspond to system test cases.

Table 2 .Summary of Path based models with uncertainty factors

S. no	Path based Model	System Architecture	Uncertainty Factors Influencing the Model
1	Shooman (1976) [12]	Markov Structure	Number of bugs in software, system operation time.

2	Krishnamurthy (1997) [13]	Component call graph	Test coverage ,Individual Component Reliability, Number of Component Executions.
3	Yacoub (SBRE) (1999) [14]	CDG	Usage Scenarios, Component Reliability, Transition Reliability.
4	Everett (1999) [15]	CDG	Test Cases, Component Reliability, Number of Component Executions.
5	Hamlet (2001) [16]	Component call Graph	Probability of occurrence of each subdomain, Operational profile, Individual component reliability.
6	Yacoub (SBRA) Enhanced (2004) [21]	CDG	Usage Scenarios, Component Reliability, Transition Reliability.
7	Zhang (Extension to Hamlet 2008) [17]	CDG	Probability of occurrence of each subdomain, Operational profile, Individual component reliability.

1. Shooman model [12] This is one of the earliest models that considers reliability of modular programs, introducing the path-based approach by using the frequencies with which different paths are run.

Based on the assumption that software execution can take fixed number of different Paths. The frequency of occurrence of each path and its failure probability are assumed to be known.

Limitations of Shooman model:

- Dependency among components is not considered as it is important issue to be discussed.
- It donot provide the solution of execution of different paths.
- Some parameters are assumed to be known. No method is defined for there evaluation.
- shooman assumed that failure rate(crash rate) is directly proportional to the number of remaining errors. But this is not valid because two bugs in less frequently occurring code is more reliable than the one bug in frequently occurring one.

2. Krishnamurthy and Mathur model [13] takes an experimental approach to obtain path reliability estimates, a sequence of components along different paths are observed using the component traces collected during testing called Path Traces. Assuming that individual components fail independently of each other, it follows that the path reliability is the product of components reliabilities.

Enhancements in Krishnamurthy model:

- Krishnamurthy addresses the problem of intra component dependency(in case of loops).
- Seeding fault method is applied to obtain component reliability (i.e component and interfaces are identified).
- Information collected using path traces was assumed to be given in shooman [12] .

Limitations of Krishnamurthy model:

- This approach doesnot consider component interface-errors although they are considerable factors in reliability analysis of component based software.
- Estimating reliability based on test cases don't take into consideration the frequency of interaction between components.

3. Yacoub, Cukic and ammar Model [14] [21] takes an algorithmic approach to estimate path reliabilities; a tree-traversal algorithm expands all branches of the graph that represents software architecture. The breadth expansions of the tree are translated as the summation of reliabilities weighted by the transition probability along each path. The depth of each path represents the sequential execution of components, and is hence translated to multiplication of reliabilities.

Enhancements in this approach [14]:

- Analysis based on execution scenarios taking into account algorithmic approach.
- Probabilistic model name CDG is constructed.It guarantees that the loops between two or more components donot lead to a deadlock.
- Using CDG model, we can incorporate effect of frequently executed components,interfaces and links.

Hence we can dedicate more testing and development effort to those critical artifacts.

Limitations of yacoub (SBRE) model [14]:

- The nature of the application: The approach is applicable to component-based application which are analyzed using execution scenarios.
- The algorithm can be used for sensitivity analysis of the application reliability to the variation in the component and interface reliabilities in a given period of execution.
- It does not consider failure dependencies between components.

An Yacoub (SBRA) model [21] have some enhancements:

- Develop a reliability analysis technique that addresses issues related to the distributed nature of software systems, such as the complexity and hierarchical composition of subsystems.
- Algorithm application results in identifying critical components, subsystems, and links which require increased attention in testing, verification, and validation.

Limitations of this approach:

The algorithm does not consider the overall application reliability growth as a function of time. Further, some scenarios may be more critical than others, but they are seldom executed.

4. Hamlet model [16] is also regarded as pathbased, as it considers the actual execution traces of component execution given the mapping from input to output profile. This model tries to address the issue of unavailability of component's usage profile in early system development phases. To do so they do not assume fixed numeric values for reliability but provide model mappings from particular input profile to reliability parameters.

Different input profiles are represented by dividing the input domain of the component to sub-domains and assigning a probability for each sub-domain to occur. This model does not consider explicitly the architecture of the system. Instead, it calculates the output profile of a component, which actually is the input for the next component and is used to calculate latter reliability.

4 Discussion

Although the path based approaches represent the failure behavior of the components using the probability of failure or reliability, the state-based approaches allow component failure behavior to be represented using three types of failure models, namely, probability of failure or reliability [7], [18], [3], [4, constant failure rate [19], and time-dependent failure intensity [20] . The difference in reliability predictions of the statebased and path-based approaches becomes evident only when the control flow graph of the application contains loops. Thus, while state-based models analytically account for the potentially infinite number of paths, pathbased models restrict the number of paths to ones observed experimentally during the testing or terminate the depth traversal of each path using the average execution time of the application [14].

In Path based models, the method of combining software architecture with components and interfaces failure behavior is not analytical. First, the sequence of components executed along each path is obtained either experimentally by testing [13] or algorithmically [14] and the path reliability is obtained by multiplying the reliabilities of the components along that path. Then, the system reliability is estimated by averaging path reliabilities over all paths.

5. Conclusion

The framework presented in this paper addresses the enhancement and limitations of the architecture based reliability models. We have classified these models based on the system architecture approach and the uncertainty factors influencing these models. A notable drawback of path-based approaches is that they provide only an approximate estimate of application reliability when the application architecture has infinite paths due to the presence of loops. Based on the solution methods of architecture based prediction the reliability of software is estimated but still many enhancements are required in the models to overcome the limitations that occurred in the above approaches.

6 References

- [1] W. Farr. Handbook of Software Reliability Engineering, M. R. Lyu, Editor, chapter Software Reliability Modeling Survey, pages 71–117. McGraw-Hill, New York, NY, 1996.
- [2] W. Wang, Y. Wu, M. Chen, An architecture-based software reliability model, in: Proceedings of the Pacific Rim International Symposium on Dependable Computing, 1999, pp. 143–150.
- [3] K. Seigrist, “Reliability of Systems with Markov Transfer of Control,” IEEE Trans. Software Eng., vol. 14, no. 7, pp. 1049-1053, July 1988.
- [4] K. Seigrist, “Reliability of Systems with Markov Transfer of Control, II,” IEEE Trans. Software Eng., vol. 14, no. 10, pp. 1478- 1480, Oct. 1988
- [5] B. Littlewood, A Reliability model for systems with Markov structure, Applied Statistics , 24 (2) 172-177 (1975)
- [6] Littlewood B. Software reliability model for modular program structure, IEEE Trans. Reliability. 1979, : 241-246
- [7] R. C. Cheung, “A User-Oriented Software Reliability Model”, IEEE Trans. Software Engineering, Vol.6, No.2,1980, pp. 118-125
- [8] J. C. Laprie, Dependability evaluation of software systems in operation, IEEE Trans. on Software Engineering , 10 (6) , 701-714 (1984).
- [9] P. Kubat, “Assessing Reliability of Modular Software”, Operations Research Letters, Vol.8, 1989, pp. 35-41.
- [10] S. Gokhale, W.E. Wong, K. Trivedi and J.R. Horgan, “An Analytical Approach to Architecture Based Software Reliability Prediction”, Proc. 3rd Int'l. Computer Performance & Dependability Symp., 1998, pp. 13-22.
- [11] J. Ledoux, Availability modeling of modular software, IEEE Trans. on Reliability, 48 (2), 159-168 (1999).
- [12] M. Shooman, Structural models for software reliability prediction, in: Proceedings of the Second International Conference on Software Engineering, 1976, pp. 268–280.
- [13] S. Krishnamurthy and A. P. Mathur. “On the Estimation of Reliability of a Software System Using Reliabilities of its Components”. In Proc. of Eighth Intl. Symposium on Software Reliability Engineering, Albuquerque, New Mexico, November 1997.
- [14] S. Yacoub, B. Cukic and H. Ammar, “Scenario-Based Reliability Analysis of Component-Based Software”, Proc. 10th Int'l. Symp. Software Reliability Eng., 1999, pp. 22-31.
- [15] W.W. Everett, “Software Component Reliability Analysis,” Proc. Application Specific Software Eng. and Technology, pp. 204-211, Mar. 1999.
- [16] Hamlet, D., Mason, D., Woit, D.: Theory of Software Reliability Based on Components, In Proc. of International Conference on Software Engineering (ICSE), Toronto, Canada, pp. 361-370, May, 2001
- [17] Zhang, F., Zhou, X., Chen, J. and Dong, Y., A Novel Model for Component-Based Software Reliability Analysis, In Proceedings of the 11th IEEE High Assurance Systems Engineering Symposium, (HASE) 2008, pp.303-309, 3-5 Dec. 2008.
- [18] S. Gokhale and K.S. Trivedi, “Reliability Prediction and Sensitivity Analysis Based on Software Architecture,” Proc. Int'l Symp. Software Reliability Eng. (ISSRE '02), Nov. 2002.
- [19] J.C. Laprie and K. Kanoun, “X-Ware Reliability and Availability Modeling,” IEEE Trans. Software Eng., vol. 15, pp. 130-147, 1992.
- [20] S. Gokhale, W.E. Wong, K.S. Trivedi, and J.R. Horgan, “An Analytic Approach to Architecture-Based Software Performance and Reliability Prediction,” Performance Evaluation, vol. 58, no. 4, pp. 391-412, Dec. 2004
- [21] S. Yacoub, B. Cukic, and H. Ammar, “A Scenario-Based Analysis for Component-Based Software,” IEEE Trans. Reliability, vol. 53, no. 4, pp. 465-480, 2004

[22] Reussner, R., Schmidt, H., Poernomo, I.: Reliability prediction, for Component-based Software Architectures, In Journal of Systems and Software, 66(3), Elsevier Science Inc, 2003

[23] Michael R. Lyu , Handbook of Software Reliability Engineering. McGraw-Hill publishing, 1995, ISBN 0-07-039400-8.

What Is the Cost of One IFPUG Method Function Point? – Case Study

Beata Czarnacka-Chrobot

Department of Business Informatics, Warsaw School of Economics, Warsaw, Poland, bczarn@sgh.waw.pl

Abstract - *Software Functional Size Measurement (FSM) methods more and more often are used worldwide as a basis for estimating/measuring the Dedicated Software System (DSS) Development and Enhancement Projects (D&EP) costs. It involves adopting specified per-unit cost measured with regard to the product's functional size unit. In this paper we present a case study on tender competition concerning enhancement of DSS of specific public administration institution in Poland where one of the two potential developers offered possibility to modify such system at the cost of 1 cent per 1 Function Point (FP) of the International Function Point Users Group (IFPUG) method, whereas another one attempted to prove that enhancement at such unit cost was not possible to carry out. The goal of this paper is to analyse likely per-unit costs of the DSS enhancement with regard to 1 IFPUG FP. These issues classify into economics problems of Software Engineering Research and Practice.*

Keywords: dedicated software systems development and enhancement projects, per-unit cost, software size measurement, functional size measurement, IFPUG method, function point

1 Introduction

Like any other product, especially of engineering character, software systems too are characterised by some attributes that should be subject to measurement. The main attribute of every product is its size. However, software engineering cannot boast about such a degree of maturity with regard to the units intended for product size measurement (in this case product being software systems) as other engineering disciplines can (e.g., construction engineering where distinct, precise measure, that is square meter, is being used for the measurement of the apartment size). This constitutes the main cause of the problems with reliable and objective estimation and measurement of such basic attributes of projects aimed at developing, modifying/enhancing and maintaining software systems as work effort, total costs, per-unit costs, execution time or productivity. "Measurement of software size (...) is as important to a software professional as measurement of a building (...) is to a building contractor. All other derived data, including effort to deliver a software project, delivery schedule, and cost of the project, are based on one of its major input elements: software size." [1, p. 149].

However, it is not possible to give answer to the question about above mentioned project's attributes, in particular of per-unit cost, without prior adoption of

adequate, i.e., sufficiently reliable and objective, software system size unit. Among the three measures of software system sizes being used in practice, that is: (1) programming units (e.g., source lines of code), (2) construction complexity units (e.g., object points), and (3) functionality units, this is just functionality units that now are the most widely recognised worldwide [2]. This has been confirmed by the fact they were accepted by the international standardization organizations: ISO (International Organization for Standardization) and IEC (International Electrotechnical Commission) as the only appropriate units of software system size – in the ISO/IEC 14143 norm, which standardizes the concept of the so-called software Functional Size Measurement (FSM) [3].

As a result of many years' verification of particular FSM methods reliability and objectivity, five of them (out of over 25) were recognised by the ISO and IEC as complying with the rules contained in the ISO/IEC 4143 norm and accepted as international standards as well. Those methods include the following:

1. International Function Point Users Group (IFPUG) Function Point (FP) method (ISO/IEC 20926 standard [4]).
2. Mark II (MkII) function point method, developed by the United Kingdom Software Metrics Association, i.e., UKSMA (ISO/IEC 20968 standard [5]).
3. Netherlands Software Metrics Association (NESMA) function point method (ISO/IEC 24570 standard [6]).
4. Common Software Measurement International Consortium (COSMIC) function point method (ISO/IEC 19761 standard [7]).
5. Finnish Software Measurement Association (FiSMA) FSM method (ISO/IEC 29881 standard [8]).

The most popular FSM method, at least in Poland, has been so far the IFPUG function point method ([9]) – and this is the method that in the discussed tender competition was chosen by the client as a point of reference for the offered per-unit costs, that is the costs measured with regard to 1 FP.

It should be mentioned that the IFPUG FP method offers calculation of function points at two levels [10]: (1) the so-called unadjusted FP; (2) the so-called adjusted FP. This is only the level of unadjusted FP that has been recognised as a standard of the software system FSM by the ISO/IEC [4]. Calculating the number of adjusted function points consists in correcting functional size (number of unadjusted FP) using the so-called VAF (Value Adjustment Factor), calculated with the use of 14 pre-defined so-called general system characteristics in order to evaluate the overall complexity of software system. Its purpose is to adjust the previously determined

functional size to the environment of the specific project by taking into account the influence of technical and quality-related requirements on the project. The VAF's range is $\langle 0.65, 1.35 \rangle$, which means that it can adjust functional size by maximum $\pm 35\%$ therefore it does have influence on the system's total cost. Since the publishing of the definition part of the ISO/IEC 14143 norm for the first time (in 1998), per-unit costs have been measured with regard to the functional size (as being recognized by those standardization organizations), i.e., with regard to unadjusted FP – hence further in this paper the IFPUG function points shall be understood as unadjusted FP.

What's more, it should be stressed that what is being considered here are per-unit costs of activities concerning software system *dedicated* to the needs of a specific client which is of significance since in case of commercial software packages designed for a mass consumer, where specified number of licences is being sold (e.g., MS Office), per-unit costs are calculated in a completely different way. Moreover, these activities make up a Dedicated Software System (DSS) Development and Enhancement Project (D&EP), in particular modification/enhancement of the existing system, and they do not contribute to maintenance project, in case of which per-unit costs require analysis of other benchmarking data resources.

Thus in this paper we present a case study concerning tender competition for enhancement of the software system dedicated to specific institution of public administration in Poland where one of the two potential developers offered possibility to modify such system at the cost of 1 cent per 1 IFPUG FP whereas another one attempted to prove that enhancement of the system at such unit cost was not possible to carry out. Hence the goal of this paper is to analyse likely per-unit costs of the dedicated software system enhancement with regard to 1 FP of the IFPUG method, and in particular to compare the offered per-unit cost against the selected resources of benchmarking data.

2 Analysis of data for per-unit costs of DSS enhancement with regard to 1 IFPUG FP

Per-unit costs of the D&EP (i.e., developing from scratch or enhancing the existing software systems) are difficult to estimate if a provider of the dedicated system does not have at their disposal their own resources of appropriate benchmarking data, on the basis of which they would be able to determine their own (organizational) per-unit costs with regard to 1 IFPUG FP. This results from the fact that such data depend on a number of specific factors – on a general level including first of all work costs that vary from country to country as well as type of project, type of software system, field of system application and technological environment of project execution (hardware platform, programming languages used, etc.) as well as many other factors having an effect on a large differentiation of development teams productivity.

However it should be pointed out that relatively few development organizations possess appropriate resources of own benchmarking data as the condition to have them

is not only effective implementation of measurement programmes, what *per se* is not a frequently found phenomenon, but having collected such data for relatively large number of similar projects having been executed in the past and, additionally, referring them to the right unit of software system size (see e.g., [11]). Even more such situation may be found in Poland where FSM methods, including the IFPUG function point method, have been employed for relatively short time [9]. This is when the usefulness of repositories with general data, offered by organizations such as for example International Software Benchmarking Standards Group (ISBSG), comes out. It is worth mentioning that according to C. Jones's estimations there are dozen or so resources of benchmarking data for the discussed types of projects now yet definite majority of them are not widely available. What is more, part of them feature data concerning relatively little number of projects, and also – they not always relate to the IFPUG FP method [12].

2.1 The ISBSG data

2.1.1. The ISBSG data repository

At the moment the ISBSG is an organization that provides the largest, commonly recognised and accessible repository containing general benchmarking data for DSS D&EP whose products are measured with the use of the IFPUG function point method [13]. The ISBSG is a *non-profit* organization that was established in the second half of the 1990s with the mission to enhance processes of software projects execution in business entities as well as in public administration institutions. This mission is being fulfilled by developing, maintaining and exploiting three kinds of repositories with benchmarking data. One of them, the largest one (current version of repository contains data concerning over 5600 projects from 29 countries), comprises data for development and enhancement projects. It is normalised in accordance with the ISO/IEC 15939 standard [14], verified and representative of current technology.

Data collected in the discussed repository are being classified by the ISBSG with regard to the following criteria – they are of importance as they have an effect on how high are per-unit costs with regard to 1 IFPUG FP ([15][16]):

- country where project was undertaken
- context of the project, including: type of organization and business area
- type of project, including: type of activities (enhancement of the system or development of the system from scratch), purpose of the project and size of development team
- type of product, including: type of application and product size (in definite majority of cases expressed in the IFPUG FP)
- project execution environment, including: programming language and hardware platform
- project development methods and tools being used.

However, when using data gathered by this organization one should keep in mind that these data are rather representative of the above-average projects, which results from the following facts:

- Criteria of data collection for ISBSG repository take into account only those organizations that use FSM

methods, including the IFPUG FP method above all, and these organizations are considered more mature than the others as they accomplish programmes concerning implementation of software measures.

- Data to be included to the ISBSG repository are chosen by the providers themselves – they may choose projects that are typical of them as well as projects characterised by the best attributes.
- The ISBSG repository does not include a good deal of data about really large projects.

However, one has to point up that those data are subject to rigorous process of verification with regard to quality. Thus the ISBSG data are valued in the IT industry while general conclusions coming from their analysis are consistent with the conclusions resulting from the analysis of other organizations benchmarking data repositories.

2.1.2. Per-unit costs according to ISBSG data

The ISBSG produces cyclical analytical reports based on the data concerning DSS D&EP. What appears of significance from the perspective of the subject matter being discussed in this paper is the ISBSG report titled “Software Project Costs” [17], which analyses the size of per-unit costs with regard to 1 IFPUG FP. Data analysed therein indicate that:

1. For definite majority of cases, per-unit costs measured with regard to the product functional size unit (1 IFPUG FP) range from USD 300 to USD 1000 per 1 FP, with an average of about USD 750 per 1 FP. Taking into account all analysed projects, the spread is from USD 17 to USD 2727 per 1 FP (extreme values for the so-called outlier projects) while the cost median is USD 716 per 1 FP. These costs are measured by taking into account development team and support personnel (e.g., data base administrators) – they are approx. 15% higher than costs estimated for development teams only.
2. For definite majority of projects, per-unit costs measured with regard to the work time unit (1 hour) range from USD 60 to USD 105 per hour, with an average of about USD 80 per hour. Taking into account all analysed projects, on the other hand, the spread is from USD 7 to USD 570 per hour (extreme values for the outlier projects) while the cost median is USD 69 per hour and the mean is USD 84 per hour. As in the previous case, these are costs measured with development team and support personnel being taken into account.

On the basis of the above, the ISBSG recommends employing the following rules of thumb for the discussed projects:

- 1) cost per 1 IFPUG FP ranges from USD 300 to USD 1000, with an average of about USD 750 per 1 FP
- 2) cost per 1 hour ranges from USD 60 USD to USD 105, with an average of about USD 80 per 1 hour.

What is more, the ISBSG data indicate that PDR (*Project Delivery Rate*)¹ median, that is middle value of

the number of person-hours necessary to deliver 1 IFPUG FP, ranges from about 8 to 11 person-hours per 1 FP – mainly depending on the project type, software system (product) type, application area and technology². Besides, productivity is significantly lower (that is PDR is higher) in case of projects consisting in enhancement of software systems rather than in case of projects consisting in developing such systems from scratch [18, pp. 8, 13, 15, 22]. Taking into account those values together with the cost per hour gives us the spread of costs from USD 480 per 1 FP to USD 1155 per 1 FP, that is on average from USD 640 to USD 880 per 1 FP, which roughly confirms the conclusions coming from the above analysis of the unit cost per 1 IFPUG FP.

Moreover, if project is executed by an outside provider, one should differentiate internal per-unit costs (provider’s per-unit work costs) from external ones (per-unit costs offered by provider to a client, including profit as well). According to the ISBSG, the latter usually exceed internal per-unit costs by 2.5 to 3 times, and in big corporations even by 6 times [19, p. 128].

Per-unit cost measured with regard to 1 IFPUG FP for given types of applications reads for example as follows: web and content management applications – USD 800 per 1 FP, CRM and administration applications – USD 400 per 1 FP, report generators – USD 200 per 1 FP.

2.2 Other sources of benchmarking data

As mentioned above, per-unit costs of DSS D&EP with regard to 1 IFPUG FP depend on numerous factors, which has been the subject of studies carried out by Capers Jones, among others (see e.g., [20, pp. 24-26]). In Table 1 and in Table 2 we present how those costs depend on work costs that vary from country to country. On the other hand, Table 3 shows the so-called effectiveness of exemplary programming languages and several tools, by which we understand the average number of source lines of code required to deliver 1 IFPUG FP depending on the programming language/tool being used.

Table 1. Countries with the highest average per-unit costs (per 1 IFPUG FP) in USD

No.	Country	Per-unit costs (per 1 IFPUG FP)
1.	Japan	1600
2.	Sweden	1500
3.	Switzerland	1450
4.	France	1425
5.	United Kingdom	1400
6.	Denmark	1350
7.	Germany	1300
8.	Spain	1200
9	Italy	1150
10.	USA	1000

Source: [21, p. 29].

number of factors – there are nearly 50 such factors mentioned in the ISBSG repository.

² In this case median is a value more reliable than arithmetic mean as the impact of several atypical (the so-called outlier) projects is thus avoided.

¹ PDR is the inverse of productivity, being the ratio of the number of function points to the effort (work effort). Naturally PDR depends on a

Table 2. Countries with the lowest average per-unit costs (per 1 IFPUG FP) in USD

No.	Country	Per-unit costs (per 1 IFPUG FP)
1.	India	125
2.	Pakistan	145
3.	Poland	155
4.	Hungary	175
5.	Thailand	180
6.	Indonesia	185
7.	Venezuela	190
8.	Columbia	195
9.	Mexico	200
10.	Argentina	250

Source: [21, p. 30].

Table 3. Programming languages table – fragment for the selected languages and tools*

Programming language/tool	Average number of lines of code per 1 IFPUG FP
Assembly languages	320
C	128
Basic (interpreted)	128
COBOL	107
FORTRAN	107
Basic (compiled)	91
Pascal	91
PL/I	80
Ada83	71
Lisp	64
Prolog	64
C++	53
Java	53
Ada95	49
AI Stell	49
Visual Basic	32
Delphi	29
Smalltalk	21
HTML	15
SQL	12
First generation languages (1GL)	320
Second generation languages (2GL)	107
Third generation languages (3GL)	80
Fourth generation languages (4GL)	20
Object languages	30
Report generators	80
Code generators	15
Spreadsheets	6

* This table comprises about 600 programming languages and is continually updated. Its current full version may be found on the Software Productivity Research website: <http://www.spr.com/products/programming.shtm>.

Source: [22, p. 117] and [23, p. 78].

What is more, in the subject literature one may also find a common view about occurrence of the phenomenon of diseconomies of scale in case of DSS D&EP [24]. This means that as the system size (measured e.g., in IFPUG FP) increases, per-unit costs grow too, and they do not decrease instead - which is contrary to the situation taking place in vast majority of other projects, including engineering ones. Data displayed in table 4 confirm this phenomenon, at the same time showing how per-unit costs for development and implementation are being determined.

Table 4. Average per-unit costs per 1 IFPUG FP with regard to the software system size in IFPUG FP

Number of IFPUG FP	Per-unit costs (per 1 IFPUG FP) for development	Per-unit costs (per 1 IFPUG FP) for implementation	Per-unit costs (per 1 IFPUG FP) - total
1501 – 2000	242	725	967
2001 – 2500	255	764	1019
2501 – 3000	265	773	1058
3001 – 3500	274	820	1094
3501 – 4000	284	850	1134

Source: [25].

It should be noted, however, that some studies have appeared recently, indicating quite an opposite phenomenon, that is occurrence of economies of scale in the execution of the discussed projects, which means decrease in costs per unit with the increase in software system size at the same time ([18][24]). This, however, applies only to specific types of systems and those D&EP projects with relatively little increase in product size.

What also is of significance to the subject matter considered here is the fact that according to the studies by C. Jones, consultants carrying out analysis with the use of the IFPUG FP method charge on average USD 5 per 1 function point calculated [26, p. 3.].

3 Concluding Remarks

The above presented data vary greatly - as there is no possibility to derive accurate values for the per-unit cost calculated with regard to 1 IFPUG FP without taking account the specificity of given development organization. Since this cost has influence on a number of factors – major ones were mentioned in the paper. However, lack of own (organizational) resources of adequate benchmarking data continues to be common situation - not only in Poland but worldwide as well. Hence there is the necessity to employ general data.

On the basis of the above presented general benchmarking data for DSS D&EP it should be stated that adopting per-unit cost for enhancement project on the level of 1 cent per 1 IFPUG FP entails the following paradoxes:

- Such cost is 1 700 times lower than the lowest per-unit cost noted in the ISBSG repository – considering per-unit cost for development team alone will not change this fact considerably (then it will be nearly 1 500 times lower).
- Such cost is 30 000 times lower than the lowest per-unit cost recommended by the ISBSG for dedicated software systems.
- Such cost is 75 000 times lower than the average per-unit cost recommended by the ISBSG for dedicated software systems.
- Given that this is an internal cost, the costs of 8 to 11 hours of work are estimated to be 1 cent – yet enhancement is characterised by significantly lower productivity than development of the system from scratch. In case of external cost, those costs are estimated to be even lower as the internal per-unit cost, with the lowest difference resulting from the ISBSG data being taken into account, is 0.4 cent per 1 FP.

- A question then arises whether this very low per-unit cost already takes into account the phenomenon of diseconomies of scale, that is increase of such cost together with the increment of system size.
- Comparing with the per-unit cost of the cheapest per system size unit types of application, such cost is 20 000 times lower.
- Comparing with the average per-unit cost for Poland such cost is 15 500 times lower (additionally it should be assumed that per-unit costs for Poland have grown since 2000 due to the increase in work costs).
- Assuming that even most efficient programming languages (of fourth generation) will be used for the software system enhancement, such per-unit cost means that writing 2 000 lines of code costs USD 1 on average.
- This cost is 500 times lower than the average consultant's pay for 1 calculated function point – even if this pay is significantly lower in Poland, it still without doubt is repeatedly higher than 1 cent.

In view of the above paradoxes, mostly diametrical differences resulting from the comparison of general benchmarking data with the adopted per-unit cost on the level of 1 cent per 1 IFPUG FP, place and time factors (as well as inflation related to them) do not really matter, similarly as the fact whether the per-unit cost is an external or internal cost.

Thus it should be stated that both general data, those collected in ISBSG repository and those coming from other sources having been recognised in the IT industry, as well as common sense rules of rational economic approach unequivocally indicate that *it is not possible to develop, and in particular to enhance software system dedicated to the client's needs at the cost per unit amounting to 1 cent per 1 IFPUG FP*, at the same time assuming the lack of subsidization for those works with maintenance costs or other project-related costs, which naturally should not have happened.

It is worth mentioning that the analysis of likely per-unit costs of the DSS enhancement with regard to 1 FP of the IFPUG method carried out following the above described manner resulted in client rejecting the provider offering such costs in the tender competition being considered.

4 References

- [1] M. A. Parthasarathy. "Practical software estimation: function point methods for insourced and outsourced projects", Addison Wesley Professional, 2007.
- [2] B. Czarnacka-Chrobot. "The Economic Importance of Business Software Systems Functional Size Measurement", Software Engineering, vol. 1, no 1, Scientific & Academic Publishing, Rosemead, California, USA, 2011, pp. 9-23.
- [3] ISO/IEC 14143 Information Technology – Software measurement – Functional size measurement – Part 1-6, ISO, Geneva, 1998-2011.
- [4] ISO/IEC 20926 Software and systems engineering - Software measurement - IFPUG functional size measurement method 2009, edition 2, ISO, Geneva, 2003-2009.
- [5] ISO/IEC 20968 Software engineering – Mk II Function Point Analysis - Counting practices manual, ISO, Geneva, 2002.
- [6] ISO/IEC 24570 Software engineering – NESMA functional size measurement method version 2.1 - Definitions and counting guidelines for the application of Function Point Analysis, ISO, Geneva, 2005.
- [7] ISO/IEC 19761 Software engineering – COSMIC: a functional size measurement method, edition 2, ISO, Geneva, 2011.
- [8] ISO/IEC 29881 Information Technology – Software and systems engineering – FiSMA 1.1 functional size measurement method, ISO, Geneva 2010.
- [9] B. Czarnacka-Chrobot. "Analysis of the Functional Size Measurement Methods Usage by Polish Business Software Systems Providers"; in: Software Process and Product Measurement, A. Abran, R. Braungarten, R. Dumke, J. Cuadrado-Gallego, J. Brunekreef, Eds., Lecture Notes in Computer Science, vol. 5891, pp. 17–34, Springer-Verlag, Berlin-Heidelberg, 2009.
- [10] IFPUG, "Function point counting practices manual, release 4.3", Part 0-5, International Function Point Users Group, NJ, January 2010.
- [11] B. Czarnacka-Chrobot. "The Role of Benchmarking Data in the Software Development and Enhancement Projects Effort Planning"; in: New Trends in Software Methodologies, Tools and Techniques, H. Fujita, V. Marik, Eds., Frontiers in Artificial Intelligence and Applications, vol. 199, pp. 106-127, IOS Press, Amsterdam-Berlin-Tokyo-Washington, 2009.
- [12] C. Jones. "Sources of Software Benchmarks", Version 13, Capers Jones & Associates LLC, November 2011.
- [13] <http://www.isbsg.org>.
- [14] ISO/IEC 15939 Systems and software engineering – Measurement process, ISO, Geneva 2002-2007.
- [15] International Software Benchmarking Standards Group. "Release 10 Repository Demographics", ISBSG, Hawthorn, VIC, January 2007.
- [16] International Software Benchmarking Standards Group. "Data demographics release 11", ISBSG, Hawthorn, Australia, June 2009.
- [17] International Software Benchmarking Standards Group. "The ISBSG Special Analysis Report: Software Project Costs", ISBSG, Hawthorn, VIC, June 2005.

- [18] Ch. Symons. "The Performance of real-time, business application and component software projects", COSMIC and ISBSG, September 2009.
- [19] "Practical Software Project Estimation", P.R. Hill, Ed., ISBSG, McGraw-Hill, 2010.
- [20] L. Buglione. "Some thoughts on productivity in ICT projects, version 1.3", WP-2010-01, White Paper, August 23, 2010.
- [21] C. Jones. "Software Benchmarking: What Works and What Doesn't?", Boston SPIN, November 2000.
- [22] M. Flasiński. "Zarządzanie projektami informatycznymi" [„IT Projects Management”], PWN, Warsaw, 2006.
- [23] C. Jones. "Software Assessments, Benchmarks, and Best Practices", Information Technology Series, Addison-Wesley, 2000.
- [24] B. Czarnacka-Chrobot. "(Dis)economies of Scale in Business Software Systems Development and Enhancement Projects"; in: Proceedings of the 10th International Conference on Software Engineering Research and Practice (SERP'11), The 2011 World Congress in Computer Science, Computer Engineering & Applied Computing (WORLDCOMP'11), H.R. Arabnia, H. Reza, L. Deligiannidis, Eds., Vol. I, pp. 80-86, CSREA Press, Las Vegas, Nevada, USA, 2011.
- [25] P. Ratford, R. Lawrie. "The Role of Function Points in Software Development Contracts", White Paper, Charismatek, 2000.
- [26] C. Jones. "A New Business Model for Function Point Metrics", Version 10.0, Capers Jones & Associates LLC, August 2009.

Software Reuse Cost Factors

Hisham M. Haddad, Nancy R. Ross, and Woranuch Kaensaksiri

Department of Computer Science, Kennesaw State University, Kennesaw, GA, USA

Abstract - Software organizations are encouraged to adopt reuse strategies into their development processes. Organizations use software reuse cost estimation models to assess the feasibility of investment in reuse programs. Each organization is unique in its resources and capabilities. Thus, the initial logical step in establishing a reuse program is to examine relevant reuse-specific cost factors applicable to the organization's situation, and then utilize relevant estimation models to assess the feasibility of such investment. This work investigates cost factors associated with adopting software reuse independent of cost estimation models. It highlights cost estimation models for software reuse, and outlines possible combinations of cost estimation models for identified cost factors to help organizations gain better understanding of the investment in reuse strategies.

Keywords: Software reuse, reuse cost factors, reuse cost estimation, reuse cost estimation models.

1 Introduction

When an organization decides to integrate software reuse into its development process, many factors must be considered. Reuse is a long-term investment that can bring about improvements in productivity, quality, and reliability as well as cost reductions. Reuse is not free [1], it requires resources to create and maintain reusable work products, a reuse library, and reuse tools. In fact, the benefits brought by a program of systematic reuse may not cover the costs. Accurate cost analysis of a reuse program is necessary in order to decide whether to utilize component reuse or not. Reuse is not always the appropriate solution and at times, the development from scratch is the feasible approach [2,3].

As each organization is unique in its resources and capabilities, the initial step is to examine cost factors applicable to the organization's circumstances. Reuse cost factors provide insight into how adopting a reuse strategy affects the entire software development life cycle. In fact, a separate software reuse lifecycle must be considered as part of the overall software lifecycle. Furthermore, several software reuse cost estimation models have evolved over the years, such as software-size-based metrics; metrics for measuring the amount of reuse in an application; and more complicated models measuring the many costs involved in reuse programs. These models can be used to assess the feasibility of adopting reuse into the development process, and they can influence the adoption of more accurate reuse cost estimation practices. However, understanding the key cost factors that affect software reuse, independent of cost estimation models, is the first step. This work describes reuse cost factors; draws a comparison of estimation methodologies; and suggests combinations of models to adequately cover identified software reuse cost factors.

The research methodology for this work is based on a study of relevant research and qualitative analysis of software reuse cost estimation methodologies to highlight the cost factors on which these models are based. The various approaches to cost estimation are also contrasted, and related combinations of such approaches are identified. A limitation in this study was the accessibility to documented practical experiences in cost estimation for real-world projects. Such information was not readily available, and organizations are reluctant to share.

2 Software development with reuse

The cost of reuse is distributed throughout the development lifecycle. The software reuse lifecycle involves two groups: the producer team and the consumer team. Producers are responsible for domain engineering and component engineering. *Domain engineering* determines whether it is feasible to develop a reusable asset; while *component engineering* includes component specification, development for reuse, verification and validation, reuse certification, and storage (phases of the software reuse lifecycle). Consumers are tasked with application engineering and corporate engineering. *Application engineering* involves requirements specification, retrieval, assessment, instantiation (using a component as is), adaption, and integration phases of the software reuse lifecycle. *Corporate engineering* involves the production of reusable assets in domain engineering and their use in application engineering.

During component engineering, the producer decides upon the necessary functions, interfaces, and performance for a specific asset. The requirements specification and detailed design document is produced during component engineering, and the reusable components are developed from the specifications in the detailed design document during the development phase. The reusable assets are verified, validated, and certified before they are stored in the software reuse library. Using requirements elicited by the consumer, the consumer searches the repository to identify and retrieve reusable assets. These assets are then evaluated to determine if they can be used as is or modified to fit the consumer-identified specifications. The consumer then integrates the components into the new applications [4].

According to various researchers, software designed for reuse is much more costly than software developed for a specific application. Software designed for reuse requires 20-25% more time to develop and to learn how to use at the beginning of a software reuse initiative. The cost of making software reusable has been found to be 60% higher than development for single use due to the additional effort required for generalization, documentation, testing, and component library support and maintenance. The cost of creating a reusable component is about two times that of creating a non-reusable version, and costs to integrate reused components into new

components range from 10-20% of the cost of creating non-reusable versions. The relative cost of producing a reusable component ranges from 120-140% of the cost of creating a non-reusable version, and integration costs range from 10-63% of the cost of creating a non-reusable version [22]. In addition, the adoption of a reuse program involves risk and the choice of a reuse strategy can be crucial to its success [5].

3 Software reuse cost factors

During development, the cost factors for incorporating reusable components fall into seven categories, described below. The cost factors involved in adopting reusable components into the development process are identified, an essential step in generating accurate cost estimates.

1. Identification and acquisition costs:

Before searching for reusable components, the producer must develop a complete description of the product and identify the requirements and environment characteristics from the consumer. With these requirements, the producer develops components generically in order to allow for future reuse. The producer may also find it necessary to modify the existing process to allow for reuse. Component identification and acquisition costs represent the cost of mining and acquiring reusable assets that involves the effort required in searching for the appropriate asset whether it exists in a reusable component repository, in other areas within the organization, in the public domain, or in the market. Such costs include the trial, verification, and subsequent purchase of an asset identified for reuse.

Domain analysis costs are generated when defining the scope and the content of reusable assets, including classification of the repository assets into categories in order to determine which candidate reusable artifacts have common functionality, and mining potential assets from past products. During reuse of a software artifact, great effort must be devoted in the retrieval of associated artifacts since reusable components may be found in both intra-organization repositories and component markets. This may involve the research classification methods, search for keywords, and choosing appropriate components by inspecting their functionalities in order to discover a reusable component. Furthermore, royalties or license fees may apply. Acquisition costs may be higher when a component is purchased, but the higher acquisition costs may be balanced by fewer revisions, which could potentially result in lower modification costs.

Additional costs for mining and acquiring reusable assets are: 1) the cost of technical staff and consultants in identifying needed components for an application; 2) any costs incurred in making a reusable component or system function properly in order to evaluate its potential reuse in the new application (including media conversions, implementation differences, non-current documentation, and cost of existing functionality evaluation for potential reuse); 3) the creation of a design document for planning and implementation of components reuse; and 4) the purchase price and maintenance fee acquired from outside the company [6,7,8,9,18].

Assets may be transitioned for reuse in the following ways: 1) buying a copy of a repository component for a certain product and making changes to it via white-box reuse (also known as cataloged asset acquisition); 2) buying a copy of a repository component and using it without making changes (known as black-box reuse); 3) buying an existing component from another product (known as mining and cataloging); 4) using a copy of a component that is not cataloged in the repository but that an employee has knowledge of (copy and paste); and 5) purchasing a component from another organization and adding it to the repository (external acquisition).

2. Modification costs: During modification for reuse, there are two methods that may be used in the transformation of repository assets: 1) adaptation for reuse, which is the modification of an existing repository asset; and 2) white-box reuse, which involves the revision of one asset into another asset inside the same application. Asset modification costs include: 1) additional effort required to modify other artifacts in order to integrate a reused component into the new product when white-box, black-box, or commercial off-the-shelf software is reused.; 2) the cost of interface design between the application and the repository asset - all functions of the reusable component and all attributes that make up the interface between the application and the reusable component must be identified and specified properly in order to achieve the desired functionality; and 3) the cost of reformatting data when an application is migrated from one platform, database management system, or operating system to another. For instance, a reusable component developed in C# and uses Microsoft Access database. The new application is developed in Java using an Oracle database, so the changes required for component reuse may require a great deal of effort [6,9].

3. New development costs: Development costs of new assets are also involved in software reuse. These costs fall into two categories: producer and consumer. Producers construct a new repository asset from scratch in a manner in which the asset will conform to specified standards that allow for reusability, and consumers incorporate the reusable assets developed by producers into software components. Consumers may develop new components as needed in order to integrate reusable software into their application [10].

When finding needed components in the reuse repository, the consumer must decide if any additional components are needed, if it is viable to build the product from the reusable components, the code needed for component integration, and the additional code needed to satisfy the requirements. If reuse is economical, the consumer chooses the necessary reusable components, codes additional required components, assimilates the retrieved components as black box reuse or white box reuse, and develops integration code to make the components work together [4].

4. Integration and testing costs: Product integration costs include: 1) the cost of partial and full integrations; 2) the cost of data transfer from a previous application to a new application in order to verify and test a component in preparation for reuse; and 3) the cost of design reviews which require effort to review a document and prepare an abstract. Integration costs also include verification and validation

activities (i.e., technical design reviews, formal code walkthroughs, and unit test plans) that are involved in software reuse coding costs of new components as well as the same costs performed directly on the reused assets.

Product testing costs include: 1) the development of a test environment; 2) unit testing and debugging; 3) acceptance testing; 4) subsystem and system testing; and 5) testing of functionality by a quality control engineer [6,9]. The interaction of the reused component with the system must be thoroughly tested to guarantee that the functionality is performing as expected. After development of the entire product, both the producer and the consumer perform tests, potentially reusing test cases and test data, to determine whether the functionality was built according to specification and validate the software to determine if the system accomplishes the intended goal [4].

5. Infrastructure costs: Prior to instating a reuse program, a new development process implementing reuse and a reuse repository must be established [2]. Costs to establish and maintain the repository include: 1) database analysis and design; 2) the cost of tool development or purchase, which requires textbook or online training in the new technology; and 3) the cost of database administration. The cost to store and catalog repository artifacts includes 1) the cost of the time required for the approval of artifacts for the repository; 2) the cost of analyzing the metadata needed in order to employ efficient searches of artifacts in the catalog; and 3) the cost of a mechanism for the retrieval of assets from the catalog. In the storage phase, the producer must classify and store assets that will be maintained in a repository for consumer retrieval [4,6,8].

The actual development cost of asset reuse cannot be determined precisely. Some costs may not apply since the catalog rarely provides a complete product; however, multiple assets may be constructed into a complete product in which several assets have been reused. The cost of development of such an asset may be the sum of its integration, verification, and validation costs.

4 Reuse cost estimation models

When the development of a new software project begins, it is necessary to compare the cost of developing new components versus integrating reusable components. To determine the cost of reuse, several cost estimation models have been developed. In this work, we focus on six common models, briefly described below.

1. COCOMO-II model (Boehm): The original COCOMO model predicts the length and effort of a project by drawing an association between the size of the system and various cost factors. The factors were weighed based on the project's domain, environment, and limitations in order to convert the estimated project size into estimated person-months which could be broken down into staff size and project length. In addition, the equations include fifteen cost drivers for each phase of a project (product design, detailed design, coding/unit test, and integration test). In 2000, the model was extended to COCOMO-II, which presented three sizing

options: object points (used during the application composition model), function points (used during the early design model), and lines of source code (used for the post-architectural model) [9,11,17]. When reusing software, the percent of reuse is determined, and the object point count or function point count is adjusted.

2. Reuse-Based model (Jasmine/Vasantha): The model is based on reuse cost metrics. Metrics are considered important in deciding whether to modify a reusable component, and they can be used to better measure, manage, and plan software application development. Such metrics may give engineers the information they need to make decisions in technical areas, and they may give management the information needed in making decisions regarding project planning. The two categories of metrics are: 1) product metrics that establish component characteristics and 2) process metrics that involve measurements of cost and time among other things. The model considers many cost factors associated with reusable components, including domain analysis, explicit documentation to increase the ease of implementing a reusable component, maintenance of and revisions to of reuse documents and components, costs in the form of licenses and royalties for artifacts acquired from other organizations, purchase, installation, and operation of a reuse repository, and costs associated with training personnel in reuse design and incorporation. Management should also consider the number of times a component is expected to be reused when building reusable components [3].

3. Basic-Reuse model (SPC): This basic reuse costing model [12] was developed by the Software Productivity Consortium (SPC). The model takes into account the following cost factors: the cost of software development, the relative cost to reuse software, the proportion of reused code in the product, the relative cost of developing a reusable asset, and the number of reuses over which the asset development costs will be amortized.

4. Rate-Based model (Intermetrics): This ad-hoc method is presented from the experience of Intermetrics, Inc. Under this model, much of the reuse cost has to do with the energy and time spent in searching for and retrieving enough of the necessary reusable components required to work together to produce the desired functionality as well as determining the condition in which the reusable software components are found. From its experience in the development of a reusable software system, Intermetrics identified the following cost factors: data transfer, data reformatting, document review, abstract preparation, facet and keyword preparation, configuration management, reusable component testing, environment testing, outside resource personnel, and consulting.

5. Risk-Based model (Lim): This model determines reuse value by adding total of reduced and avoided consumer costs to the greater profit generated by software reuse and subtracting out the producer costs. This number is multiplied by a probability that factors in risk and adjusts the result in order to consider the value of money over time. The model utilizes the following cost factors: cost to create product without reuse, cost to create product with reuse, cost to create

an asset for reuse, profit from increased revenues enabled by reuse, probability of receiving cash flow, interest rate by which cash flows are discounted, number of time periods under consideration, and net Present Value [10].

6. Product-Line model (Tomer): A software product line is a group of software applications using a shared set of components that meet specific requirements of a certain application domain. The product line approach achieves considerable savings and is economical due to the reuse and integration of related components from the common asset library [13]. In this approach, transformation and transition are involved in the cost of a reusable component. The transformation operation's cost consists of the cost required to modify the reusable component; while the transition operation's cost is made up of the cost incurred in copying code from one component into another in the application or black-box reuse [6]. The model uses these cost factors: cost of adaptation for reuse, cost of development from scratch, cost of white box reuse, cost of constructing the target private artifact from scratch as new development, cost of catalog acquisition (replicating a copy of repository), cost of copy and paste, cost of mining, and cost of externally acquiring.

In addition, the calculations of reuse costs depend upon the reuse scenario put into place, such as Systematic Reuse Cost for adapting assets from artifacts mined during domain analysis; Systematic Reuse Cost for new assets developed from scratch; Controlled Reuse Cost; Opportunistic Reuse Cost; and Pure Development Cost. In order to determine the most cost-effective reuse scenario that should be used in the production of the desired functionality from the original source assets, one must be able to compare the costs of other possible reuse scenarios. Reuse scenarios are any sequence of transition and transformation operations performed while practicing reuse [12,13].

5 Comparison of estimation models

In order to compare the models analyzed in this work, Table-1 depicts a list of common cost factors. The cost factors from each model are linked with the common cost factors in Table-2 in order to portray the similarities and differences in each model. To be clear, a producer is a programmer who creates reusable components from scratch while a consumer is a programmer who uses reusable components to create other applications.

Table-1: Cost Factors of Reuse Cost Estimation Models (adapted from [10]).

QUANTITIES		LINES OF CODE	
NR	Number of reuses	ASLOC	Adapted source lines of code in project
CM	Percentage of code modified	RLOC	Reused lines of code in product
DM	Percentage of design modified	RLOCL	Reusable lines of code in library
IM	Percentage of integration effort required	PLOC	Lines of code in product
CONSUMER		PROFIT	
Cc,wrp	Cost to create product without reuse	P	Profit from increased revenues enabled by reuse
Cc,rp	Cost to create product with reuse	RISK	
Cc,mwr	Cost to maintain product created without reuse	p	Probability of receiving cash flow
Cc,mr	Cost to maintain product created with reuse	pl	Probability of asset being found in library
Cc,ra	Cost to consumer to reuse asset	TIME VALUE	
Cc,r1	Cost of royalties and licenses for external assets	i	Interest rate by which cash flows are discounted
Cc,r2	Cost to adapt asset	M	Number of time periods under consideration
Cc,r3	Cost to acquire, generalize, search, and retrieve	PRODUCER	
Cc,r4	Cost to utilize assets: instantiation training	Cp,r	Cost to producer to create asset for reuse
Cc,r5	Cost to reuse: identify, retrieve, understand, validate, integrate, and test an asset	Cp,lm	Cost to producer/maintain reusable assets in the library; configuration and change management
Cc,r6	Cost of incentive paid to component developers	Cp,wr	Cost to create non-reusable version of asset
DOCUMENTATION		OVERHEAD	
DD	Documentation development	O1	Library overhead costs
DU	Documentation maintenance	O2	Cost of Domain Engineering
OUTPUT		O3	Infrastructure costs - repository mechanisms
E	Effort in programmer months	O4	Infrastructure costs - domain analysis, architecture, training
C	Cost of effort		
NPV	Net present value		
SZ	Project size – LOC, object, or function points		

Table-2: Comparison of Software Reuse Cost Models by Cost Factor.

	COCOMO-II (Boehm)	Reuse-Based (Jasmine/Vasanth)	Basic-Reuse (SPC)	Rate-Based (Intermetrics)	Risk-Based (Lim)	Product-Line (Tomer)	
Cost Factor							Total
<i>Quantities</i>							
NR			Y				1
CM	Y						1
DM	Y						1
IM	Y						1
<i>Consumer</i>							

Cc,wrp		Y			Y		2
Cc,rp					Y	Y	2
Cc,mwr							0
Cc,mr		Y					1
Cc,ra			Y			Y	2
Cc,r1							0
Cc,r2	Y	Y				Y	3
Cc,r3						Y	1
Cc,r4							0
Cc,r5	Y	Y		Y		Y	4
Cc,r6							0
Documentation							
DD				Y			1
DU				Y			1
Lines of Code							
ASLOC	Y						1
RLOC	Y		Y				2
RLOCL							0
PLOC	Y		Y				2
Profit							
P					Y		1
Risk							
p					Y		1
pl		Y					1
Time Value							
i					Y		1
M					Y		1
Producer							
Cp,r			Y		Y	Y	3
Cp,lm				Y			1
Cp,wr							0
Overhead							
O1							0
O2							0
O3							0
O4							0
TOTAL	8	5	5	4	7	6	35
Comparison of Reuse Cost Estimation Models by Output							
E	Y	Y		Y			3
C			Y	Y		Y	3
NPV					Y		1
SZ	Y						1

5.1 Analysis of similarities

Thirty-three cost factors were considered in this study. Table-2 reveals the following commonly used cost factors:

- Cost to reuse: cost to identify, retrieve, understand, validate, integrate, and test an asset (4 models)
- Cost to producer to create asset for reuse (3 models)
- Reused lines of code in product (2 models)
- Cost to adapt asset (2 models)
- Cost to create product/system without reuse (2 models)
- Cost to create product/system with reuse (2 models)
- Lines of code in product (2 models)
- Cost to consumer to reuse asset (2 models)

The most common cost factors make up fifteen of the total of thirty-five instances resulting from the comparison or 43% similarity, and they make up eight of the thirty-three or 24% of the cost factors used for the

comparison. The greatest similarity among the models involves the costs to create or adapt components for reuse. The most common forms of output for the reuse cost models are effort (38%) and cost of effort (38%), taking into consideration that some models provide more than one output. As far as costs to the consumer, Jasmine/Vasantha, Tomer, and Boehm models all include the cost to adapt an asset and the cost to acquire reusable assets in their calculations. The adaptation of an asset includes the adoption of requirements as well as code and test cases among other artifacts. The consumer may choose to modify the artifact, incorporate it in a black box way, or modify it and integrate it into the system.

Both Boehm and SPC include lines of code in the product (PLOC) as well as reused lines of code in the product (RLOC) in their calculations. For Boehm, PLOC is similar to his equivalent source lines of code, and for SPC, RLOC and PLOC are used to determine the proportion of reused code in the product. Producer costs are included in the calculations of the Rate-based, SPC, Lim, and

Tomer's models. All of them include the cost to the producer to create an asset for reuse. These costs include the creation of new requirements that are necessary in the integration of the reused design as well as the development of new software components that allow the reused component to operate properly with the existing software.

All of the identified costs in the list of similarities specified above occur in the design, development, and integration/test phases. When reuse is implemented, requirements, code, and test cases are affected. Documentation of the reused artifact as well as the developed/modified artifact becomes very important in order for the reused components to be implemented more easily and accurately. The overhead costs of reuse are not included in the formulas of any models; however, Jasmine/Vasantha's approach considers additional overhead reuse costs, specifically domain analysis, creation and operation of a reuse library, and training of personnel in design for and with reuse.

None of the six models explicitly considers the time required to create or revise a particular component. Such measurements include the start and end times of an assignment or project duration. Some examples of this are the date on which a consumer begins to create a product and the date on which the consumer completes the product with reuse or without reuse; the date on which the consumer begins to incorporate an upgrade and the date on which the consumer completes the incorporation and upgrade with or without reuse; and the date on which a producer begins creating an upgrade and the date on which the producer completes the creation of the upgrade [26].

5.2 Analysis of differences

Not one of these models indicates whether the producer or consumer should be responsible for the cost of a reusable asset. The producer and consumer information found in this paper comes from Lim's 1996 paper [10] on Reuse Economics. Both Jasmine/Vasantha and Lim include the cost to create a product/system without reuse in their calculations of the cost with reuse. Lim and Tomer's calculations of the cost of development with reuse include the cost to the consumer to create a product with reuse.

Only Jasmine/Vasantha considers the cost to the consumer to operate and maintain a product/system created for reuse, but they do not include this information in their calculations. They simply suggest that this information be considered when determining the cost of reuse. Only SPC and Tomer include the cost to the consumer to reuse an asset in their calculations. SPC uses the relative cost to reuse software, and Tomer uses the cost of black box reuse. Only Tomer's Controlled Reuse Cost formula uses the cost to the consumer of external acquisition, searching, and retrieval.

As far as producer costs, Boehm, Jasmine/Vasantha, and

the Rate-based model do not consider the cost to the producer to create a new asset for reuse. Jasmine/Vasantha focus on consumer costs, the Rate-based model focuses on the search for reusable components, and Boehm focuses on estimation of effort using a reuse percentage and an adaptation adjustment factor. Only the Rate-based model considers the cost of configuration management. The importance of documentation to the reuse effort is only mentioned in one model (Jasmine/Vasantha) - increased documentation facilitates reuse, and maintenance and enhancement of reuse documents is suggested; However, even in this model, the cost of documentation is not specified in any formula.

5.3 Combinations of models

Although each of the identified reuse cost estimation models cover from five to eight identified cost factors, no one estimation model covers all of those cost factors. Because of this, combinations of several methods may give a more accurate estimation of the actual cost of such a project. Table-3 shows the possible combinations of cost factors among the six estimation models investigated in this work and the model that produces it. This information helps managers and engineers determine more accurate estimates for cost factors that apply to their situation.

Table-3: Cost factors and associated estimation models.

Cost Factor	Model
Consumer Costs	
Cost to consumer to create product/system without reuse	Reuse-Based
Cost to consumer to create product/system with reuse	Product-Line
Cost to consumer to reuse asset	Product-Line
Cost to consumer to adapt asset	Product-Line
Cost to consumer to acquire reusable assets: acquisition, generalization, searching, and retrieval	Product -Line
Cost to consumer to reuse: identify, retrieve, understand, validate, integrate, and test an asset	Product-Line
Cost to develop documentation	Rate-Based
Cost to maintain documentation	Rate-Based
Producer Costs	
Cost to producer to create asset for reuse	Product-Line
Cost to producer for configuration management	Rate-Based
Cost to producer to develop documentation	Rate-Based
Lines of Code	
Adapted source lines of code	COCOMO-II
Reused source lines of code	Expert Judgment
Source lines of code in product	Analogy-Based

Table-3: Cost factors and the Expert Judgment method involves multiple experts who provide estimates based on experience. PERT and the Delphi technique, expert-consensus methods, are used to work out differences in the various estimates [14]. The Analogy-Based estimation method formulates the cost

estimation for the new project based on the actual costs of one or more previous similar projects. Through analogy, it involves the following stages: 1) select relevant project size estimates; 2) retrieve data of all historical projects and compute the similarity; and 3) estimate the effort required for the new project [15,16]. With such a combination of approaches, fourteen of the thirty-three identified cost factors would contribute to the reuse cost estimation. This is almost twice the number of cost factors covered by COCOMO-II, which appears to be the most thorough of the approaches to cost estimation.

6 Conclusion

No guidance exists for the analysis and selection of alternative approaches for software reuse including no reuse. Instituting a successful reuse strategy should be based on understanding of the factors that impact reuse in the organization's context. None of the models researched in this work takes into consideration all costs that a software reuse program might generate. The comprehensive list of these costs might be considered, but they are not included in the formulas specified by these models. This work identified thirty-three reuse-specific cost factors. These factors were mapped to six common costing models. The results of this work also shows possible combination of cost factors among the studied estimation models, given an organization alternative ways to obtain accurate estimate of instituting a reuse program.

AN effective reuse strategy may require changes in organizational structure and additional personnel to manage reusable assets and to stay informed about reuse products in the software community. The *Reuse Manager* must have knowledge of organizations supporting reuse, reuse economics, reuse metrics, and reuse products. The reuse repository *Librarian* provides a useful, usable, and efficient library to the producers and consumers; ensures the generality, correctness, reliability, and clarity of the components in the repository; and determines the best way to represent and catalog reusable assets. The *Domain Analyst* explores application domains to determine components to be included in the library; identifies commonly recurring assets and/or problem solving patterns; and decides whether the source code assets or design patterns are appropriate for the specific domain. The *Application Engineer* decides how to implement selected reusable components (blackbox or whitebox); and determines the risks (program quality and programmer productivity) when modifying a component to develop a new component from scratch, or to use the component "as is" in the application. Components are modified and re-tested when whitebox is chosen.

7 References

- [1] Lim, W. C. (1994). Effects of reuse on quality, productivity, and economics. *IEEE Software*, 11(5), 23-30.
- [2] Succi, G., and Baruchelli, F. (1996). Analysing the return of investment of reuse. *ACM SIGAPP Applied Computing Review.*, 4(2), 21-25.
- [3] Jasmine, K.S., and Vasantha, R. (2008). Cost estimation model for reuse based software products. *Proceedings of the International MultiConference of Engineers and Computer Scientists 2008*, 951-954.
- [4] Chmiel, S. F. (2000). An integrated cost model for software reuse. (Doctoral Dissertation). Retrieved from the ACM Digital Library database.
- [5] Rothenberger, M. A., Dooley, K.J., and Kulkarni, U. R. (2003). Strategies of software reuse: A principal component analysis of reuse practices. *IEEE Transactions on Software Engineering*, 29(9), 825-837.
- [6] Tomer, A., Goldin, L., Kuflik, T., Kimchi, E., and Schach, S. R. (2004). Evaluating software reuse alternatives: A model and its application to an industrial case study. *IEEE Transactions on Software Engineering*, 30(9), 601-612.
- [7] Ravichandran, T., and Rothenberger, M. A. (2003). Software reuse strategies and component markets. *Communications of the ACM*. 46(8), 109-114.
- [8] Krutz, W. K., Allen, K., and Olivier, D. P. (1991). The costs related to making software reusable: Experience from a real project. *Proceedings of the Conference on TRI-Ada' 91: Today's Accomplishments, Tomorrow's Expectations*, 437-443.
- [9] Boehm, B., Abts, C., and Chulani, S. (2000). Software development cost estimation approaches - A survey. *Annals of Software Engineering*, 10(1-4). Springer, Netherlands, November 2000.
- [10] Lim, W. C. (1996). Reuse economics: A comparison of seventeen models and directions for future research. *Proceedings of the Fourth International Conference on Software Reuse 1996*, 41-50.
- [11] Kemerer, C. F. (1987). An empirical validation of software cost estimation models. *Communication of the ACM*, 30(5), 416-429.
- [12] Data & Analysis Center for Software (DACs). (2010). Assess reuse risks and costs. *Software Tech* 13(3). Retrieved on April 29, 2012 from the World Wide Web: <http://www.goldpractices.com/practices/arrc/index.php>
- [13] Software Engineering Institute (SEI) (2009). A framework for software product line practice, version 5.0: What is a software product line? Retrieved April 29, 2012, Wide Web: http://www.sei.cmu.edu/productlines/frame_report/what.is.a.PL.htm.
- [14] Hughes, R.T. (1996). Expert judgment as an estimating method. *Information and Software Technology*, 38(2), 67-75.
- [15] Li, Y., Xie, M., and Goh, T.N. (2008). A bayesian interface approach for probabilistic analogy based software maintenance effort estimation. *14th IEEE Pacific Rim International Symposium on Dependable Computing*, 176-183.
- [16] Shepperd, M., and Schofield, C. (1997). Estimating software project effort using analogy. *IEEE Transactions on Software Engineering*, SE-23:12, 736-743.
- [17] Hari, CH., Reddy, P., Kumar, S., and Ganesh, G. (2009). Identifying the importance of software reuse in COCOMO81, COCOMOIL. *International Journal on Computer Science and Engineering*, 1(3), 142-147.
- [18] Nasir, M. (2006). A survey of software estimation techniques and project planning practices. *Proceedings of the ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing*.

SESSION

MODELING, SOFTWARE DEVELOPMENT, USER INTERFACE METHODS + VISUAL PROGRAMMING

Chair(s)

TBA

A Two-Dimensional Overall Software Customization Classification and Visualization

Michaela Weiss¹, Norbert Heidenbluth

Inst. for Applied Information Processing, Ulm University, 89069 Ulm, Germany

¹ Scholar of the Wilken Foundation, Ulm

Abstract— *Customization is widely used in non-software areas and is a crucial part in current software engineering as an option to adapt software to end-user needs. Since existing customization classifications are very rough, previous studies only evaluate the overall acceptance of software customization but do not evaluate specific adaptations.*

In this paper, we analyze research to date and extend existing customization classifications to get an embracing, two-dimensional classification for non-software products. We also develop a chart to visualize this classification. The findings are carefully applied to the software sector and validated with the help of an empiricism. Thus we create a comprehensive two-dimensional software customization classification and a Software Customization Chart (SCC).

This helps vendors identify shortcomings in customization options and judge the acceptance of their customization features. Hence software producers could gain a competitive advantage and users could get software that meets their needs.

Keywords: Customization, Adaptability, Tailoring, GUI Design, Human-Computer Interaction

1. Introduction

Surveys show that customers increasingly demand products or services that exactly meet their individual needs [1] rather than accepting the *customer sacrifice* [2], [3]. This describes the gap between a customer's needs and the supplied products and services. This is why the *long tail phenomenon* [4], [5] starts to dominate the market. Large traditional markets are now being replaced by millions of small markets with low sales. These tiny niche markets all together offer immense revenues because demand does not shrink. Empiricisms also show that customers value products that match their needs [6], [7]. As a result of this changing market situation, customization is of vital importance for companies and is part of our daily life.

The trend of customization is also visible in the software sector. For instance, apps and web portals grant their users access to the required functionalities which in terms of web-based resources can be combined with the help of *mashups* [8], [9]. The *Technical Report on Software Engineering* (ISO/IEC TR 9126-2) even names the "suitability of the software for individualization" as a product quality aspect because homogeneous standard software can only rarely meet the needs

of an individual customer. Additionally, customization can help increase the *customer experience* [10] and cope with the *experience economy* as Gilmore and Pine called our society as far back as in 1998 [11]. The immaterial product software offers many customization opportunities. In order to exploit the full capabilities, developers need a classification to structure adaptations to improve their work and avoid customer confusion. Moreover, a classification is needed to analyze the various adaptation options. This classification should focus on users and their perception because customers are the basis of the market, and—according to Peter Drucker—a company can only prosper if it focuses on its customers and their needs [12].

We introduce a comprehensive classification of software customization from a customer's point of view to comply with these needs. This high-level groundwork serves as a basis for software implementation and analysis. Furthermore, we present a new chart to illustrate customization options. This helps software companies analyze their software and brave competitive pressure. Customers can be attracted to products addressed to their needs and cross and up-selling could be enhanced. Moreover, these products create higher value for the customers than standard software and increase customer loyalty and *willingness to pay* (WTP).

In terms of software classification development, we carefully apply a new classification that focuses on non-software products to the software sector. Although non-software products and software differ greatly in characteristics and manufacture, there are similarities in customer perception. We believe that findings from the non-software area in terms of perceived customization could be valuable for the software sector. This practice is useful since the non-software sector has a much longer history. Thus non-software products are well-known to a wider audience whereas software knowledge could still be limited.

The following Section 2 lists existing customization classifications for non-software and software products and emphasizes the requirements of a new, more detailed categorization. In Section 3, we extend existing customization classifications to get a two-dimensional customization classification. Furthermore, we introduce a method to visualize customization. These findings are applied to the software sector and presented in Section 4. Section 5 illustrates an empiric validation of the software customization classification. Finally, Section 6 summarizes the results and shows aspects for further research.

2. Background and Related Work

Customization is a topic that is widely studied and discussed in the literature. In the following, we focus on the evolution of customization and previous classifications.

2.1 Evolution of Customization

In the 1990's, new business strategies arose which focused on individual customers [13] rather than standardization. These strategies can be traced back to the production strategy of *mass customization* (MC), that links differentiation with cost leadership. The term MC was introduced by Stanley Davis in 1987 [14] and made popular by Joseph Pine [15] in 1993.

Customization helps meet customer needs by adapting product features to differing customer requirements [16]. In 1999, Åhlström and Westbrook identified increasing customer satisfaction and market share as the greatest benefits of customization. In contrast, longer delivery times and increasing material and manufacturing costs are the biggest disadvantages [1].

This trend and the increasing importance of accessibility stimulated research in customized software development. In order to support older or disabled people many authors focused on the customization of *graphical user interfaces* (GUIs) [17], [18], [19], [20], [21]. Furthermore, much technical work was done to discover the best practices for designing menus [22], [23], GUIs [24], [25], [26], and product lines [27], [28].

In comparison to the amount of technical studies, comparatively less conceptual work has been done. In 1991, however, Mackay analyzed the triggers and barriers to software customization. She named the reusing of repeated patterns, the retrofitting after a system change, and the avoidance of annoying behaviors as the main reasons for software customization. In contrast, barriers are a lack of time and knowledge [29]. In 1996, a study by Page et al. showed similar results [30].

2.2 Classification of Customization

There are different categorizations to specify customization. In 1995, Coates and Wolff established the terms *soft customization* and *hard customization* [31]. They classified customized products according to the person who adapts them. Products adapted by the customer or the retailer are referred to as soft customized products and need no adaptations in production and distribution. In this case, standard goods with adaptation options are produced. In contrast, hard customization is done by the manufacturer and results in higher complexity of production and distribution. However, the adaptation options increase. The different manufacturing processes needed justify this categorization. Nevertheless, this classification is rather rough and does not consider the particular adaptations.

In 1996, Lampel and Mintzberg extended this classification. They specified *pure standardization*, *segmented standardization*, *customized standardization*, *tailored customization*, and *pure customization* [13]. These categories differ in the degree of freedom, thus this subdivision is slightly more detailed. However, a closer examination of implemented adaptations is

also missing. The necessity of a classification that focuses on the starting points of customization is documented by Piller and Müller [6] as well as Ponn et al. [32]. Both empiricisms analyze non-software customization and categorize the adaptations by means of the starting points. Piller and Müller use the categories *style*, *fit and comfort*, as well as *functionality*. Ponn et al. use similar categories, namely product *fit*, *form*, and *function*. An established classification could help create uniformity.

There are only few classifications of software customization. The most popular one is the diversification in *adaptable*, *adaptive* and *mixed-initiative* customization. Adaptable strategies are based on the implementation of adaptation options which can be tailored by the end-user. This gives users the control but causes barriers to customization such as a waste of time as well as problems caused by a lack of knowledge on adapting. In contrast, adaptive methods are based on an intelligent software system which analyzes software usage. Problems arising from a low quality of adaptations as well as a lack of control and transparency could occur. This adaptation, however, does not require any additional effort. Mixed initiatives combine the advantages of both strategies. A number of writers have conducted studies on this adaptation implementation [33], [22], [23], [34], [35], [36]. Furthermore, there are studies on the automatic generation of customer-optimized GUIs [37], [38], [39], [40]. The classification in adaptable, adaptive and mixed initiatives is based on who carries out customization and thus can be regarded as the equivalent to the above named classification of Coates and Wolff. Unfortunately, it only considers technical criteria and does not draw any conclusion about which aspect of the software is really adapted. Thus Oppermann and Simm divided software customization into *adaptations of the functionality* and *adaptations of the interface* in 1994 [41].

Yet, for users the GUI is their connection to the software, and they sometimes even feel it is the software itself. Thus the classification of Oppermann and Simm could not be used to handle perceived customization. It is also very rough and needs to be improved. Thus we introduce a more detailed classification, combining both the execution and the starting points of the customization. This enables detailed analysis of software customization with focus on user experience. This classification also helps compare different software systems in terms of customization.

3. Non-Software Customization

In this section, we present *DEFS Customization Classification*, which is based on starting points. It unifies and extends the categories used by Piller and Müller [6], as well as Ponn et al. [32] (cf. Section 2). We add the *services* category to get an overall classification. In the following, we introduce the four areas of adaptation, *design*, *ergonomics and fitting*, *functionality*, and *services*. Afterwards, we merge this classification with that of Coates and Wolff [31]. Thus we provide a comprehensive two-dimensional classification that is illustrated with the help of our *customization chart* (CC).

3.1 Design (D)

Since the whole product look, e.g. the styles, colors, and patterns can be adapted to customer-specific preferences, the first category to customize a product is design. This is highly promising because of the subjective perception. Hence design customization can easily raise attachment to the product. Furthermore, this facilitates risk reduction because of the *made-to-order-principle* [42] that replaces uncertain predictions, avoids warehouse stock, and leads to a higher WTP.

3.2 Ergonomics and Fitting (E)

The second category for customization is the adaptation of the ergonomics and fitting to allow for the customer-specific body. In many areas, above all in the health sector, these adaptations are absolutely necessary. Nevertheless, in industrial sectors such as footwear, automobile, and sporting goods many companies also offer ways of improving the ergonomics. The huge advantage of ergonomics customization is that even customization critics cannot refute the benefit of such an adaptation because it is objectively noticeable.

3.3 Functionality (F)

Requirements of the customers differ greatly and provide a sound basis for customization. Thus functionality is another pillar of customization. Many products that offer such adaptation options use modularization to fulfill customer needs by merging favored modules. Those build-to-customer programs are often used in the computer hardware sector.

3.4 Services (S)

As a result of increasing competitive pressure many companies now offer additional services. This started—as Rust and Lemon called it in 2001—the *service revolution* [43]. Customers also consider extended benefits as an important part of the solution to their specific problem. Hence customization in services is a great way of achieving a better fitting of product characteristics and customer needs. Such adaptations only affect the last step of the value-adding process. Thus they can be seen as the easiest way of customization and could be assigned to Gilmore and Pine's customization strategy of *cosmetic customization* [44]. However, only voluntary services that afford additional benefits offer customization capabilities.

3.5 Two-Dim. Classification and Visualization

The DEFS classification focuses on the starting points of adaptations but does not consider who really makes them. This distinction, however, was already made by the classification of Coates and Wolffs [31]. Thus we merge both classifications to achieve a new comprehensive two-dimensional classification.

We develop a CC to visualize this classification. It illustrates the DEFS customization categories *design*, *ergonomics and fitting*, *functionality*, and *services* with the help of bars. The bar width indicates the amount of available customization options

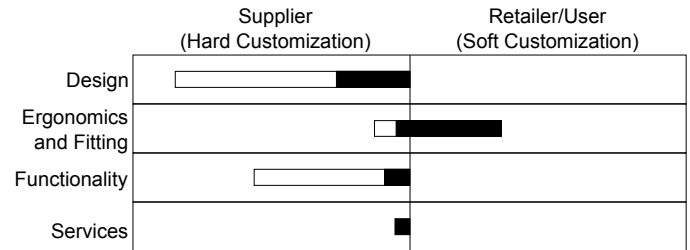


Fig. 1: SLK CC and usage of customer A

in each category. A two-dimensional view is made by adding a distinction between hard and soft customization.

Table 1 lists the customization options of the Daimler SLK product line. They can be visualized with the help of a CC (cf. Figure 1). The SLK especially offers many options for a hard customization. As a consequence, the hard customization column of the CC is highly pronounced. Soft customization is only available in terms of the ergonomics and fitting (adjustment of seats, steering wheel stand, and mirrors). Thus the only bar in the right column is in this category. The CC easily illustrates the customization options and the viewer immediately knows that the SLK offers particularly many design adaptations but also focuses on functionality customization. However, service customization is only rarely available. Daimler offers the updating of navigation maps as well as prepaid maintenance.

DEFS	Customization Options
Design	color of exterior/interior, wheels, seat materials, wheel hub inserts, wheel locks, tire valve stem caps, trim, sunroof, spoiler, sport body styling, trunk handle, chrome fins, licence plate frames,...
Erg/Fitting	steering, keyless go, wipers with rain sensor,...
Functionality	navigation system, media interface, radio, sound system, DVD/CD changer, memory card, lighting, child safety seats, climate control, heated seats, car-theft protection, park distance control system,...
Services	navigation map updates, prepaid maintenance

Table 1: SLK customization options^{1,2} and usage of customer A

This example shows that the two-dimensional classification and its visualization in a CC enables an intuitive overview of customization. In contrast, tables are predestined to list customization options, but quickly become complex and confusing. Even though this example is limited to one product, a CC can be used to compare similar products.

By adding a filling to the bar the usage of these options can be illustrated for a special customer or a customer group. The CC example shows the used customization options of the imaginary customer A that are highlighted in Table 1.

¹Genuine Mercedes-Benz Accessories for the SLK Class, Mercedes-Benz USA, ACC-11-R1721-10000 (04/2011)

²Mercedes-Benz vehicle builder, <http://www.mbusa.com/mercedes/vehicles/build>

4. Software Customization

Software may be adapted in various ways to user preferences, characteristics and tasks. To handle this variety, we introduce the starting-point based *software customization classification DUFs* which is modeled on the DEFS classification (cf. Section 3). To provide the characteristics of software the categories are applied to *design, usability, functionality, and service customization and communication*. Moreover, a comprehensive, two-dimensional software customization classification and the software customization chart are introduced.

4.1 Design (D)

The software customization starting point design can be directly derived from the DEFS classification. The term software design is often used in conjunction with software architecture. This architecture is not visible for the users. Thus perceived software design customization refers to the GUI appearance.

Design adaptations are an easy way of allowing for customer preferences which vary greatly in design. Even color combinations cause different reactions in customers so there are many options for design customization. Adaptations can affect colors, contrasts, as well as font type and size. Individual photographs, icons, and user name and initials could also be added. All these changes are minor but can increase customer acceptability and are quite common in the *business-to-business* (B2B) sector.

It is important to point out that design is related to usability. For example, different color combinations also differ in their suitability for use and customers should be supported in their design decisions. Moreover, design adaptations can help make software accessible for people with disabilities. For instance, enlargement of the font size or icons and use of different colors can deal with the needs of people with impaired motor or vision skills. As most users are not disabled and see these adaptation as design aspects, they belong to the *design* category.

4.2 Usability (U)

Another category of software customization is usability, reflecting the non-software customization category *fittings and ergonomics*. In accordance with DIN EN ISO 9241 Part 11 this is how “effective, efficient, and task satisfying a software can be used by a user to fulfill his specific tasks”. GUIs which are characterized by good usability are easy to handle, well-arranged, and contain self-explanatory icons and menu items. The operating manual, the integrated help, and the compatibility also affect usability. All these aspects help optimize software usage, save time and physical effort.

Problems fulfilling the task can either occur as the needed functionality is not available (see the next paragraph) or cannot be used because of incorrect handling. Since the usage of a specific GUI can be totally intuitive for one user but absolutely incomprehensible for another, customization offers a great benefit. Compliance with the customer-specific intuitiveness means adapting the appearance of the GUI. This can be done by adapting the layout and grouping of the buttons, menu items,

and bars or by offering a choice between different kinds of GUI elements. Furthermore, icon images and labels can be customized. These adaptations affect the appearance of the GUI but are done to improve the usage rather than the design. Thus these adaptations belong to the *usability* category.

Moreover, coping with the customer-specific hardware and the user’s native language as well as giving users the possibility to create shortcuts influences overall usability. As mentioned above, usability can also be used to make software accessible for everyone and comply with DIN EN ISO 9241-171.

4.3 Functionality (F)

Functionality is the heart of the software and can affect usability as mentioned above. However, due to increasing technological possibilities rather than insufficient functionalities, software offers a multitude of features. Thus software is often called *bloated*. Bloated software can be separated into *objective* and *subjective* bloat [45]. Objective bloat identifies functionality that is neither wanted nor needed by any user and should be avoided. In contrast, subjective bloat sums up functionalities that are not needed by the specific user and can be reduced with functionality customization. Many companies try to come up with this functional abundance by offering software variants or plug-ins. The plug-in concept enables customers to add the needed features. A very easy way of offering adaptation in terms of functionality is to hide and show functionalities. Even though undesired functionalities are only hidden but still available, the software seems to be adapted.

4.4 Services and Communication (S)

Software is an immaterial product itself, but several software companies offer auxiliary services to distinguish themselves from the competition and offer customization. Examples of such customizations are individual customer care, an update service, and customer-specific maintenance. The focus on communication prevents the customer from having the feeling of being only one of a million.

Furthermore, characteristics of software allow another kind of customer contact because software itself is able to communicate with the customer. For instance, the software could include the user’s name that is entered through installation in order to create individual greetings. Thus an emotional attachment to software and perceived customization can be promoted.

4.5 Two-Dim. Classification and Visualization

To get an overall view of software customization it is important to match the DUFs starting points with the knowledge of the customization executor. Thus we now introduce a two-dimensional, overall classification. In contrast to Chapter 3, we use the categorization of adaptive, adaptable and mixed initiatives to distinguish between the executors.

We develop a software customization chart to visualize software customization. The bars show how many customization options the software offers, subdivided into the DUFs

DUFS	Windows 7 Calculator	DeskCalc 4.2.12	SFR CalcTape Pro 5.0.0	fnA 1.31a
Design	-	views, schemes, number block visibility	views, font colors (value-constrained), document colors, visibility of ribbons	input font color
Usability	number format, history, ex post value changes	number format, ex post value changes, language, shortcuts, round method, sound, exit handling, constants; user-confirmed autostart, laptop mode	round method, language, ex post value changes, shifting, variables, font size, user buttons, shortcuts, toolbar; user-confirmed autostart	startup mode or user-confirmed saving of the last one, user-confirmed autostart
Functionality	basic/scientific/statistic functionalities can be chosen, date calculator/economic calculator/ unit converter can be faded in	commercial/formel functionalities can be chosen	-	-
Services/Com.	-	user-confirmed software and currency rate update service	user-confirmed update service	-

Table 2: Customization options of calculator subset

categories. By using two columns the distinction between the customization executor is made. The *system/supplier* column marks adaptive customization. The *user* column shows the adaptable customization. An SCC can also visualize mixed initiatives. On the one hand, a mixed initiative may exist because the software offers adaptive and adaptable customization options. This is shown by bars in both columns. On the other hand, a mixed initiative can be based on the requirement of an immediate user communication to execute an adaptive change. This could be an adaptive customization which is triggered by a customer call or requires a confirmation. These communication-based mixed initiatives are marked by a bar in the adaptive column that starts with a circle. The circle is filled if every adaptive customization needs a direct user interaction, or not filled if it is only needed for several adaptations.

In the following, we compare the customization options of several desktop calculators (cf. Table 2) by using the SCC shown in Figure 2. We choose this example because it is a minimal working example that is small, simple, and well-known. Nevertheless, it contains all adaptation options and easily explains the SCC. The subset contains the Windows 7 Calculator, DeskCalc 4.2.12, SFR CalcTape Pro 5.0.0 (SFR Software GmbH) and fnA 1.31a (RJ Software).

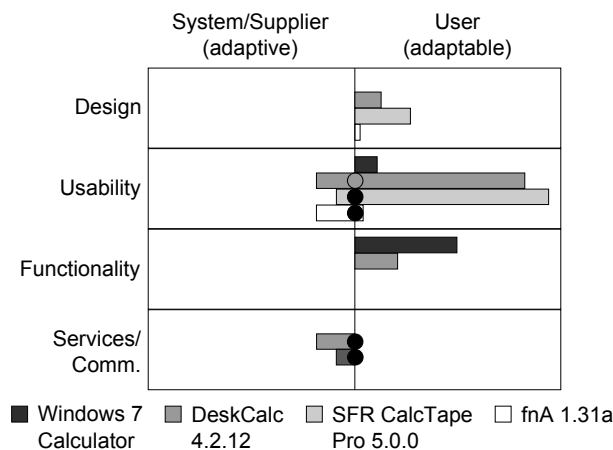


Fig. 2: SCC for calculator subset

The SCC points out that the Windows 7 Calculator provides only customization options that affect usability and functionality. The bar width shows that this calculator offers only little usability customization, but is the most customizable one with regard to functionality aspects. All adaptations offered by the Windows 7 Calculator have to be done by the user, so an adaptable initiative is used. The SCC figures out that DeskCalc and CalcTape provide above all usability adaptations that also have to be executed by the user. However, CalcTape and fnA contain adaptive features in terms of usability and with service customization and communication. All these adaptive changes have to be confirmed by the user. This is symbolized by the filled circle and highlights that a mixed initiative based on user interaction is implemented.

DeskCalc automatically adapts to laptop characteristics in terms of usability but offers a user-confirmed autostart. Thus a mixed-initiative is also used. Since not every adaptive change needs user interaction, the circle in the SCC is empty.

The example illustrates that the two-dimensional classification can help gain valuable insights into software customization and helps visually compare different software. In contrast to large, detailed, and complex tables this allows an intuitive overview of software customization. Additionally, a filling could be used to show customization usage. The width of a bar shows the amount of all available customization options, whereas the filling illustrates how many of them are really used by a specific customer or customer group.

5. Empiricism

Our two-dimensional customization classification is based on two customization aspects. On the one hand, we use the well-known and popular distinction between adaptable, adaptive, and mixed initiatives. On the other hand, we develop a new DUFS classification to handle the customization starting points. Since the DUFS classification is not yet evaluated, we conducted a large study to get insights into customer opinions on customization and validate the DUFS classification. In this section the used methods are outlined and we present the part of the survey that deals with the DUFS categories.

5.1 Methods

The cross-sectional study was conducted in 2010 in South Germany with 284 participants. The answers of ten interviewees could not be used because of missing data. Thus the study includes the answers of 274 participants. 43.43% of them are female and 56.57% male. The survey could be completed either electronically (20.44%) or in paper form (79.65%). Figure 3 shows the age distribution of the participants.

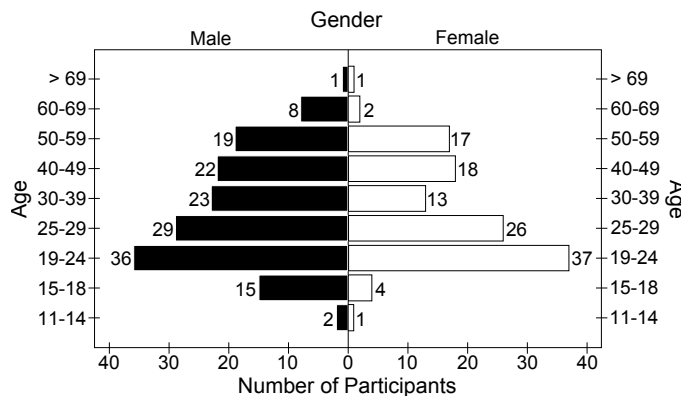


Fig. 3: Age Distribution

The proximity to Ulm University leads to several differences in the sample in comparison to the whole German population. It includes comparatively many young, well-educated participants and the opinions of male participants are slightly overweighted. However, the large and heterogenous sample allows conclusions to be drawn on the perceived customization.

5.2 Validation of DUFs

In Section 1, we explained that a classification is needed that focuses on the customer's point of view. The most important theory to explain human behaviour is the concept of the *homo oeconomicus*. It explains the behaviour of human beings with the help of the benefit involved. Thus we investigated whether there are differences in the benefit between several existing and well-known software customization options. Such differences could be used to find clusters and validate our classification.

The participants had to judge 15 customization features. They are mainly located in the areas of *operating systems*, *Office products* and *world wide web* because participants of all age categories are familiar with them. In terms of usability there are many adaptation options available. We chose the *creation of links* and *bookmarks* that help quick access to customer-specific data or websites. The *quick launch bar* of the operating system as well as the *tool bars* in Office programs also enable quick access and improve usability. Moreover, we listed the feature to decide on different options to handle *updates* and to choose one's *native language*. With regard to design adaptations, we specified the options of adapting *fonts*, *colors and contrast*, *icon size*, *desktop background*, and *mouse pointer*. Participants also rated the possibility to customizing the *screen saver*.

Furthermore, the participants judged functional customization offered by *iGoogle* and *Windows gadgets*. In the B2C sector, service customization is only rarely available. Thus we limited the survey to adaptive purchase proposals in terms of shopping in online shops.

The survey gave a brief description of each customization option to ensure that the participants understood the questions and conclusions could be drawn. Afterwards, the participants had to evaluate each customization option with a benefit value between 0 and 5 on a Likert scale. A benefit of 0 means that this customization feature is completely useless. In contrast, a benefit of 5 shows that this option is very useful.

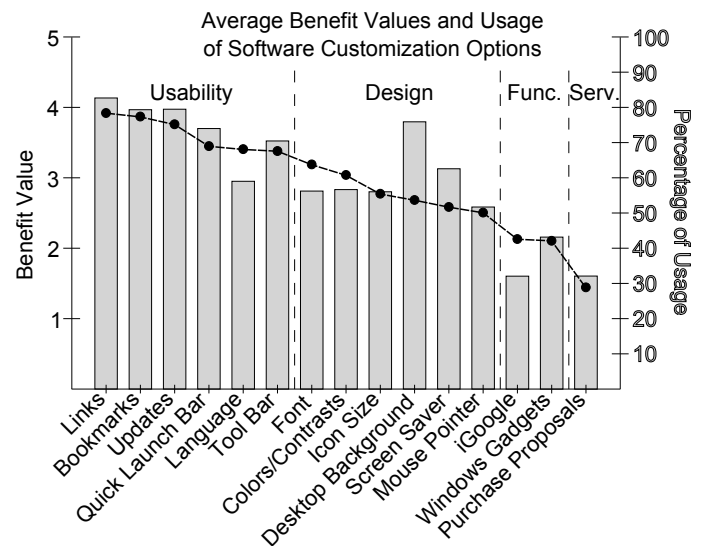


Fig. 4: Benefit and Usage of Software Customization Options

Figure 4 illustrates the results and demonstrates the mapping of the named customization options to the *DUFs* software customization categorization. With regard to the benefit values, clusters became visible that comply with the *DUFs* categories. All customization options that affect usability are rated similar (3.38 to 3.92). Such common benefit values can also be seen in terms of design (2.50 to 3.19) and functionality adaptations (2.09 to 2.12). Even the services and communication category is different in terms of benefit values (1.44) to the other customization options. Consequently, the *DUFs* classification reflects the benefit values mentioned by the participants. Since the benefit is the center of customer behaviour the *DUFs* classification is a significant method for classification.

We also analyzed the connection between benefit values and usage. Thus the chart also figures out participant usage of the customization options. To get meaningful conclusions, we excluded the participants that did not know that the specific adaptation option exists. A correlation between benefit values and usage became visible. Hence the *DUFs* classification is a valuable categorization of software customization and could help vendors create software that fulfills user requirements in terms of customization.

6. Conclusions and Future Work

This paper showed that there are many ways of implementing customization and gave a detailed overview of previous research. Existing classifications in the non-software and software sector are very rough and do not focus on the customer's view. Thus we presented the new starting-point based *DEFS* and *DUFS* categorizations to classify and structure perceived customization in both the non-software and the software sector. The *DUFS* categorization is a conceptual basis for software customization and enables customer-oriented analysis. It helps software developers to decide about adaptations that meet customer needs.

Each customization classification was consolidated with an existing categorization to get a comprehensive two-dimensional classification. These new classifications contain both the starting points of the customization and the customization executor. Two new kinds of charts, the *Customization Chart (CC)* and the *Software Customization Chart (SCC)*, were also implemented. These illustrate the customization options of a product and enable comparisons of similar products. The charts also help compare the available customization options with the customization usage of a customer or a group of customers. Thus software developers could easily discover shortcomings in customization and recognize if the implemented customization options are accepted by the customers. This would help gain a competitive advantage.

The analysis of existing software with focus on customization options and the illustration and comparison in *SCCs* are interesting for future research.

References

- [1] P. Åhlström and R. Westbrook, "Implications of mass customization for operations management," *Int. J. of Operations and Production Management*, vol. 19, no. 3, 1999.
- [2] C. W. Hart, "Mass customization: conceptual underpinnings, opportunities and limits," *Int. J. of Service Industry Management.*, vol. 6, no. 2, pp. 36–45, 1995.
- [3] A. Bardakci and J. W. AND, "How "ready" are customers for mass customisation? an exploratory investigation," *European J. of Marketing*, vol. 38, no. 11/12, pp. 1396 – 1416, 2004.
- [4] C. Anderson, "The long tail," *Wired Mag.*, vol. 12, pp. 170–177, 2004.
- [5] C. Anderson, *The Long Tail: How Endless Choice is Creating Unlimited Demand*. Random House, 2010.
- [6] F. T. Piller and M. Müller, "A new marketing approach to mass customisation," *Int. J. of Computer Integrated Manufacturing*, vol. 17, no. 7, pp. 583–593, 2004.
- [7] F. T. Piller, K. Moeslein, C. M. Stotko, and C. M. Stotko, "Does mass customization pay? an economic approach to evaluate customer integration," *Control*, vol. 15, no. 4, pp. 435–444, 2004.
- [8] C. Schroth and O. Christ, "Brave new web: Emerging design principles and technologies as enablers of a global soa," *SCC*, pp. 597–604, 2007.
- [9] V. Hoyer, K. Stanoesvka-Slabeva, T. Janner, and C. Schroth, "Enterprise mashups: Design principles towards the long tail of user needs," *SCC*, vol. 2, pp. 601–602, 2008.
- [10] S. Marathe and S. S. Sundar, "What drives customization? control or identity?" in *CHI '11*, 2011, pp. 781–790.
- [11] B. J. Pine II and J. H. Gilmore, "Welcome to the experience economy," *Harvard Business Review*, vol. 76, no. 4, pp. 97–105, 1998.
- [12] P. F. Drucker, *The Practice of Management*. HarperBusiness, 2006.
- [13] J. Lampel and H. Mintzberg, "Customizing customization," *Sloan Management Review*, vol. 38, no. 1, pp. 21–30, 1996.
- [14] S. M. Davis, *Future Perfect*. Addison-Wesley, 1987.
- [15] B. J. Pine II, "Mass customization: The new frontier in business competition," Harvard University Press, 1993.
- [16] M. Spring and J. Dalrymple, "Product customization and manufacturing strategy," *Int. J. of Operations & Production Management*, vol. 20, pp. 441–467, 2000.
- [17] A. Dickinson, R. Eisma, and P. Gregor, "The barriers that older novices encounter to computer use," *UA in the Inform. Society*, pp. 1–6, 2010.
- [18] P. Gregor and A. F. Newell, "Designing for dynamic diversity: making accessible interfaces for older people," in *WUAUC'01*, 2001, pp. 90–92.
- [19] A. Newell and P. Gregor, "Design for older and disabled people - where do we go from here?" *UA in the Inform. Society*, vol. 2, pp. 3–7, 2002.
- [20] A. Newell, "Accessible computing – past trends and future suggestions: Com. on comp. and people with disabilities," *TACCESS*, vol. 1, 2008.
- [21] G. Pullin and A. Newell, "Focussing on extra-ordinary users," in *UAHCI'07*, 2007, pp. 253–262.
- [22] L. Findlater and K. Z. Gajos, "Design space and evaluation challenges of adaptive graphical user interfaces," *AI Magazine*, vol. 30, no. 3, pp. 68–73, 2009.
- [23] L. Findlater and J. McGrenere, "A comparison of static, adaptive, and adaptable menus," in *SIGCHI '04*, 2004, pp. 89–96.
- [24] A. Bunt, C. Conati, and J. McGrenere, "What role can adaptive support play in an adaptable system?" in *IUI '04*, 2004, pp. 117–124.
- [25] A. Bunt, C. Conate, and J. McGrenere, "Supporting interface customization using a mixed-initiative approach," in *IUI '07*, 2007, pp. 92–101.
- [26] J. McGrenere, R. M. Baecker, and K. S. Booth, "An evaluation of a multiple interface design solution for bloated software," in *SIGCHI '02*, 2002, pp. 164–170.
- [27] K. Pohl, G. Böckle, and F. van der Linden, *Software product line engineering*. Birkhäuser, 2005.
- [28] J. Bosch, "Maturity and evolution in software product lines: Approaches, artefacts and organization," in *SPLC'02*, 2002, pp. 257–271.
- [29] W. E. Mackay, "Triggers and barriers to customizing software," in *SIGCHI '91*, 1991, pp. 153–160.
- [30] S. R. Page, T. J. Johnsgard, U. Albert, and C. D. Allen, "User customization of a word processor," in *SIGCHI '96*, 1996, pp. 340–346.
- [31] J. F. Coates and M. F. Wolff, "Customization promises sharp competitive edge," *Research Technology Management*, vol. 38, no. 6, pp. 6–7, 1995.
- [32] J. Ponn, C. Baumberger, and U. Lindemann, "Guidelines for the development of individualized products," in *DESIGN'04*, 2004.
- [33] S. Greenberg and I. H. Witten, "Adaptive personalized interfaces: A question of viability," *Behaviour and Information Technology*, vol. 4, no. 1, pp. 31–45, 1985.
- [34] K. Z. Gajos, M. Czerwinski, D. S. Tan, and D. S. Weld, "Exploring the design space for adaptive graphical user interfaces," in *AVI'06*, 2006, pp. 201–208.
- [35] E. Horvitz, "Principles of mixed-initiative user interfaces," in *SIGCHI '99*, 1999, pp. 159–166.
- [36] C. G. Thomas and M. Krogsæter, "An adaptive environment for the user interface of excel," in *IUI '93*. AC, 1993, pp. 123–130.
- [37] K. Z. Gajos, R. Hoffmann, and D. S. Weld, "Improving user interface personalization," in *UIST '04*, 2004.
- [38] K. Z. Gajos, J. J. Long, and D. S. Weld, "Automatically generating custom user interfaces for users with physical disabilities," in *Assets '06*, 2006, pp. 243–244.
- [39] K. Z. Gajos, J. O. Wobbrock, and D. S. Weld, "Automatically generating user interfaces adapted to users' motor and vision capabilities," in *UIST '07*, 2007, pp. 231–240.
- [40] K. Z. Gajos, J. O. Wobbrock, and D. S. Weld, "Improving the performance of motor-impaired users with automatically-generated, ability-based interfaces," in *SIGCHI '08*, 2008, pp. 1257–1266.
- [41] R. Oppermann, Ed., *Adaptive User Support : Ergonomic Design of Manually and Automatically Adaptable Software*. Erlbaum Assoc., 1994.
- [42] C. Berger and F. Piller, "Customers as co-designers," *Manufacturing Engineer*, pp. 42–45, Aug./Sep. 2003.
- [43] R. T. Rust and K. N. Lemon, "E-service and the consumer," *Int. Journal of Electronic Commerce*, vol. 5, no. 3, pp. 85–101, 2001.
- [44] J. H. Gilmore and B. J. P. II, "The four faces of mass customization," *Harvard Business Review*, vol. 75, no. 1, pp. 91–101, Jan.-Feb. 1997.
- [45] J. McGrenere and G. Moore, "Are we all in the same "bloat"?" in *Graphics Interface*, 2000.

A Graphical Approach to the Development of Deployment Agnostic Systems

Dr. Mark.B.Dixon

School of Computing and Creative Technologies, Leeds Metropolitan University, Leeds, England

Abstract - *The ever expanding number of environments in which computer systems are being used has led to the evolution of numerous development languages, tools and techniques. This paper discusses a predominantly graphical approach to software development that is deployment platform agnostic. The aim is to provide engineers with an approach to development that is general enough to be applied across the multitude of problem domains. By using a purely component based approach, in which target platform specifics are hidden from the language design, it has become possible to build a set of interrelated tools which allow for the development, manipulation and exchange of implementation solutions.*

Keywords: modeling; graphical; component; deployment

1. Introduction

One of the main difficulties faced by software engineers is the sheer array of available development languages, run-time platforms and deployment architectures. The decision of which combination of available tools and techniques to be used is often dictated by the nature of the target system. Broadly speaking target systems can be classified as being desktop applications, mobile applications, embedded control systems, web applications, and distributed applications including Service Oriented Architectures (SOAs). Many of these systems are multi-tier of course, meaning that many modern solutions are actually a hybrid of the aforementioned categories.

The presence of multiple deployment possibilities has led to fragmented development approaches, not just at a language level but also at a tools level. For example, embedded systems development is vastly different to SOA development in terms of implementation languages, development approaches, tracing, debugging etc. The work described within this paper aims to provide a single development technique underpinned by a set of tools capable of supporting different target platforms.

The suggested technique and supporting tools, which has been named the Razor Development Environment (RDE), consists of a component oriented graphical notation supported by an underlying 3GL type language. The enforced use of a component based structure and strong

support for re-use among components allows the majority of a solution to be developed using existing generic components which do not include any target specific information. A small number of platform specific components can then be linked with the main solution to produce a deployable system.

2. Background

The graphical notation of the RDE was initially devised as part of a project to develop an embedded operating system [1]. This initial work was extended upon in an effort to allow the technique to be applied to a wider variety of problem domains, including those outside the embedded systems arena. During this work much consideration was given to the best mechanism of supporting deployment of RDE based systems. The possibilities considered included run-time interpretation; native compilation; and language translation. The former of which has been implemented in a reference implementation.

The fact that there were so many deployment possibilities led to initial confusion. It became clear however that this level of flexibility was actually a strength of the approach since it provided the basis of a platform neutral technique. By removing certain constructs from the core RDE, such as direct support for dynamic component instantiation and multiple threading, a fully deployment agnostic approach was developed.

3. The Razor Development Environment

3.1 Overview and Architecture

The RDE provides a selection of tools and techniques that allow the development of software systems using a reusable component based approach. At the core of the environment sits a Document Object Model (DOM) that provides a canonical representation of the application under development. This DOM defines the available components along with how they are configured and connected via their interfaces.

The DOM may be populated using a graphical notation, a textual 3GL type language or an XML document. These three representations are 100% semantically equivalent, hence DOM contents can be manipulated using any representation, independent of the original input method. Any number of deployment tools can then be developed to

produce systems via interrogation of the DOM. A graphical representation of the environment [2] is shown in Fig. 1.

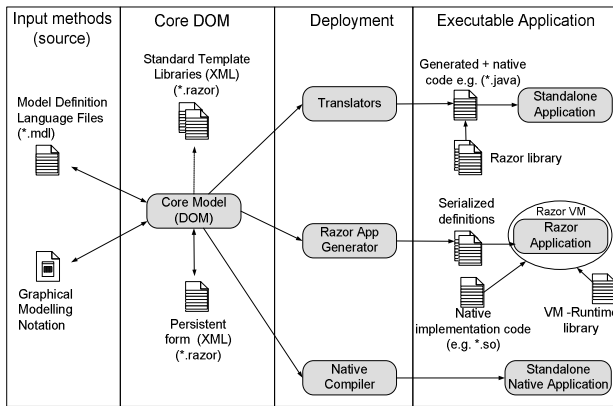


Figure 1. The Razor Development Environment

A developer can work on components with only limited regard for the architecture on which they are to be deployed, hence most components are generic re-usable objects. Only target specific components need to take into account deployment and domain specifics. Due to loose coupling and strong reusability support, both generic and platform specific components can be easily sewn together to provide a complete solution.

The RDE has implicit support for automated testing. The ability to specify compliance tests when defining component interfaces ensures better support for mature concepts such as the Programming by Contract [3] approach along with contemporary development practices such as Test Driven Development (TDD) [4]. Implicit testing support also allows for better independent development of components, since associating well defined tests with specific interfaces in effect provides a mechanism of ensuring semantic compliance.

3.2 Design Principles

The RDE was designed by taking into account many commonly agreed upon design principles, mainly derived from the Object-Oriented paradigm. Many of these concepts enhance the capability for component re-use, which is fundamental to the RDE approach.

The RDE is extremely 'interface centric' in nature. In fact, only interface definitions may represent a type within the system (languages such as Java, C++ and Objective C allow implementation classes to be used as the type of variables and parameters). This design principle is very important for re-use since only interfaces are ever passed as values between component services, also the inheritance model is simplified since implementation inheritance is no longer required or even possible. The demotion of the traditional 'Class' may seem radical but it allows for much better support for the Open-Closed principle [5] which states that elements should be open for extension but closed to modification. It also helps address the well known fragile

base class problem [6] since inheritance hierarchies are interface based, rather than implementation based. Finally, the Liskov Substitution Principle [7] is well supported, not only due to the interface centric nature but due to the ability to ensure semantic compliance of interfaces via implicit testing support.

Systems are defined by identifying component instances and binding them together via their external ports. Each port represents a single interface which defines a number of services, attributes or signals. Each component instance acts as an implementer of one or more port interfaces, either directly through terminal ports or by delegation to sub-components. This multi-interface ability supports the Interface Segregation Principle [8] which promotes the idea of providing many fine grain interfaces in order to help reduce dependencies.

The rules that determine the legality of port bindings between components are based on a signature which does not include the name of the port, only the type information. This loosens the coupling somewhat between components, again promoting re-use. This in-turn also enhances support for the Dependency Inversion Principle [9] that suggests higher level components should not be dependent on lower level components. Within the RDE lower level components can be connected via ports rather than being embedded within the higher level components. The dependency injection pattern [10] is also easily supported by the binding together of higher and lower level components. The weakened port binding semantics also mean that components can be developed more independently, since their visible namespace is only defined within the component itself. Hence, the implementation never needs to be aware of port names that exist outside of the immediate component. Well known classical design patterns [11] are much easier to support in an interface centric component based approach which exhibits low coupling.

3.3 The Graphical Notation

One of the primary aims of the RDE was to support a predominantly graphical model based approach to development. Abstracting to a graphical representation not only simplifies development but better supports the ability to hide deployment specifics. It has been pointed out that model driven approaches are better placed to deal with the complexities of modern platforms, while also allowing better representation of problem domain concepts [12].

A graphical notation has been defined to allow the declarative definition of RDE based software systems. This notation, known as (Razor's) EDGE, allows for the construction of an entire system via the use of a single diagram type. Rather than base the notation on an existing notation, for example by defining a UML profile [13], the decision was made to create the simplest notation possible; while still supporting all required concepts of the underlying DOM.

Unsurprisingly the key elements represented within the notation are Interface and Component definitions. Each of these elements is capable of hosting one or more Ports. Both

Interfaces and Components are represented using the same graphical shape, which may seem odd at first but allows for the use of a single model to define all parts of a system. An interface defines a number of 'provided' and 'required' ports, which in essence determines the direction of dependency when a port based on the interface is bound. The example presented in Fig.2 shows the definition of a 'Flow' interface that allows controlled flow of data. A single service is 'provided' and two signal ports are 'required' in this particular example.

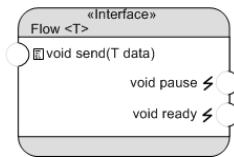


Figure 2. An interface defined using the graphical notation

A component realizes the implementation of a number of interface types. This is accomplished by either providing a terminal port; or delegating to a sub-component instance. Components can also 'provide' and 'require' a number of ports, which are independent of any ports that are defined on implemented interfaces.

An example of a Component is shown in Fig. 3. In this example several contained part instances both provide and require the previously defined 'Flow' interface.

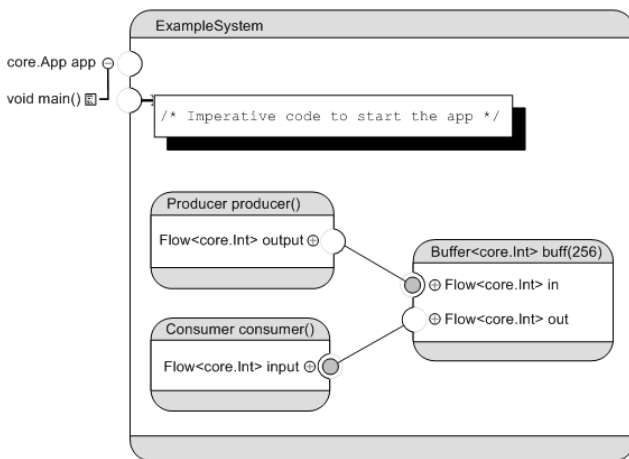


Figure 3. A component defined using the graphical notation

As well as being able to represent the internals of a component using a structural model, components can also be defined as finite state machines. This allows for the definition of control processes such as those used in real-time embedded systems or SOA orchestration languages such as the Web-Services Business Process Execution Language (WS-BPEL) [14]. Interestingly enough almost exactly the same notation can be used to show either

definition style, with only the addition of transitional flows being required to support the state machine approach.

All ports have a *nature* that determines whether they provide a service, store an attribute, represent a signal or are compound, i.e. are nested ports based on other interfaces. Support for compound ports is a very important abstraction mechanism and allows interfaces which are commonly used together to be wrapped into a single conduit type port.

Configuration values and configuration types are used to support customization of components during instantiation. These are analogous to constructor parameters and generic types respectively.

Terminal service implementation code is defined using a simple 'C' like grammar. Within the initial reference implementation Javascript was used but has since been replaced as many concepts supported by Javascript are simply not required within the RDE. The language is only required to consist of standard statements and expressions. Support for creating new objects for example is handled using components, since this is a deployment specific aspect and is not always supported by certain domains such as resource constrained embedded systems.

3.4 The Model Definition Language

A Model Definition Language (MDL) was developed as an alternative mechanism for populating the RDE DOM. Although regular use of the EDGE graphical notation is the final aim, during the research phase the ability to quickly change the grammar rules within a parser make a 3GL textual language more appealing at this time.

The MDL provides the same constructs as the graphical notation as keywords within the legal grammar. The language supports definitions of interfaces, components (using the 'implementation' keyword) and binding of ports. The service implementation code uses exactly the same grammar as it does within the EDGE graphical notation; hence MDL is a superset of the imperative code used in the EDGE model. Example MDL code that is equivalent to the previous example is shown in Fig. 4.

```

implementation ExampleSystem {
    provides {
        core.App app;
    }

    parts {
        Producer producer();
        Consumer consumer();
        Buffer<core.Int> buff(256);
    }

    bindings {
        producer.output -> buff.in;
        buff.out -> consumer.input;
        app.main { /* Imperative code to start the app */ }
    }
}
    
```

Figure 4. A component defined using MDL based code

4. Related Work

The existence of component based techniques is well established. Technologies such as DCOM [15], Corba [16] and JavaEE [17] tend to focus on higher level distributed components however, and often act as a wrapper for existing languages. Hence, they represent a different layer in the software stack than the RDE which constructs systems from components but does not necessarily deploy them as such. OSGi [18] has a little more in common with the RDE since it is not specifically designed to create distributed systems. However it has not been designed to be deployment agnostic and requires a very specific run-time environment.

The work most closely related to the RDE appears to be work carried out by France Telecom within the Fractal Project [19]. This too is a component based approach in which models can be graphically defined. An Architectural Description Language [20] is supplemented with C++ or Java in order to build a full system. Although there are similarities they are also differences regarding environment semantics and the graphical notation. As with the more established technologies, Fractal is primarily based on wrapping existing programming languages. Also the bindings between components within Fractal can be representative of higher level network connections etc.

The RDE is in effect an instance of the Model Driven Engineering (MDE) approach. Hence, from a modeling point of view the UML and related MDA technologies [21] cannot be ignored. The model driven paradigm as defined by the Object Management Group (OMG) aims at providing platform independent models which are mapped to platform specific models using transformation rules. This concept however is fairly complex and requires the definition of multiple mapping and translation rules for each deployment target. Introducing more complexity into the development process is the opposite of what the RDE approach is trying to accomplish.

In terms of purely graphical development of software systems, the MIT App Inventor software [22] and its underlying technologies seem to be based on similar concepts to the RDE. However, the graphical aspects of the RDE are provided to support an architectural level of design, whereas the App Inventor supports graphical design of the user interface along with the procedural aspects of the system. Hence the RDE EDGE notation provides a higher level of abstraction with the lower level imperative code being defined using a 3GL type language (MDL).

The fundamental difference between the RDE and the related work is that its primary aim is to provide a single format that allows for the definition, exchange, and deployment of components which when combined can be used to create software solutions for a diverse set of application domains. In many respects it is synonymous to the philosophy that drove the development of the XML, but instead of being data centric, it is behavioral centric in nature. The tools and techniques developed within the RDE are all specifically designed to provide a realization of this core concept.

5. Evaluation

An initial Java based implementation of the RDE DOM and MDL parser has shown the concept to be a viable approach. These tools are likely to form the basis of an initial release of the environment and have been developed with a commercial strength product in mind. In contrast however, the current run-time environment in which the systems are deployed is unlikely to be usable for real systems. This is because it is a simple interpreter which is capable of executing DOM defined models. Although this serves as a good reference implementation for testing purposes it lacks good performance and does not currently apply any optimization prior to execution.

Although a full implementation of the EDGE notation is not yet complete a prototype has been developed as an Eclipse based GEF [23] dependent plug-in. A screenshot of the graphical editor is shown in Fig.5. At the moment this is not yet functional enough to support development of test systems, thus all current evaluation has been done by developing systems using the purely textual based language (MDL).

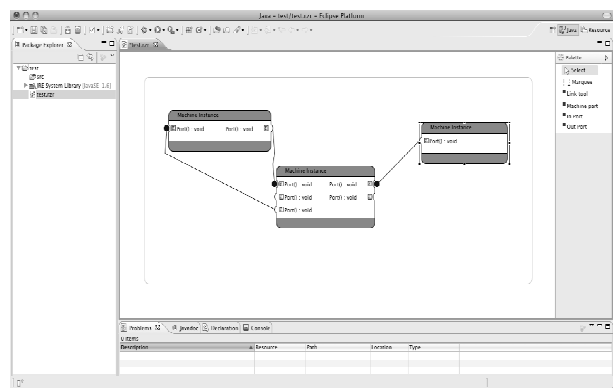


Figure 5. A screenshot of the prototype EDGE tool

The development of the test systems has gone some way to validating the supported constructs along with the mechanics of the approach. This work has also shown however that development via the textual based language alone is a fairly difficult process when compared to traditional OO based programming in languages such as Java. This appears to be due to the fact that the RDE was always designed to be predominantly graphical in nature; hence using a textual language to define the architectural properties of a system is often likely to be counterintuitive.

The development of the run-time environment has also highlighted difficulties when providing an implementation mechanism for compound natured ports. There is a large amount of complexity involved in ensuring that bindings between nested ports are configured correctly during deployment. The ability to pre-examine models and produce optimized compiled code is likely to reduce this problem in the future when compilation or language translation becomes the preferred mechanism of deployment.

6. Conclusions and Future Work

Once a full set of development tools are available the system will need to be more thoroughly evaluated via the production of some industrial strength solutions. The significant point is that the available set of tools will be applicable to the development of all Razor based systems irrespective of the target platform. Only the deployment specific compilers or run-time environments need to be target aware.

A component model such as this could only be a practical reality if mechanisms for the locating and matching of components were provided. Hence publishing and discovery services, such as the Web Services Description Language (WSDL) [24] and Universal Description Discovery and Integration (UDDI) [25], need to be adapted to work within the scope of the RDE.

There needs to be more work undertaken on creating both deployment agnostic and deployment specific components. Once a library of components is available the tool can be released to a wider audience in order to gather feedback. The creation of an open source tool chain, along the same lines as the GNU GCC project [26], would help maximize availability of the proposed approach and provide a low cost of entry for prospective developers. The compilation of Razor compliant components directly into native machine code is the next major aim of this work.

References

- [1] K.Tindell and M.B.Dixon, 'Scalios', A scalable Real-Time Operating System for resource-constrained embedded systems, computer software. Published by JK Energy Ltd. 2008. Available from: <https://github.com/jkenergy/scalios/>
- [2] M.B.Dixon, "Supporting component oriented development with reusable autonomous classes," ARPJ Journal of Systems and Software, vol.1, no.5, August 2011, pp. 182-193, ISSN 2222-9833.
- [3] B. Meyer, Applying "Design by contract," Computer (IEEE), vol. 25, issue 10, October, 1992, pp. 40-51, doi:10.1109/2.161279.
- [4] K. Beck, Test-Driven Development by Example. (The Addison-Wesley Signature Series), Addison Wesley, 2002.
- [5] B. Meyer, Object Oriented Software Construction. Prentice Hall, p 23, 1988.
- [6] L. Mikhajlov and E. Sekerinski, "A study of the fragile base class problem," Proc. ECOOP'98 - 12th European Conference on Object-Oriented Programming, Brussels, Belgium, 1998, pp 355-382.
- [7] B. Liskov and J. Wing, "A behavioral notion of subtyping," ACM Transactions on Programming Languages and Systems (TOPLAS), vol 16, issue 6, November, 1994, pp. 1811 - 1841.
- [8] R. Martin, The Interface Segregation Principle, C++ Report, August, www.objectmentor.com/resources/articles/isp.pdf, 1996.
- [9] R. Martin, The Dependency Inversion Principle, C++ Report, May, www.objectmentor.com/resources/articles/dip.pdf, 1996.
- [10] M. Fowler, Inversion of Control Containers and the Dependency Injection Pattern, <http://martinfowler.com/articles/injection.html>, Jan 2004.
- [11] E. Gamma, R. Helm, R. Johnson and J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley, 1995.
- [12] D. C. Schmidt, "Model Driven Engineering," Computer (IEEE), vol. 39, issue 2, February, 2006, pp. 25-31.
- [13] J. Rumbaugh, I. Jacobson and G. Booch, The Unified Modeling Language Reference Manual, 2nd Edition. Object Technology Series, Addison Wesley, Chapter 12, 2004.
- [14] OASIS, Web Services Business Process Execution Language Version 2.0 standard. <http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html>, April 2007.
- [15] T. L. Thai, Learning DCOM, O'Reilly Media, 1999.
- [16] Object Management Group, Common Object Request Broker Architecture (CORBA), <http://www.omg.org/spec/CORBA>, 1997.
- [17] J.Farley and W.Crawford, Java Enterprise in a Nutshell, O'Reilly Media, 3rd Edition, 2005.
- [18] OSGi Alliance, "OSGi Service Platform Release 4," OSGi Alliance Specifications, <http://www.osgi.org/Specifications/HomePage>, 2009.
- [19] OW2 Consortium, The Fractal Project, <http://fractal.ow2.org>, 2009.
- [20] M. Leclercq, A.-E. Ozcan, V. Quéma and J.-B. Stefani, "Supporting heterogeneous architecture descriptions in an extensible toolset," Proc. 29th International Conference on Software Engineering (ICSE 2007), Minneapolis, MN, USA, May 2007.
- [21] A.Kleppe, J.Warner and W.Bast, MDA Explained: The Model Driven Architecture: Practice and Promise. Object Technology Series, Addison Wesley, 2003.
- [22] D.Wolber, H.Abelson, E.Spertus and L.Looney, App Inventor, May 2011. O'Reilly.
- [23] The Eclipse Foundation, The Graphical Editing Framework (GEF), <http://www.eclipse.org/gef>, 2010.
- [24] W3C, Web Services Description Language (WSDL) Version 2.0 Part 1 : Core Language. <http://www.w3.org/TR/wsdl20>, 2007.
- [25] OASIS, UDDI Specification 3.0.2. http://uddi.org/pubs/uddi_v3.htm, 2004.
- [26] Free Software Foundation, GCC, the GNU Compiler Collection, <http://gcc.gnu.org/>, 2011.

Non-Simultaneous Round-Trip Engineering for 3D Applications

Matthias Lenk¹, Arnd Vitzthum², Bernhard Jung¹

¹Virtual Reality and Multimedia Group, TU Bergakademie Freiberg, Freiberg, Germany

²University of Cooperative Education, Dresden, Germany

Abstract—*This contribution presents a novel extension of round-trip engineering (RTE) to the development of interactive 3D applications. RTE is a model-driven software development process which combines forward (model-to-code) with reverse (code-to-model) engineering in an iterative fashion. Today, several tools exist that support the simultaneous editing of UML diagrams and program code. However, development of 3D application involves different developer groups, i. e. 3D content designers and programmers, who make use of different tools, and produce different types of program code, i. e. the graphical 3D objects and the application logic. Thus, a non-simultaneous means of synchronizing the different code bases with the common model is required. To address the challenges of non-simultaneous RTE for 3D development, we propose a multi-tiered approach comprised of a common domain model written in a DSL for 3D applications, an intermediate model, abstract syntax trees, and program code in the respective target languages. A persistent intermediate model serves as central data structure for the non-simultaneous synchronization of the various models and code artifacts.*

Keywords: *Round-trip engineering, model transformation, code generation, X3DOM*

1. Introduction

Today, 3D computer graphics are indispensable in many domains, such as CAD and architectural planning, information visualization, training simulations, medical and educational applications as well as entertainment. Many 3D applications are implemented for different device classes from ordinary PCs, over mobile devices, up to immersive Virtual Reality installations. Further, development of 3D applications is an interdisciplinary process that involves disjoint groups of 3D content designers and programmers. Tooling support is mostly limited either to the 3D modeling or to the programming task. For instance, the 3D content may be declared in X3D or VRML and is edited within a 3D modeling tool, like Blender¹ or 3DS Max², while the program code, that could be encoded e. g. in JavaScript or Java, is edited in an appropriate IDE, such as Eclipse³ or

Netbeans⁴. However, at some point the 3D content as well as the program code have to be integrated into the overall 3D application. Due to the concurrent development process and different terminologies used in the two developer groups, inconsistencies may appear within the developed software. For example, a 3D designer might change the name of a 3D object while the programmer depends on the original naming to address the 3D object from the application code. Therefore, 3D applications are usually developed within an iterative process.

As argued in [22], the above challenges call for an iterative, model-driven development (MDD) process, particularly with round-trip engineering (RTE) methods. MDD, by using a common abstract model for the 3D and program code, supports multi-platform development and can be instrumental in the resolution of the above-mentioned inconsistencies. RTE is a software development process that combines automated forward (model-to-code) with reverse (code-to-model) transformations [4]. In the context of iterative, model-driven 3D development, RTE offers to simplify the necessary synchronization between the common model and the different code bases.

RTE has proven useful in the development of “conventional” software as exemplified by several existing integrated tools supporting the simultaneous editing of UML diagrams and program code. However, development of 3D applications involves (at least) two types of program code, i. e. 3D code and application logic (in case of multi-platform development, there will even be several variants of the program code). Due to the concurrent development process in conjunction with the preference for very different modeling/programming tools between 3D designers and programmers, the use of a (yet to be developed) integrated tool for the various tasks seems not advisable for 3D development. Instead, an approach should be taken, where the distinct developer groups (software modelers, 3D designers, and programmers) each can employ their tools of choice. As a consequence, synchronization between the model and program code bases must occur in a non-simultaneous fashion.

In the following, we introduce a novel RTE approach for development of 3D applications. Its main features are:

- A domain specific language (DSL) called SSIML [23] is

¹<http://blender.org>

²<http://autodesk.de>

³<http://eclipse.org>

⁴<http://netbeans.org>

used to specify an abstract model of a 3D application. SSIML models can be specified in a graphical editor (Section 2).

- Forward model-to-model transformations (*M2M*) and code generation (model-to-text, *M2T*) for multiple target languages, such as JavaScript and X3D (Sections 3.2 through 3.4).
- Reverse transformations and model merging for synchronization of modified models and source code artifacts (Section 3.5).

We also discuss the state of our implementation and first experiences (Section 4), relate the proposed approach to other work (Section 5), before we finally conclude (Section 6).

2. Preliminaries

2.1 An illustrating example

For a better illustration of 3D application development, involving our approach for round-trip engineering, we introduce a small example that is a simulation of a robot in a factory. The 3D scene is built up of the following 3D objects: A factory hall (Figure 1(a)) that already contains accessories like boxes, a simplified model of an industrial robot (Figure 1(b)) that is explicitly made up of its single components and a console with separated buttons and desk (Figure 1(c)). Furthermore, the application shall provide an interactive part that allows the user to control the robot directly within the 3D scene, by pressing buttons on the console using the mouse. To declare the 3D scene and to exemplify our round-trip process we chose X3D [9] as the first target language. To easily publish the example application in the web we use X3DOM [5] that allows for rendering X3D scenes in modern web browsers using WebGL. Additionally, we use JavaScript as second target language to animate the 3D robot.

2.2 Graphical Model Editor

We use SSIML (*Scene Structure and Integration Modeling Language*) [23], which allows for modeling so called 3D scene graphs at a high level of abstraction. 3D scene graphs are directed acyclic graphs (DAGs) forming transformation hierarchies of 3D objects. Furthermore, in interactive 3D applications, there typically exist interrelationships between elements of the 3D scene and *application components* that are responsible for flow control and that hold additional data. For instance, an application component needs to modify node attributes of the 3D scene, e. g. to animate the robot.

We developed an editor to create SSIML domain models to describe the 3D application, including the 3D scene as well as interrelationship elements. The graphical SSIML editor is implemented using the Graphical Modeling Framework⁵ (*GMF*) atop of the Eclipse Modeling Framework⁶ (*EMF*).

⁵<http://eclipse.org/modeling/gmf/>

⁶<http://eclipse.org/modeling/emf/>



Fig. 1: (a) The 3D factory hall includes the structure and accessories. (b) The single parts of the industrial robot. (c) A simple console with buttons, meant to rotate the segments of the robot.

To improve the interdisciplinary development process with 3D designers, an evaluated icon design is used.

Figure 2 shows the graphically modelled example described in Section 2.1. The scene model contains several group nodes to structure the 3D scene as well as the single objects. The object nodes contain relevant attributes, e. g., each instance of the four console buttons is connected to a touch sensor attribute. When a button is pressed by a user, the associated sensor will notify the RobotControl class in the interrelationship model. This application component, which will later be realized as a set of JavaScript functions, is able to access the transformation attributes and therefore can animate the robot.

3. Concept: RTE for 3D Applications

3.1 Basic approach

RTE combines forward engineering with reverse engineering, while preserving manually developed code, that cannot be reflected in the domain model. The algorithm which synchronizes different code bases has to resolve conflicts that may appear due to the concurrent code development.

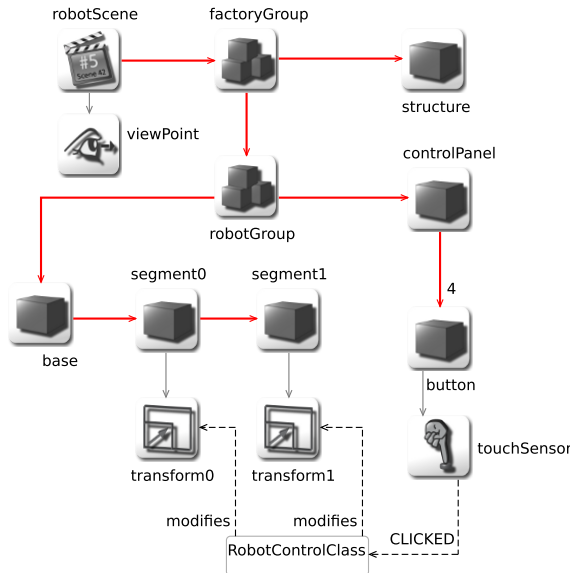


Fig. 2: The robot example in the graphical SSIML editor.

Common methods for (simultaneous) RTE, as provided by various tools (cf. Section 5), are not applicable in the domain of 3D applications. As mentioned above, 3D applications are made up of at least two target languages, e.g. JavaScript and X3D. 3D content designers will use their favorite 3D modeling tools whereas an adequate IDE will be used by programmers. Therefore, modifications to the program code and 3D content cannot be tracked and mapped to the domain model in real time. Thus, an asynchronous – or non-simultaneous– round-trip is required.

Two methods are common for storing the model and the source code, both using a decentralized approach: In the first approach, model and source code are stored separately, in a model file and in the respective source code files. The manually created source code contains the detailed information of the application, that cannot (or should not) be reflected in the domain model. In the reverse step, changes are extracted from the source code and applied to the model. In the forward step of the next iteration, where changes in the model are applied

to the code, manually developed source code from prior iterations has to be preserved. This can be achieved by *active code generation*, in particular through *weaving* the code or *protected regions* [19], [16]. While this storage approach allows for synchronizing all language elements that can be reflected in the domain model, it is also limited to them. The platform independent SSIML model does not contain specific features, e.g. a concrete color value of a 3D object. Therefore, with this approach, same elements from different target languages, e.g. pure X3D code and slightly different X3DOM web code, cannot be synchronized.

In the second approach, the model is not stored separately, but within the source code [6]. However, in 3D development multiple target languages (TLs) are used and thus, the model would be redundantly contained within several code bases. Further, to preserve consistency of the full application, synchronizations have to be performed between each pair of the TLs in both directions. Therefore the number of merge transformations quickly increases.

To overcome the synchronization issues of the above approaches we use a multi-tiered approach to transform between model and code. Specifically, we introduce a generic *intermediate model* (IM) and target language-specific *Abstract Syntax Trees* (ASTs) [12] as additional representations to the transformation process. As a positive side effect, complex transformations are split into sequences of simpler ones and the extensive code generation process becomes more manageable, cf. [4]. The introduction of language-specific models has been proposed before, e.g. in the context of multi-target user interfaces [7].

Figure 3 describes our overall round-trip process. The SSIML model (during forward transformation) as well as the ASTs (during reverse transformation) are transformed into a corresponding IM representation. Each IM is then merged into a *persistent* IM, that serves as a central storage for the transformed SSIML model as well as for every transformed AST (that contains the code semantics). Therefore, the IM allows for the complete re-generation of the SSIML model and the ASTs. Code is generated from the ASTs in the last forward step (Section 3.4).

The IM describes the SSIML and AST contents in a generic fashion (much like an object-oriented database would), see Figure 4. IM classes contain attributes to store data as well as specific meta data of the involved languages and the Ecore-based SSIML model. In this way, the IM can accommodate all relevant information from the SSIML model and code artifacts. Since IMs from different code based have to be merged into one single IM during the reverse step, conflicts may appear. To handle these conflicts without the loss of data each intermediate attribute provides a history that holds information about recent changes, in particular the modification type and the previous value. The IM is not meant to be viewed or edited by a user.

In order to facilitate a non-simultaneous round-trip, it must

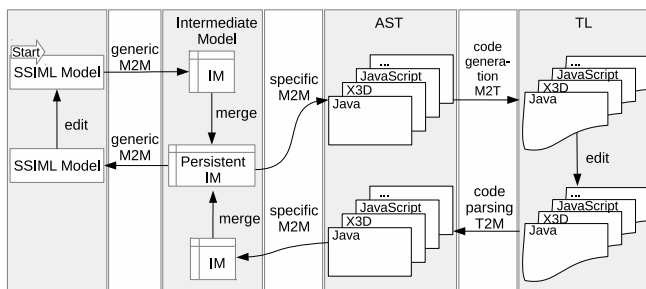


Fig. 3: The overall round-trip process with the participating models and transformations between them.

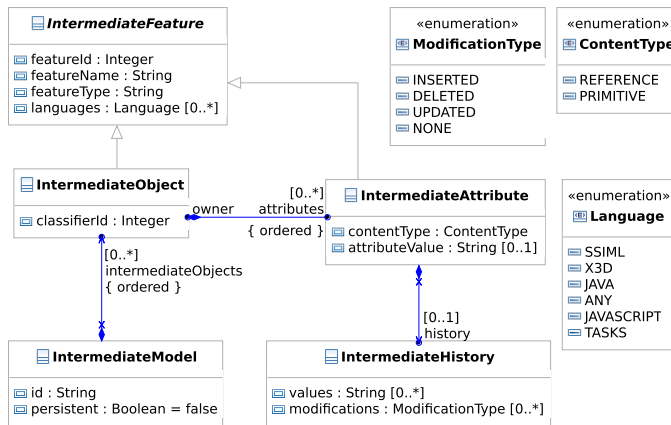


Fig. 4: The generic structure of the intermediate metamodel.

be possible to uniquely identify an element's representations in the various models and code bases, even if the element is substantially changed during the editing process. Therefore, as in [3], we provide a unique ID (UUID) for each element. The ID is stored in its model element and in comments within the source code.

3.2 Generic transformations (SSIML to IM)

The first step in our round-trip scenario is to transform the graphically created SSIML model into a new IM. As noted above, the IM has a generic structure and thus generic M2M transformations are appropriate (Figure 3, generic M2M). Transformations between SSIML and IM are performed with EMF, that provides a reflective API [20], which is comparable to the Java reflection API but operates on Ecore classes. We transform each SSIML Element to an *intermediate object* and accordingly each associated attribute to an *intermediate attribute*. Since no further conversions are performed during the forward transformation, the SSIML model can also be reconstructed from the IM, by using generic reverse transformations.

3.3 Specific transformations (IM to AST)

The next step of the forward phase is the transformation of the (persistent) IM into several ASTs. Referring to our example, where a 3D web application based on X3D and JavaScript has to be generated, two transformations are performed. The description of the 3D scene is extracted from the IM and converted into an AST for the X3D code. Analogously, the definition for the application is transformed into an AST for the JavaScript code. Generic transformations are not suitable for this task, since target language-specific, complex mappings and assignments have to be performed. Rule-based and hybrid transformation languages like ETL [14] operate on models and allow for the declarative transformation of model elements as well as for further assignments in an imperative manner. Rules to assign cross references within the AST or to create the concrete structure of the

AST assume that the involved elements are present [18]. Using ETL, we split each IM-to-AST transformation into three phases: In the first phase, all elements of the AST are created, while their content and the concrete structure of the tree remain disregarded. In order to reduce the amount of necessary transform rules, we additionally make use of a simple mapping model, i.e. *Metamodel Mappings* [15] that defines associations of IM elements (actually SSIML elements) to language elements of the AST. Listing 1 shows the rule to transform relevant intermediate objects (according to the mapping model) into their equivalent JavaScript representation.

```

rule      IObjectToJavaScriptElement
transform imObj:im!IntermediateObject
to Any
{
    var javascriptfile:jsm!JavaScriptFile
    = jsm!JavaScriptFile.allInstances.first();

    var mapping = mapping!SourceElement.allInstances
    .selectOne(s|s.type = imObj.featureType)
    .mappingContent.asSequence();

    for(targetElement in mapping)
    {
        var jsElement = createJavaScriptElement(...);
        javascriptfile.objects.add(jsElement);
    }
}

```

Listing 1: An ETL Rule to instantiate JavaScript elements from intermediate objects, using an additional mapping.

In the second phase, we establish the concrete structure of the AST and assign uninitialized attributes. In the third and final phase, we remove inconsistencies and evaluate the AST in a *post block* [14]. E.g., the viewpoint attribute in Figure 2 has no association to any JavaScript element. However, due to the static mapping a corresponding JavaScript element is created during the transformation. Since this viewpoint attribute is not meant to be addressed from the JavaScript code it has to be removed from the AST.

3.4 Code generation (AST to code)

In the final step of the forward transformation pipeline, the ASTs need to be converted to code skeletons in the respective TLs. Code generation can be performed with various transformation languages, with some popular ones being JET [1], Xpand [2], EGL [19] and MOFScript [16]. However, these languages do not provide support for reverse transformations. Therefore, we opted for another transformation language, Xtext [10], that allows us to re-use parts of the specification of the AST-to-code transformation (Figure 3, M2T) for the reverse code-to-AST specification.

Besides a grammar, Xtext requires a metamodel for each involved TL (X3D and JavaScript, in our case). For our purposes, it suffices that these metamodels cover relevant subsets of the TLs as not every construct of the TL has to be represented as a distinct type in the AST [12]. In contrast to essential JavaScript elements, such as the functions

RobotControlClass or *init*, arbitrary hand coded statements within JavaScript functions have no equivalent representation in the domain model. Therefore, it is not necessary to semantically distinguish between these statements and they can be generalized to arbitrary content.

```

/**
 * @id: JavaScriptFile 12...
 */
function init( ) {

  /**
   * @id: Assignment 38...
   */
  var robotControlClass = new RobotControlClass( );

  /**
   * @id: Object 1b...
   * @id: NodeAttributeRelationship 1f...
   * @id: TouchSensor db...
   * @id: EventRelationship c4...
   */
  document.querySelector( ' [DEF="button0"] ' )
    .addEventListener( "click",
      robotControlClass.button0_CLICKED );
  ...
}

/**
 * @id: JavaScriptClass 38...
 */
function RobotControlClass( ) {

  /**
   * @id: ValueAccessRelationship bf...
   * @id: Object c1...
   */
  var transform0
    = document.querySelector( ' [DEF="transform0"] ' );

  /**
   * @id: ValueAccessRelationship 24...
   * @id: Object 83...
   */
  var transform1
    = document.querySelector( ' [DEF="transform1"] ' );

  /**
   * @id: UserFunction c4...
   */
  this.button0_CLICKED = function( obj ) {
  }
  ...
}

```

Listing 2: JavaScript code skeleton to access the 3D scene.

Listing 2 shows the JavaScript code skeleton that will be generated from its respective AST. The JavaScript code emulates the structure of object orientated languages, such as Java or C++, to simplify the adaption to other target languages in future work. The generated code is related to the original SSIML domain model of Figure 2 in the following way: The SSIML application component *RobotControlClass* results in a JavaScript function of the same name that creates a suitable JavaScript object. The application component's action relationships to nodes *transform0* and *transform1* result in member variables of the JavaScript object. In the SSIML model, each of the 4 buttons of the control panel is associated with a *TouchSensor*. The *CLICKED* event relationship to *RobotControlClass* results in 4 event listen-

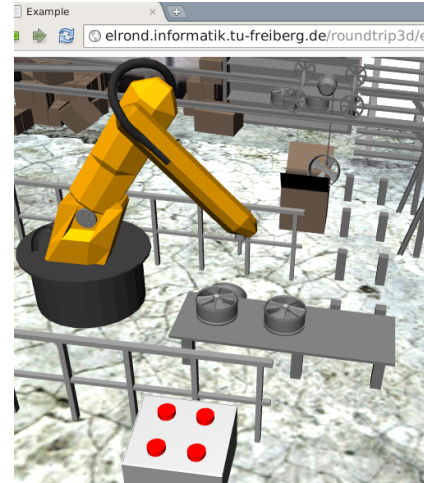


Fig. 5: The 3D application with the modified scene and interactive control functionalities.

ers *button0_clicked*, etc. Further application logic, e.g. to calculate the rotation values for a robot animation, will be manually programmed by filling in code stubs or in additional functions. Concurrently to the JavaScript refinement, the generated X3D code is customized by 3D designers, e.g. by placing the 3D objects in their final positions, fixing lighting and viewpoints, etc.

The edited X3D model and JavaScript code now define (a first version of) the 3D application⁷. E.g. the robot in Figure 5 is animated by JavaScript functions, when the respective buttons on the control panel are pressed.

3.5 Reverse transformations and model merging

For the reverse transformation from code to AST (Figure 3, T2M), Xtext can be used to generate parsers for converting the source code into corresponding ASTs. Subsequently, each AST is transformed to a corresponding IM. Every AST element is converted into the intermediate representation of the respective SSIML element (Figure 3, M2M). Reverse transformations have been implemented for X3D and JavaScript. For the reverse way we use a mapping model, which is the inverse to the one of the forward transformation (Section 3.3). Remaining elements from the AST without equivalent SSIML representation will be stored in the IM without further conversions.

Next, each IM needs to be merged into the persistent IM. We distinguish between three basic modification types that may occur during an editing process:⁸

⁷Available at: <http://elrond.informatik.tu-freiberg.de/roundtrip3d/example>

⁸Sometimes a further edit operation "move elements" is considered [8]. We can reduce the move operation to an update of the respective parent-child properties, since the IM has a flat hierarchy.

- 1) **Updating** model elements. The changing of attributes, e. g., the renaming of 3D objects, is probably the most frequent modification.
- 2) **Inserting** new model elements. For instance a 3D designer adds a new light source to the scene.
- 3) **Deleting** existing model elements. E. g., a 3D designer removes a button, that is associated with a touch sensor in the JavaScript code.

Due to the unique identifiers, changes to each element can be detected. However, in case of conflicting modifications, a fully automatic merge will not be possible [3]. For instance, when a programmer has changed the name of the 3D object *segment0* to *seg0* in the JavaScript code, while the 3D designer renamed it to *middle_segment* in the X3D code, it is unclear which name should finally be applied. To resolve such conflicts, we store a history for every intermediate attribute. This history contains applied modifications (*updated*, *inserted* or *deleted*) and prior attribute values (Figure 4). Intermediate objects and attributes from each participating IM are merged into the persistent IM. Afterwards, every intermediate attribute may contain a history with conflicting operations. To reestablish consistency of the persistent IM, the software designer is prompted which of the conflicting operations finally has to be applied.

In the final reverse transformation from the IM to the SSIML model (Figure 3, generic M2M), all relevant SSIML elements are extracted from the persistent IM in a generic manner, as described in Section 3.2.

In RTE, of course, merging also has to be performed during the forward transformation (SSIML to IM). This is done analogously to the described merging during the reverse phase.

4. Discussion

Our first approach for RTE was to directly generate code from a SSIML model with MOFScript. This straightforward model-to-text transformation, however, turned out to be hard to manage and extend. Therefore, our multi-tiered approach aims at manageable transformations, which can be achieved best by using different transformation languages. Reflection-like transformations in Java convert between the SSIML model and its generic intermediate representation. Combining hybrid ETL with an declarative and extendable mapping model allows for well structured conversions between the IM and ASTs. Through Xtext, model-to-text as well as text-to-model transformations are obtained automatically from the language descriptions without the need of implementing them manually. Our implementations of forward and reverse engineering confirm the practicality of the multi-tiered approach in contrast to the immediate model-to-code conversion. For merging, we use the Epsilon Merging Language (EML) [13] which also allows for user interactions during the merge transformations. Storing model information and the complete code base in one common IM representation reduces

the merging complexity dramatically. Instead of numerous language specific synchronizations among all participating languages, only one model merging algorithm is needed.

A potential drawback of our approach results from the use of IDs for tracing elements through the different models and source codes. While the use of IDs is a common method for this problem, it will also result in the *pollution* of models and source code with IDs. In future work, we plan to experiment with tree-diff algorithms in order to trace elements without the need for IDs.

To synchronize different platform variants of code (e. g. X3D and VRML), the metamodels and grammars must be specified in detail. Implementing the complete X3D specification is also very extensive. For our round-trip scenario, we mainly cover the basic X3D *Interchange profile*.

5. Related work

Existing tools, such as *UML LAB*⁹ or *Together*¹⁰, provide support for *simultaneous* RTE. Models (rendered as UML class or sequence diagrams) and source code are edited in the same IDE, while changes to each are directly applied to its counterpart. Borland's LiveSource-Technology supports multiple TLs (Java, C++ and IDL), while the complete model information is stored in the source code. Synchronizations are performed between domain model and the respective TLs and therefore are limited to artifacts which can be reflected in the model [6]. However, the domain of interactive 3D applications is not covered by any of these tools.

The above tools mostly realize the concepts described in [21]. Different views of a software application, e. g. a model view and an implementation (code) view, have to be synchronized continuously to minimize the difference in between them. Furthermore it is assumed that an element – or actually a meta-element – remains the same within all views. These two preconditions are inconsistent with the assumptions of our RTE approach, where complex mappings between model elements and the source code are performed and continuous, simultaneous synchronization is not possible.

Previous work on model-centric development [17] describes methods for multi-platform adaption of existing source code. More concretely, an X3D scene can be used to derive a SSIML model, which then can be used for further MDD, although not full iterative RTE.

The authors of [3] describe model management in the context of a version control system. To merge two modified models into the original one, their difference (the *delta*) is calculated. This delta, given as a sequence of operations, is used to detect conflicts which are resolved (automatically or manually) by modifying the operations. Storing the applied modifications (update, insert and delete) during the merge

⁹<http://www.uml-lab.com>

¹⁰<http://www.borland.com/us/products/together/>

of the IMs (Section 3.5) is a simplified determination of conflicting operations.

Abmann abstracts RTE to mathematical domain transformations and gives definitions for an *Automatic Roundtrip Engineering*, where an inverse transformation to a given forward transformation can be calculated [4]. In the context of our RTE, an automatic determination of an inverse transformation would be far from trivial, not least because such an inverse needs not to be unique.

The authors in [11] introduce a new approach to model RTE by using abductive reasoning. Thereby, modifications of the target model are observed and a hypothesis for changes to the source model with respect to the transformations is derived. Although this approach eases the reverse phase of RTE, in our case, it does not allow for resolving conflicts from different intermediate models during the merge.

6. Conclusion

We presented a new approach to the structured development of 3D applications based on round-trip engineering. 3D and program code are generated from a common model specified in SSIML, a DSL for modeling 3D applications. Reverse transformations and model merging are used for the non-simultaneous synchronization of the SSIML model with changes at the code level. The approach was designed in such a way, that it honors the concurrent development process involving 3D designers on the one side and programmers on the other. To the best of our knowledge, the presented approach is first to apply round-trip engineering techniques to the development of 3D applications.

In the current state of our implementation, SSIML models can be specified in a visual editor. Forward and reverse transformations, including the generation of code skeletons as well as the non-simultaneous merging of intermediate models, are implemented for X3D and JavaScript. Building on the X3DOM framework, functional 3D applications for the web using X3D and JavaScript can be developed.

For future work, we intend to use tree diff algorithms to overcome the necessity for UUIDs. Also, support for further platforms such as immersive Virtual Reality or Augmented Reality will be investigated.

Acknowledgements

This research was supported by the Deutsche Forschungsgemeinschaft (DFG).

References

- [1] The Eclipse Foundation: The Model to Text (M2T) project, JET (2012), <http://www.eclipse.org/modeling/m2t>
- [2] The Eclipse Foundation: The Model to Text (M2T) project, Xpand (2012), <http://www.eclipse.org/modeling/m2t>
- [3] Alanen, M., Porres, I.: Difference and union of models. pp. 2–17 (2003), <http://www.springerlink.com/content/y13a5qt85f6f1uak>
- [4] Abmann, U.: Automatic roundtrip engineering. *Electr. Notes Theor. Comput. Sci.* 82(5) (2003), proceedings of the Fifth Workshop on Quantitative Aspects of Programming Languages (QAPL 2007)
- [5] Behr, J., Eschler, P., Jung, Y., ZÄüllner, M.: X3dom: a dom-based html5/x3d integration model. In: Spencer, S.N., Fellner, D.W., Behr, J., Walczak, K. (eds.) *Web3D*. pp. 127–135. ACM (2009)
- [6] Borland: Case study template (2008), http://www.borland.com/resources/en/pdf/products/together/together_faq.pdf
- [7] Calvary, G.: A unifying reference framework for multi-target user interfaces. *Interacting with Computers* 15(3), 289–308 (2003)
- [8] Chawathe, S.S., Rajaraman, A., Garcia-Molina, H., Widom, J.: Change detection in hierarchically structured information. *SIGMOD Rec.* 25, 493–504 (June 1996)
- [9] Daly, L., Brutzman, D.: X3d: extensible 3d graphics standard. In: *SIGGRAPH Asia '08: ACM SIGGRAPH ASIA 2008 courses*. pp. 1–6. ACM, New York, NY, USA (2008)
- [10] Efftinge, S., Völter, M.: oAW xText: A framework for textual DSLs. In: *Eclipsecon Summit Europe 2006* (Nov 2006), <http://www.eclipse.org/Xtext>
- [11] Hettel, T., Lawley, M., Raymond, K.: Towards model round-trip engineering: An abductive approach. In: *Proceedings of the 2nd International Conference on Theory and Practice of Model Transformations*. pp. 100–115. ICMT 2009, Springer-Verlag, Berlin, Heidelberg (2009)
- [12] Jones, J.: Abstract syntax tree implementation idioms. *Pattern Languages of Program Design* (2003), proceedings of the 10th Conference on Pattern Languages of Programs (PLoP2003)
- [13] Kolovos, D., Paige, R., Polack, F.: Merging Models with the Epsilon Merging Language (EML). In: Nierstrasz, O., Whittle, J., Harel, D., Reggio, G. (eds.) *Model Driven Engineering Languages and Systems*, vol. 4199, chap. 16, pp. 215–229. Springer Berlin Heidelberg, Berlin, Heidelberg (2006)
- [14] Kolovos, D.S., Paige, R.F., Polack, F.: The epsilon transformation language. In: Vallecillo, A., Gray, J., Pierantonio, A. (eds.) *Theory and Practice of Model Transformations, First International Conference, ICMT 2008, ZÄijrich, Switzerland, July 1-2, 2008, Proceedings*. *Lecture Notes in Computer Science*, vol. 5063, pp. 46–60. Springer (2008)
- [15] Miller, J., Mukerji, J.: Mda guide version 1.0.1. Tech. rep., Object Management Group (OMG) (2003)
- [16] Oldevik, J.: MOFScript Eclipse Plug-In: Metamodel-Based Code Generation. In: *Proceedings of the Eclipse Technology eXchange workshop (eTX)* (2006)
- [17] Pleuss, A., Vitzthum, A., Hussmann, H.: Integrating heterogeneous tools into model-centric development of interactive applications. In: Engels, G., Opdyke, B., Schmidt, D.C., Weil, F. (eds.) *MODELS. Lecture Notes in Computer Science*, vol. 4735, pp. 241–255. Springer (2007), <http://dblp.uni-trier.de/db/conf/models/models2007.html#PleussVH07>
- [18] Rahimi, Lano: Development and evaluation process of model transformation. *SERP* (2011)
- [19] Rose, L.M., Paige, R.F., Kolovos, D.S., Polack, F.: The epsilon generation language. In: Schieferdecker, I., Hartman, A. (eds.) *Model Driven Architecture - Foundations and Applications, 4th European Conference, ECMDA-FA 2008, Berlin, Germany, June 9-13, 2008, Proceedings*. *Lecture Notes in Computer Science*, vol. 5095, pp. 1–16. Springer (2008)
- [20] Steinberg, D., Budinsky, F., Paternostro, M., Merks, E.: *EMF: Eclipse Modeling Framework*. Addison-Wesley, Boston, MA, 2. edn. (2009)
- [21] Van Paesschen, E., De Meuter, W., D'Hondt, M.: Selfsync: a dynamic round-trip engineering environment. In: *Companion to the 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*. pp. 146–147. OOPSLA '05, ACM, New York, NY, USA (2005)
- [22] Vitzthum, A., Jung, B.: Iterative model driven VR and AR development with round trip engineering. In: *Proc. SEARIS Workshop at the IEEE Virtual Reality 2010 Conference*. Shaker (2010)
- [23] Vitzthum, A., Pleuß, A.: SSIML: Designing structure and application integration of 3D scenes. In: *Proceedings of the tenth international conference on 3D Web technology*. pp. 9–17. *Web3D '05*, ACM, New York, NY, USA (2005)

Domain-specific transformation of the REA enterprise ontology

Zdenek Melis

University of Ostrava
Czech Republic
e-mail: Zdenek.Melis@Osu.cz

Jaroslav Zacek

University of Ostrava
Czech Republic
e-mail: Jaroslav.Zacek@Osu.cz

Frantisek Hunka

University of Ostrava
Czech Republic
e-mail: Frantisek.Hunka@Osu.cz

Abstract—The paper deals with the general description of the methodology of the transformation of basic concepts of the REA enterprise ontology (Resource, Event, Agent) from the initial visual modeling interface to model source codes. The paper describes the structure of the script of the domain-specific language (DSL) and its subsequent transformation into the executable source code using abstract classes for defining the structure of the template code. The aim is the creation of the basic structure of the concept of templates and layout the code generated from the created model. The model is due to excessive complexity limited to the use of three basic concepts of REA enterprise ontology.

Keywords- REA ontology; DSM; DSL; transformation

I. INTRODUCTION

Due to the increasing complexity of information systems, demands for the control clarity and the simplicity are increasing. Technologies that use a visual environment come to the fore. One of the leading technologies based on a visual programming is a domain-specific modeling (DSM) [1]. It focuses on one particular domain, which provides the syntax and the semantic of the visual language and the ability to transform created models into the executable source code [1]. Business process modeling is one of complex problems that use the visual modeling technology. With regard to the structure of the DSM and code generation possibilities, the most appropriate technology of the description of business processes seems to be the REA ontology [2]. The object-oriented structure of the REA ontology allows transformation of models into other structures, such as source code or database schema [8, 9, 10].

II. DOMAIN-SPECIFIC MODELING

Domain-specific modeling is a software engineering methodology for software development [1]. The primary focus of DSM is automated development of applications in a specific domain based on principles of visual programming. Unlike traditional modeling principles that have a universal focus, DSM is based on the entirely specific modeled domain.

The DSM architecture is three layered, although individual layers overlay to each other. The highest and for the user the most visible layer is the language. The narrow

focus on a specific domain allows language to correspond by its whole structure to domain terms. Syntactic aspects of the language are not limited to textual form, but they can take any visual form representing concepts of particular domain. Unlike generally focused traditional principles, semantic aspects of the language are contained. The second layer is the generator, which transforms created model into language of the target platform. The last layer is the domain framework. It creates a supportive environment for the generator, which includes many features, such as defining the interface for the generator, integration of the generated code into the system, removing duplicate parts, and many others.

The main advantage of DSM is the automated transformation without mapping that is a frequent source of errors. Using of the domain language brings simplicity and easy manipulation with tool even for non-technical users who can intuitively use the tool through the knowledge of the problem domain. The toughest stage in the development process is creating of the modeling tool. Once the tool is developed, using this tool for creating software is very simple and fast. It can increase development productivity up to ten times [5, 6].

III. THE REA ONTOLOGY

The REA enterprise ontology is the concept for designing and model creation of enterprise infrastructures. It is based on resource ownership and its exchange. The aim of most businesses is the profit generation and therefore they must ensure the effectiveness, the efficiency and the adaptability of their business processes and it cannot be done without their modeling and subsequent analysis [4]. There are many of business processes modeling tools, but due to inadequate level of abstraction and the use of general concepts they are not usable enough for the business process modeling. Rather than general modeling techniques companies use expensive software created directly to the specific requirements of a particular enterprise.

REA ontology does not use general concepts but specific. They increase the amount of represented data, while maintaining the simplicity of the model. The REA ontology model offers 4 levels of abstraction [2, 7]. The highest level is Value System Level, which represents view of the flow of

resources between the company and its business partners. The second level is Value Chain Level describing links between business processes within a company. The third level, REA model level, describes a specific business process and represents the change in the value of resources. The various concepts of this level can be divided into two groups – the operational level and the policy level. The operational level includes the basic concepts describing the specific facts and events that have already happened. Here are included concepts forming the name of this ontology - economic resource, event and agent. The policy level is an extension of the operational level by concepts specifying rules or allowing planning. The lowest level of an abstraction is the Task level, which describes the model instance level, making it an implementation-dependent.

This paper deals only with the transformation of the operational level of the REA model level. It has three basic concepts: economic resource, event and agent. The resource represents an economic material that the business wants to plan, manage and monitor. It can include products, money, services or for example workmen. The economic agent is a person, group of people or the whole company that has control over resources. The economic event describes incremental or decrement change of resource. From the model perspective the economic event is key for preserving information, because it determines who, when, why and how the resource was changed [3].

One of reasons for choosing the REA ontology is object-oriented structure support that is necessary for successful transformation. The REA ontology model also includes internal rules for verifying the consistency of the model, ensuring correctness of created links. At the same time models are simple and understandable for ordinary users who will work with it, but sufficiently precise for its automation [2].

IV. MODEL TRANSFORMATION

The transformation itself consists of several steps. At the beginning the user creates a model of a business process in the visual interface. This interface contains methods for ensuring the basic model validation by preventing incorrect links creation alternatively prevent execution of the second phase - generation.

During the second phase the DSL script is generated from a visual model, containing basic elements of the structure of the model and on this basis the source code is generated.

A. Visual interface

The user creates a business process models in the visual environment that provides the basic user interface and performs the validation and the partial verification of the model. It provides basic semantic correctness of the model and prevents the generation of incomplete structures. The visual interface contains the common label of an entity type and its name and basic data attributes defining its properties. Figure 1 shows the visual form of the model with attributes.

For simplicity and clarity only basic entities representing resources, events and agents are used.

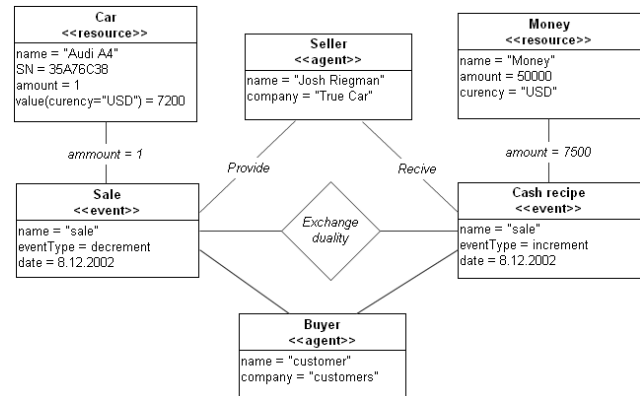


Figure 1: Visual representation of the model

The header of each element contains the name and the type of entity. Under the header there are attributes with a predefined structure specified by the type of entity. If necessary the user can expand, or completely change the structure by adding new attributes or changing existing ones. The entity *Car* can serve as an example. In addition to basic attributes *Name* and *Amount* the entity is expanded by *SN* attribute representing a specific serial number of the car and *Value* that represents the value that the resource has for the company (for example total production costs).

The value on the link between the resource and the event indicates the amount of resources that is changed within the event.

B. DSL script

Once the model is completed and validation criteria are met, a domain-specific language (DSL) script constituting the regulation for the generator is created. The script is basically the export of the created model into XML format with the omission of data related to the visual interface of the model (such as the placement of elements, their size, color ...).

The following code fragment represents the DSL script showing the transfer of the resource entity from the model shown at Figure 1:

```
<resource title="Car">
  <id type="int">1</id>
  <name type="String">Audi A4</name>
  <sn type="String">35A76C38</sn>
  <amount type="int">1</amount>
  <value type="int" currency="USD">
7200</value>
</resource>
<resource title="Money">
  <id type="int">2</id>
  <name type="String">Money</name>
```

```

    <amount type="int">50000</amount>
    <currency type="String">
USD</currency>
</resource>

```

The code contains new items added by user - *value* that represents the value of the economic resource and *currency* for defining the exchange of funds handled by business.

Individual links are merged with the appropriate entities. An example may be the *Stackflow* link (link between the Resource and the Event) containing information on the amount of increase / decrease of resources within a single event. This attribute is converted into an Event entity. As mentioned before, the main carrier of information is an Event entity, which has a significant role in obtaining data from the model. For this reason, most of links of this model moves into this entity. Each entity has a unique ID, which is used for the unique identification of the element and for replacing individual links by creating reference to that ID.

The following code fragment shows one of event entities:

```

<event title="Sale">
  <id type="int">1</id>
  <name type="String">Sale</name>
  <eventType
type="EventType">decrement</eventType>
  <agentReceiveID
type="int">8</agentReceiveID>
  <agentProvideID
type="int">2</agentProvideID>
  <resourceID
type="int">1</resourceID>
  <date type="String">8.12.2002</date>
  <amount type="int">1</amount>
</event>

```

EventType attribute is determined by the type of link provide/receive, if the event is from the business point of view incremental or decrement and according to these links agents are determined too.

Exchange duality saves all references to events connected with duality.

```

<exchangeDuality>
  <id type="int">1</id>
  <eventID type="int">1</eventID>
  <eventID type="int">2</eventID>
</exchangeDuality>

```

C. Source code generator

The last phase of the transformation itself is generating the source code from the created DSL script. Creating a complete general generator would be inefficient and implementationally difficult due to the unchanging structure of REA ontology elements. Fixed domain structure ensures durability and stability of domain terms, which allows predefining the general structure of basic elements. Due to their limited number it is possible to predefine a common part of the code that is same in all cases of the use of the entity as a template. In case of the above model, which is restricted for using of only three basic entities, two types of templates are used - an abstract class and a class template.

An abstract class contains the basic code structure, layout of methods and predefined basic attributes that the given entity must contain. In the case of transformation of the model restricted to using only three entities the same number of abstract classes are fully enough. When extending the model by other semantic abstractions, the number of abstract classes is not linear to the number of used entities because some entities may have more abstract classes based on specific uses of that entity in the model. Basic abstract classes are therefore *AbstractAgent*, *AbstractEvent* and *AbstractResource*.

The abstract class defining the agent entity contains the basic attributes *id*, *name* and *company* and their access methods, as shown in Figure 2.

AbstractAgent	
-	company: String
-	id: int
-	name: String
+	getCompany() : String
+	getId() : int
+	getName() : String
+	setCompany(String) : void
+	setId(int) : void
+	setName(String) : void
+	toString() : String

Figure 2: Abstract class for the agent entity

These attributes are common to all agents and can be individually extended by additional parameters specifying a particular agent. Using default parameters is not mandatory, but it is recommended. The creator of the model can ignore these parameters and creates new one. The only restriction is that variable names defined in the parent class cannot be used again with different data type.

The abstract class for a resource entity is defined in a similar way (Figure 3). *Amount* attribute indicates the amount of resources from the business perspective. The other attributes (*id* and *name*) are the same as attributes in *AbstractAgent* class.

AbstractResource
- amount: int - id: int - name: String
+ getAmount(): int + getId(): int + getName(): String + setAmount(int): void + setId(int): void + setName(String): void + toString(): String

Figure 3: Abstract class for the resource entity

The last abstract class is *AbstractEvent* that creates a draft for the event entity (see Figure 4). Unlike the other two abstract classes it contains the most methods because the event is carrier of basic properties of the model. Attributes *agentProvideID*, *agentReceiveID* and *resourceID* store links with individual agents and a resource.

AbstractEvent
- agentProvideID: int - agentReceiveID: int - amount: int - date: java.util.Calendar - eventType: EventType - id: int - name: String - resourceID: int
+ getAgentProvideID(): int + getAgentReceiveID(): int + getAmount(): int + getDate(): String + getEventType(): EventType + getId(): int + getName(): String + getResourceID(): int + setAgentProvideID(int): void + setAgentReceiveID(int): void + setAmount(int): void + setDate(String): void + setEventType(EventType): void + setId(int): void + setName(String): void + setResourceID(int): void + toString(): String

Figure 4: Abstract class for the event entity

In addition to basic attributes such as *id* and *name* the attribute *amount* also figures here. It indicates the amount of resources that is changed within the event. Another necessary attribute is *date*, recording the date of the past event. Calendar class is intended for its preservation, but the output of the visual interface returns the date as a string and therefore it is necessary to convert the date inside access

methods. The last parameter is *eventType*, that specifies the type of an event, whether it is from the business perspective incremental or decrement. It is determined by a constant of enumerator *EventType*, which is part of this class.

Abstract classes are used to define basic parameters for the generated class. They are generated using class templates. They operate on a simple principle: instead of generating the whole structure of the code, such as headers of classes, methods, etc., the appropriate template that contains all of these structures will apply and on the basis of the script missing data will be added. Part of data can be added by just replacing non-terminal symbol for the specific name from the script, other data provides a simple automaton according to the specified grammar. Non-terminal symbols are written in a template as non-terminal name between two percent signs. The following code shows a template for a resource entity:

```
public class %className% extends
AbstractResource{

    %attributesDeclaration%

    /*
     * Default constructor
     */
    public %className%(int id, String
name, int amount){
        setId(id);
        setName(name);
        setAmount(amount)
    }

    /*
     * Empty constructor
     */
    public %className%(){}

    %fullConstructor%

    %attributesGetterSetter%

    public String toString(){
        return super.toString()%toString%;
    }
}
```

Generator processes the template by its division by the % character to array strings - tokens. Each token is compared with the list of keywords and if the comparison did not find any results, the token is written to the file.

If the token is recognized as a keyword, the output of automaton corresponding to that keyword is written to the file. In above template there are many non-terminals. The first of them is an attribute *%className%*, which is

overwritten by the value specified in the script as a title. Any spaces or illegal characters are removed. The attribute *%attributesDeclaration%* is responsible for the declaration of new variables. This must be performed by automaton using the following grammar:

```
%attributesDeclaration%:
  %attributesDeclaration% -> private
%attribDataType% %attribName%;%n%
%attributesDeclaration%

%attributesDeclaration% -> %e%
```

The grammar has 2 rules. Until there are other undefined variables, the first rule is used, but once all new variables from the script are processed, the second rule is used. Known data types defined at an abstract class are skipped by the generator at this stage. Non-terminal *%attribDataType%* is replaced by the appropriate data type corresponding to the attribute type in the DSL script in the section behind the element name. The element name itself is used to replace *%attribName%* attribute. Non-terminal *%n%* is used by automaton as the command new line and *%e%* is empty non-terminal, in this case the automaton ends and the generator continues processing other parameters.

Another non-terminal symbol in the template is *%fullConstructor%*, that is used to generate the constructor containing all used variables. Its structure is defined by following grammar:

```
%fullConstructor%:
  %fullConstructor% -> public %name%(int
id, String name, String
company%attributes%){
  this(id, name, company);
  %setAttributes%
}
%fullConstructor% -> %e%
```

```
%attributes%:
  %attributes% -> ,%attribDataType%
%attribName% %attributes%
%attributes% -> %e%
```

```
%setAttributes%:
  %setAttributes% ->
this.set%attribName%(%attribName%);
%n%%setAttributes%
%setAttributes% -> %e%
```

The processing of this non-terminal occurs only when the DSL script contains new attributes, otherwise the output would be identical to the default constructor. That is why this structure is implemented by using the non-terminal instead of fixed placement in the template. The first part of the template generates the general structure of the

constructor with fixed set of variables of the abstract class. In the header of the method there is non-terminal *%attributes%* which generates a list of all newly added variables including their data types separated by commas.

The body of method starts with calling the default constructor. Then *%setAttributes%* non-terminal is used for generation of the code to ensure the assignment of method's input values to particular variables.

The last non-terminal in the template is *%toString%*, which is used for completing *toString()* method. This non-terminal is performed only if the element contains new attributes.

```
%toString%:
  %toString% -> +"\n%attribName%:
"+get%attribName%()
%toString% -> %e%
```

After the template processing is complete, the output is the class containing the complete source code corresponding to the particular element in the visual interface:

```
public class Money extends
AbstractResource{
  private String currency;

  /**
   * Default constructor
   */
  public Money(int id, String name, int
amount){
    setId(id);
    setName(name);
    setAmount(amount);
  }

  /**
   * Empty constructor
   */
  public Money(){

  public Money(int id, String name, int
amount, String currency){
    this(id, name, amount);
    setCurrency(currency);
  }

  public void setCurrency(String
currency){this.currency = currency;}
  public String getCurrency(){return
this.currency;}

  public String toString(){
```

```

    return super.toString()+"\ncurrency:
"+getCurrency();
}
}

```

In a similar way templates for agents and events are created. In addition to these classes the new class is generated, which creates instances of created classes and fills them with data. By extending this class there is a space for the instance level of visual modeling or to extension the model by the simulation ability.

V. DISCUSSION ABOUT USING OF ABSTRACT CLASSES

Using abstract classes can greatly simplify the process of the source code generating, because it is not necessary to generate repeatedly general and often used structures. The disadvantage of this solution is the loss of generality of generated models. Globally, it is not possible to determine the exact structure of abstract classes of the REA ontology, because it depends on the specific modeled business process. It is only possible to determine the estimated parameters, but not mandatory parameters. For example, the agent has estimated parameter *Name*. An abstract class defines it as a String, but the model creator may require an instance of some object. Although it is possible to add a new attribute, it is necessary to choose a different label of the attribute. The question is, when it is appropriate to use an abstract class? If the usage of complex data structures in attributes is not expected, the application of abstract classes will significantly facilitate the creation of the generator. On the other hand, if generality of models and their greater modeling expressiveness is required, then the application of abstract classes is not recommended.

ACKNOWLEDGMENT

The paper is supported by the grant reference no. 6141 provide by IGA Faculty of Science University of Ostrava.

REFERENCES

- [1] M. Fowler, "Domain-specific Languages", Addison Wesley Longman, Inc., 2010, ISBN 0-321-71294-3
- [2] P. Hrubý, M. Hučka, F. Huňka, J. Kašík, D. Vymětal, "Víceúrovňové modelování podnikových procesů (Systém REA)", 2010, VŠB-TU Ostrava, ISBN 978-80-248-2334-8
- [3] P. Hruby, Model-Driven Design Using Business Patterns, Springer 2006, ISBN-13 978-3-540-30154-7
- [4] G. L. Geerts, W. E. McCarthy, "The ontological foundation of REA enterprise information systems", American Accounting Association Conference 1-34 (2000)
- [5] S. Kelly, J. Tolvanen, „Domain-Specific Modeling Enabling Full Code Generation“, John Wiley & Sons, Inc., 2008, ISBN 978-0-470-03666-2
- [6] Metacase, "Domain-specific modeling with Metaedit+: 10x faster than UML", White paper
- [7] Ch. L. Dunn, J. O. Cherrington, A. S. Hollander, "Enterprise Information Systems – A Pattern-Based Approach", 3 edition, McGraw-Hill/Irwin, ISBN-13: 978-0072404296
- [8] J. Ševčík, Z. Meliš, J. Žáček, F. Huňka, "Enterprise Information System Design Using REA Enterprise Model", Strategic Management and its Support by Information Systems, VŠB - TU Ostrava 2011, ISBN 978-80-248-2444-4
- [9] Z. Meliš, J. Žáček, J. Ševčík, F. Huňka, "Transformation of selected policy level REA concepts into a relation database", VŠB-TUO, 2011, ISBN 978-80-248-2487-1
- [10] T. Halpin, "Information Modeling and Relational Databases: From Conceptual Analysis to Logical Design", 2001, Academic Press, ISBN 1-55860-672-6

An Image Comparing-based GUI Software Testing Automation System

Hyunjun Jung, Sukhoon Lee, Doo-Kwon Baik

Department of Computer and Radio Communications Engineering, Korea University,
Seoul, Republic Of Korea

junghj85@gmail.com, leha82@korea.ac.kr, baikdk@korea.ac.kr

Abstract - This paper proposes an automated GUI (Graphical User Interface) testing method to support regression testing when a company requests new functionality additions or program changes during the development phase. In this paper, we propose an automated GUI testing method based on two processes. An event-driven testing method can test the target program directly. A capture and replay testing method can repeat recordings of the tester's actions. GUI verification is image comparing-based. To demonstrate the advantages of our proposed method, we implemented a prototype system based on C#.

Keywords: Capture & Replay test, Event-driven test, Image comparing-based test, Test automation, Test automation tool

1 Introduction

Quality assurance (QA) is very important in large scale software projects but manual testing cannot guarantee the quality of large scale programs. A company may request the addition of a new functionality or software changes during the development phase. Software changes can contain new types of errors. Thus, software changes demand repeated testing to ensure there are no errors. Software is needed for regression testing to meet the specified requirements [1]. The regression testing method uses previously executed test scenarios to check for the presence of errors. Weak version management can lead to problems with the loss of the previous bug fixes. Previously, fixing the software version solved the problem. However, temporary fixes of software often lead to the same problem when software is redesigned by refactoring. Thus, this method is not a fundamental solution. To solve the problem, software is locked up with the software version while a test scenario is created to find the bug. The test scenario should also be repeated when the software changes. QA testing proceeds by manual or automated software testing. However, this repeats a lot of regression testing when using automated testing tools, which can be expensive in terms of cost and efficiency [2]. However, quality management with automated test tools can facilitate the early detection of defects in modified software. If errors are detected before complete software development, this also has the advantage of reducing time and costs. Automated testing method include

code-driven and GUI testing. Code-driven testing methods use a class, module, or library interface and they return results that confirm whether a variety of input arguments is satisfied. The GUI testing method uses a mouse click or mouse input to generate user interface events, such as changes in a program, to ensure that the program functions correctly while observing the results. In this study, we proposed event-driven testing and capture and replay testing for automated GUI testing. The GUI also includes image comparing-based validation, such as graphs and charts. The paper is organized as follows. Section 2 provides a brief review of previous automated GUI testing tools. Section 3 describes the automated GUI testing method. Section 4 presents an implementation of a prototype system. We conclude the paper and consider future work in Section 5.

2 Relate Work

Table 1 GUI automated testing tools

Testing Tool	Feature	input	Report Function
abbot	<ul style="list-style-type: none"> Measured via a test script GUI state An interface for controlling the replay Event-based testing 	Java Application	Coverage Report
Guitar	<ul style="list-style-type: none"> Provide a test case generator plug-in Event flow measurement is useful 	Java Application	Unsupported
Pounder	<ul style="list-style-type: none"> Records test scripts and provides an interface for measuring the results 	Java Application	Unsupported
Selenium IDE	<ul style="list-style-type: none"> Records the actions of the tester using HTML script 	Web UI	Unsupported

Many studies have been conducted to support automated testing [3-11]. For example, Abbot [12] assists Java UI testing where test scripts are used to measure the state of the GUI, while event-based replay provides a control interface. GUITAR [13] is a Java and Microsoft Windows application that provides a GUI testing framework with a test case generator and event flow measurement. Pounder [14] records a test script that can be measured while the results are provided via an interface. Selenium IDE [15] records the behavior of a test script as HTML generated in the tests. Table 1 shows examples of automated GUI testing tools.

3 GUI testing automation methods

The automated GUI testing procedure is shown in Figure 1. The tester selects the appropriate method during the Method Selection phase. Special control testing is difficult, image-based buttons and user creation control are provided. In this case, the tester has selected the capture and replay method. The event-driven method sends the event directly to the target program, depending on whether scenario testing is performed by sending an event. In addition, a scenario can be created using the event to perform tests rapidly and without error. The exact value of the unit tested is checked before verifying whether an event-driven method is required.

Method selection during this stage of the target application development requires a unit test or verification of the exact value when it is necessary to initiate the event. Figure 2 s shows the use of the event-driven method. The event-driven method can be used to send events directly to the target program to perform precision tests of the control value of the property involved in the event, by comparing the verification accurately. This would be available at the code level with other methods for scripting an event in a direct target program. An event-driven method is required to perform tests using the event name and to create a test script for the operation. Test scripts are written in scripting languages, while parsing the script will create the event. The event script is a list of events that are passed to the actual event. All events are dispatched to the target program, before ending the test.

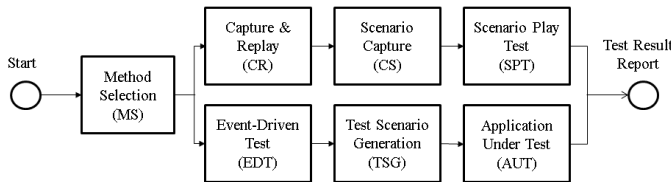


Figure 1 Automated GUI testing process

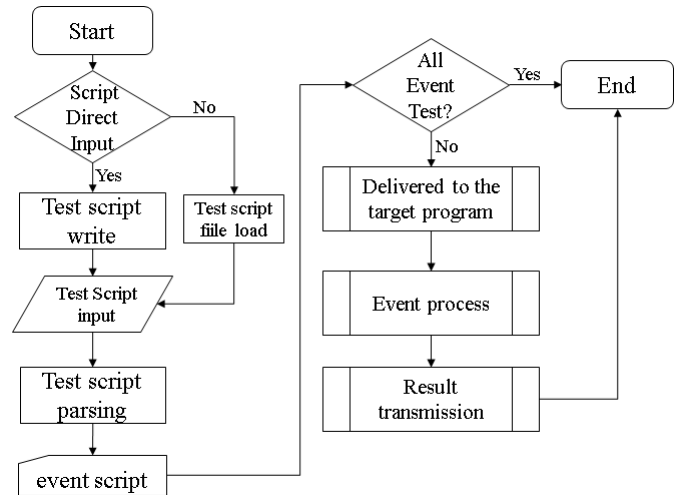


Figure 2 Process of the event-driven method

During the capture and replay method selection step, as shown in Figure 3, the tester selects how the test is performed throughout the process. The test script is written by hand or previously written scripts are reused where available. This method is based on the coordinates of the target program features when performing a test of added or changed events. In this case, you the previous scenario can be reused. Test scenarios keyboard strokes and mouse movements can be used to automatically generate scripts by recording. Replaying the generated script reproduces the movement of the testers, which alters the program or input data so the test can be performed.

The capture and replay method does not communicate directly with the event in the target program. Targeted programs are recognized based on their coordinates. A comparison of the output image should verify the correct value. A comparison of images is shown in Figure 4 using the verification procedure. The tester extracts and generates a script to control the capture program, which is targeted directly in the middle of the verification process for features that require an output value from the GUI and the image.

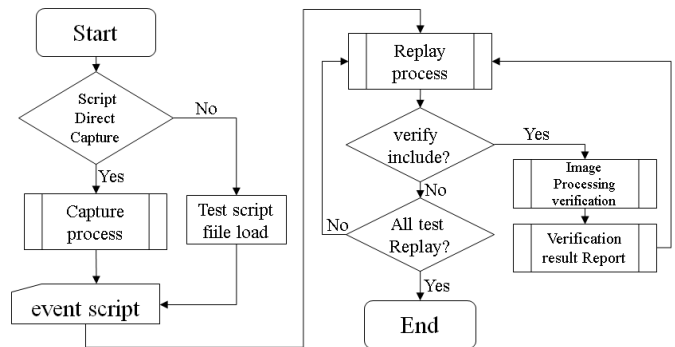


Figure 3 Event-driven method process

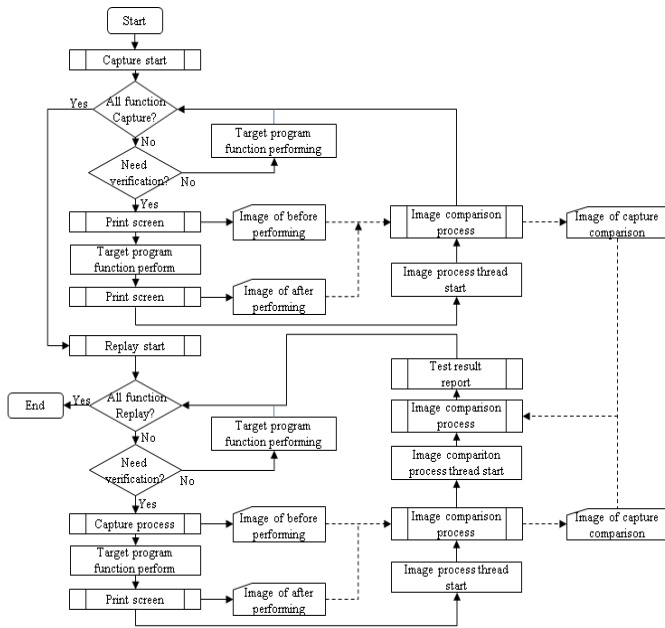


Figure 4 Verification method based on image comparing

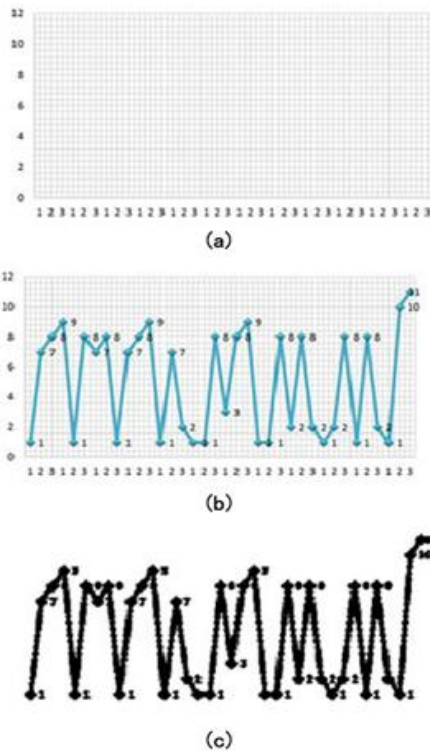


Figure 5 Example of image comparing

The necessary features are implemented for each verification result before comparing the differences in the output between the images on the screen and the previous screen output, which creates an image that is recorded.

The image shown in Figure 5 is generated by comparing the images. Figure (a) is the previous screen output results on the GUI. Figure (b) shows a GUI screen with the results for the test output. Replaying the created image facilitates testing of the changed GUI and a change in the target program will be detected as an error. Figure (c) shows a comparison of the image before and after performing the image testing. The result of performing the functions on the replay screen during testing are compared with the extracted features to determine whether they perform correctly.

4 An Image Comparing-based Automated Software Testing System

4.1 Conceptual Model

Figure 6 shows the concept model of our event-driven method. The tester extracts information to perform a test scenario by parsing the XML-based test. The event generator produces an event script based on this information. A saved script file can be reused if necessary for future testing of the same units as those found in the target program. An event testing engine script is generated for events and passed to the target program to perform the test. Verifying the results of the control target program for events yields the value of the property.

The event-driven method can be verified directly to yield the property values of controls, which is advantageous for verifying the exact value. The XML tester must specify the expected results to verify the values directly. The test result is deduced from the tests performed by the function generator, including successful and failed test results.

Figure 7 shows a conceptual model of the capture and replay method. The capture engine generates test scripts according to the tester while entering the time for recording mouse and keyboard inputs automatically. The tester input is activated when the target program is generated. The capture engine uses Global Hooker global event analysis of the OS to generate the script. This function is necessary for the extracted image in the target program verification system to activate the image verification engine accelerator to deliver the results. Replayed functions during image performance are passed through a procedure to test the capacity to compare the output of the identified portions and to verify the test results that are based on it.

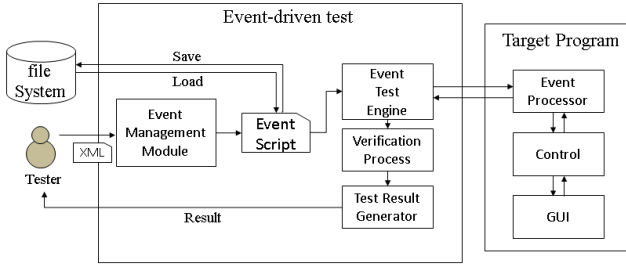


Figure 6 Conceptual model of the event-driven method

4.2 Test Case

Figure 8 shows an example implementation of a target program when applying the automated test tool. The path is typed in the target program to enumerate the files by selecting the check box, which can be deleted. Menus, list boxes, and edit boxes are common features in the GUI.

Events in the target program are tested using the event-driven Method, with the tester shown in Figure 9 to transfer the event to XML-based scripts. The root node of an XML-based script in the <test> center node of each test is divided into a function <step> to perform any function that indicates whether the value was passed as an argument.

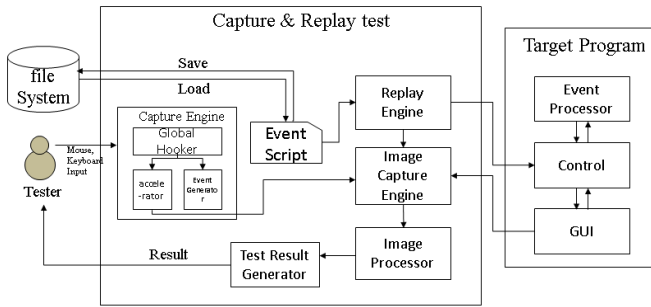


Figure 7 Conceptual model of the capture and replay method

Figure 10 shows the main screenshot of the event-driven method. In this stage, the tester has loaded XML-based scripting of the test step to select the requirements. The step chosen by the tester moves to the automated GUI test tool, which is passed through a targeted program of events. The tester includes a previously created XML-based script step scenario for mouse and keyboard inputs. The automated GUI test tool passes sequentially through each step of the scenario at the rate set by the tester.

The capture and replay method is shown in Figure 11 where the number and speed of the repeat can be set. The capture start button is used to record the movements of the tester. When the capture the tester is finished, the Stop button is pressed to create a script and test the tool automatically when it is ready to Replay. Capture and replay is performed using a script, which can be reused.

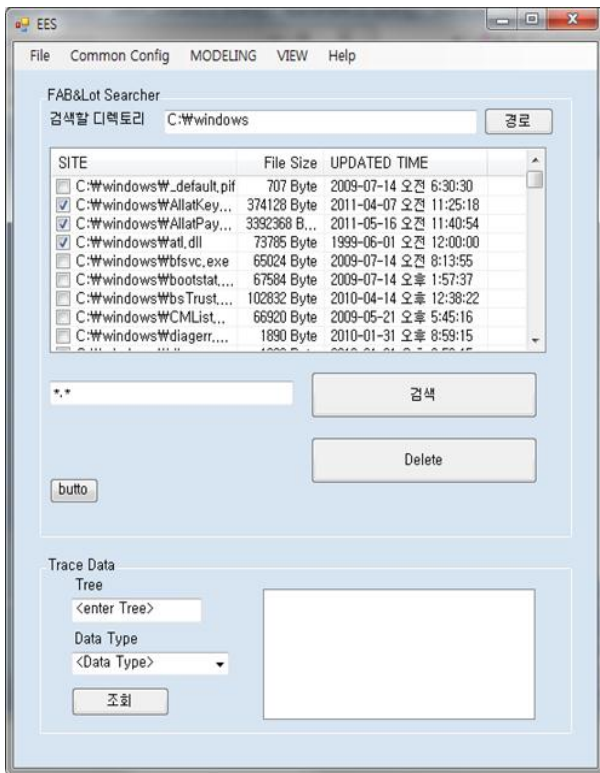


Figure 8 Example of a test target program

```

<?xml version="1.0" encoding="utf-8"?>
<test>
  <step>
    <comment>menu->View->Trace Data</comment>
    <type>click</type>
    <content>Trace Data</content>
    <eventName>listViewToolStripMenuItem_Click</eventName>
  </step>
  <step>
    <comment>TREE select</comment>
    <type>text</type>
    <content>tree selection</content>
    <eventName>textBox1</eventName>
  </step>
  <step>
    <comment>Data Type selection</comment>
    <type>text</type>
    <content>User Define</content>
    <eventName>comboBox1</eventName>
  </step>
  <step>
    <comment>Seach Button</comment>
    <type>click</type>
    <content>seach</content>
    <eventName>button2_Click_1</eventName>
  </step>
  ....
</test>
    
```

Figure 9 Example of a written script

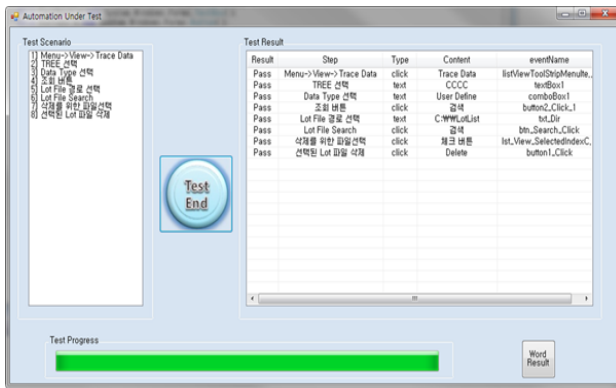


Figure 10 Event-driven testing

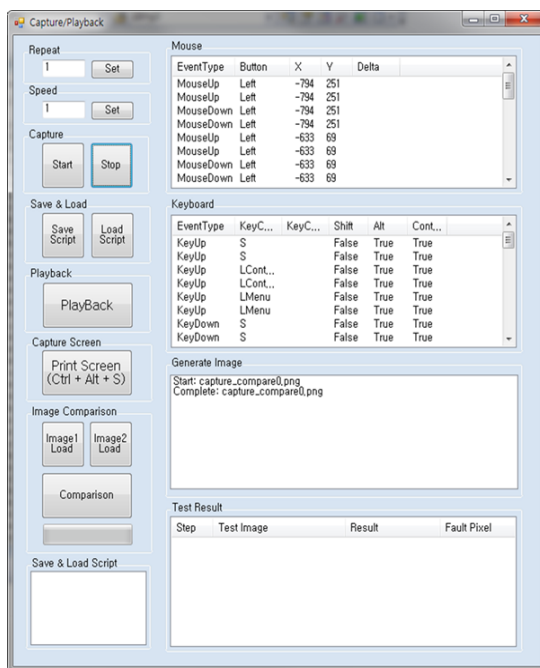


Figure 11 Capture and replay testing

5 Conclusions

This paper proposed an Automated GUI Testing Method to support regression testing. The GUI provides image comparing-based validation using graphs and charts. The automated test method uses two processes. The event-driven method generates an event that tests the target software. The capture and replay method records the action, which needs to be repeated. We developed the test method, which was defined for this prototype system. Using the proposed method, automated testing is possible without knowing the source code. Further comparisons of the proposed automated test method and other commercial tools are required. The test method also requires simpler scripting to improve the efficiency of the input method.

6 Acknowledgement

This study was supported by Second Brain Korea 21 Project and by the National IT Industry Promotion Agency (NIPA) under the program of Software Engineering Technologies Development. The corresponding author is Doo-Kwon Baik.

7 References

- [1] A. M. Memon and M. L. Soffa. "Regression testing of GUIs", In ESEC/FSE-11: Proceedings of the 9th European Software Engineering Conf/11th ACM SIGSOFT Int'l Symposium on Foundations of Software Engineering, pages 118-127, 2003
- [2] B. A. Myers. "User interface software tools", ACM Trans. Comput.-Hum. Interact., 2(1):64-103, 1995
- [3] E. Dustin, J. Rashka, and J. Paul. "Automated Software Testing", Addison-Wesley Publishing Company, 1999
- [4] A. Memon, I. Banerjee, A. Nagarajan. "GUI ripping: reverse engineering of graphical user interfaces for testing", In Reverse Engineering 2003 WCRE Proceeding 10th Working Conference on, pages 260-269, 2003
- [5] T.Ostrand, A. Anodide, H. Foster, T. Goradia. "A visual test development environment for GUI systems", SIGSOFT Softw. Eng. Notes, 23(2):82-92, 1998
- [6] Q. Xie, A. M. Memon. "Designing and comparing automated test oracles for GUI-based software applications", ACM Trans. Softw. Eng. Methodol., 16(1):4, 2007
- [7] T. Yeh, T. H. Chang, R. C. Miller. "Sikuli: Using GUI screenshots for search and automation", In UIST '09, pages 183-192, 2009
- [8] A. Kervinen, M. Maunumaa, T. Pääkkönen, M. Katara. "Model-based testing through a gui", In Formal Approaches to Software Testing, volume 3997, pages 16-31, 2006
- [9] M. Finsterwalder. "Automating acceptance tests for GUI applications in an extreme programming environment", In Proc. Second Int'l Conf. eXtreme Programming and Flexible Processes in Software Eng., pages 114-117, 2001
- [10] J. Steven, P. Chandra, B. Fleck, A. Podgurski. "jRapture: A capture/replay tool for observation-based testing", In Proc. Of the 2000 ACM SIGSOFT Int'l Symposium on Software Testing and Analysis, pages 158-167 ACM Press, 2003
- [11] L. White, H. AlMezen, N. Alzeidi. "User-based testing of GUI sequences and their interactions", In Proc. 12th Int'l Symposium on Software Reliability Engineering, pages 54-63, 2001
- [12] Abbot. Abbot framework for automated testing of JAVA GUI components and programs, <http://abbot.sourceforge.net/doc/overview.shtml>
- [13] Guitar. A GUI testing framework, <http://guitar.sourceforge.net/index.shtml>.
- [14] Pounder. Java GUI Testing Utility, <http://pounder.sourceforge.net/> Selenium IDE. Web application testing tool, <http://seleniumhq.org/projects/ide/>

A Usability Evaluation Process for Plastic User Interface Generated With an MDE Approach

Lassaad Ben Ammar, Adel Mahfoudhi, and Mohamed Abid

CES Laboratory, University of Sfax, Sfax, Tunisia
National Engineering School of Sfax

Abstract—*The challenge of developing a practical plastic User Interface (UI) has been the subject of several recent studies. Usability evaluation method and technique consider usability only at the last stage of the development process. At this stage, solving the usability problems is a very difficult and expensive task. Actually, we wish to integrate the usability evaluation into a plastic UI development process that follows the principles of the MDE. To do so, we propose a usability evaluation process which is aligned with the ISO/IEC 25000 standard. In this paper, our utmost objective is to show how our usability evaluation process can improve usability. A case study was performed in order to reach such objective.*

Keywords: Usability Evaluation Process, ISO/IEC 25000, Plastic User Interface

1. Introduction

The Usability of Plastic User Interface (PUI) is a crucial factor since the plasticity of a User Interface (UI) denotes its ability to withstand the context variation while preserving usability. The functional aspect of UI adaptation is the well-developed part of the plasticity. Nevertheless, usability is a key factor that determines the success or the failure of a UI. The need for a usability evaluation method (process) that is crafted for PUI has become critical since the PUI are increasing their importance in industrial domains. Typically, usability evaluation is considered at the last stages. Intermediate artifacts are used to guide developers and document the application. Due to the lack of the well-defined traceability between intermediate artifact and final product, the correction of the Usability Problems (UPs) is usually a difficult and expensive task. This problem can be alleviated in an MDE development process. We argue that performing the evaluation at each stage of the development process is a critical part of ensuring the effectiveness of the PUI for their purpose. For that reason, we are opting for making use of an MDE-compliant method for generating usable PUI. Such methods basically transform models that are independent of implementation details (Platform Independent Model - PIM) into others that contain specific details from the platform (Platform Specific Model - PSM). The source code of the

UI is automatically generated by transforming the PSM. So, the aim of this paper is to propose a Usability Evaluation Process (UEP) that can be integrated into an MDE-compliant method in order to generate a usable PUI. Our proposal is intended to perform usability evaluation at each stage of the development process. Such operation provides feedback that can be used to suggest changes in the evaluated artifact or in his predecessor, at some extent, in order to improve usability of the generated PUI.

The remainder of this paper is structured as follows. While section 2 presents an outline of the usability evaluation process quoted in the literature, section 3 describes the proposed usability evaluation process. Besides, section 4 provides a case study illustrating the integration of the usability evaluation process into an MDE-compliant method for generating PUI. Finally, section 5 draws the conclusion and provides perspectives and future research work.

2. State of the art

Several usability evaluation methods have been proposed over the last years. The most important part of these methods have been defined building on existing standards such as ISO 25000.

The ISO/IEC 25000 standard aims to provide a guideline for requirements specification, measurement and evaluation. The evaluation process can be performed for the intermediate artifact or the final product. It consists of five phases: establish evaluation requirements, specify the evaluation, design the evaluation, execute the evaluation, and conclude the evaluation.

In the first phase, the evaluation requirements such as the purpose of the evaluation or the identification of the product parts to be evaluated are established. The quality model of ISO/IEC 9126-1 standard is used as the basis for the requirements specification. During the second phase, suitable measurements for each attribute are identified. The decision criteria for the metrics as well as the assessment criteria are defined. In the third phase, evaluation and evaluator action are scheduled in an evaluation plan. The evaluation phase consists of the following activities: applying measurement, applying rating, and assessment. At the last stage of the

Table 1: Usability Evaluation Process ISO/IEC 25000.

Phase	Activity
Establish evaluation requirements	<ul style="list-style-type: none"> - Establish the purpose of the evaluation. - Obtain the software product quality requirements. - Identify product parts to be included in the evaluation.
Specify the evaluation	<ul style="list-style-type: none"> - Define the stringency of the evaluation. - Select measures. - Define decision criteria for measurements. - Define decision criteria for evaluation.
Design the evaluation	<ul style="list-style-type: none"> - Plan evaluation activities.
Execute the evaluation	<ul style="list-style-type: none"> - Make the measurements. - Apply decision criteria for measures. - Apply decision criteria for evaluation.
Conclude the evaluation	<ul style="list-style-type: none"> - Review the evaluation results. - Dispose evaluation data.

process, evaluators determine to which extent the software product meets the quality requirements.

Based on this standard, [1] propose a usability evaluation process for web application generated with an MDD-compliant method (WUEP). The aim of this work is to show the feasibility of the WUEP in discovering the UPs. The accuracy of this work is the main question since the analysis of changes step was not developed. The number of evaluators (seven) is also more than the recommended for a usability inspection method (three to five). The use of a non representative user; two web usability experts for the evaluation designer role and four web usability experts in the evaluator role, has led the authors to speak about the limitation of their proposal.

[2] define an evaluation process in four phases: quality requirement definition and specification, elementary evaluation, global evaluation and conclusion. In the first step, evaluators should clearly identify the evaluation goals, the users need and the requirement tree based on the ISO/IEC 9126-1 quality model. Next, metrics were associated with each attribute from the requirement tree. An elementary criterion function was defined for each metric to interpret the measured values. In the third step, evaluators select the aggregation function of the overall elementary quality preference obtained in the last stage. The global quality preference represents the degree to which the software product meets the stated requirements. Finally, evaluators analyze the assessed product with regard to the established goals and users need, and suggest some recommendations to improve the global product usability. The main limitation of this work is that it quantitatively assesses web applications. Besides, it requires the final product to do the evaluation and does not define the context in which UI will be executed. The WebTango methodology presented in [3] addresses the web usability issues. In fact, it assesses web sites by

comparing them to well-designed interfaces. The evaluation process starts with identifying an exhaustive set of quantitative interface measures to assess as many aspects of web interfaces as possible. After that, measures will be computed for a large scale of rated interfaces. Then, a statistical model will be derived from the measurements and ratings. This model will be used to predict ratings for new interfaces, and finally validating the model prediction. Like the other methods presented in this section, this method has some limitations. Indeed, it is based only on objective measurements that make difficult to capture the subjective preferences of the user. It should be susceptible to changes in the web site content (improving measures and techniques to address the current state of the web). The assessment of the web site is based on comparing it to well-designed web sites. Such web sites are well designed in a well-defined context that may not be the same as that of the interface in question. Such assessment need final product to be done.

After reviewing several UEP, we have noted that the mentioned UEP present some limitations. Next, we will evaluate the aforementioned UEP in the basis of some criteria that have been extracted from several works ([4], [5],...). According to [4] defining the *context* in which the product will be used as a key factor that affects the accuracy of a UEP. It is important, when evaluating the usability, that the conditions for the test are representative of the important aspect of the overall context of use. The *type of the evaluator* is also an important factor that affects the accuracy of an UEP ([5]). Nielsen said that it is more appropriate to use a system developer as an evaluator since it will focus on the aspects of the UI that an expert would not. We argue that the means of *improvement* provided by a UEP is a key factor. In fact, the ultimate goal of an evaluation process is to ensure that the software product meets the requirements stated in the specification stages. After detecting UPs, an evaluation process should suggest changes to improve the product. The *type of the product* to be evaluated (intermediate, final) can be an evaluation criterion since the objective of recent works, from the present work is part, is to integrate this activity at all stages of the development process. It is important for a UEP to consider intermediate artifact when evaluating usability. Other criteria can be used for the evaluation UEP such as the *group/individual* evaluation. According to [5], it is more effective to perform a usability method as a group than with individual evaluator.

To the best of our knowledge, there is no generic process that integrates usability evaluation into an MDE-compliant method for PUI generation. As it is shown above, the majority part of works that tackles usability issues, although they contain almost the activities that can occur in a usability evaluation process, they suffer from some drawbacks. We

Table 2: Evaluation of some Usability Evaluation Method.

Method	[2]	[3]	[1]
Context	Limited	No	Limited
Evaluator Type	Expert	Expert	Expert
Improvement	No	No	No
Product Type	Final	Final	Intermediate
Grp/Ind	Individual	Individual	Group

will use them as sources that can be drawn from our proposal. The ultimate objective of this paper is, actually, to propose a UEP that fills these gaps and that aims to be used as a guideline for generating a usable PUI. The section that follows presents our proposed UEP for such a purpose.

3. Proposed Usability Evaluation Process

Figure 1 presents an overview of our proposal UEP. In fact, the proposed UEP adapt and extend the ISO/IEC 25000 standard. The following sub-sections describe each activity of the proposed UEP.

3.1 Establish the Purpose of Evaluation

The aim of this activity is to clearly identify the purpose of the evaluation and to ensure that the PUI provides the required usability that meets the users' needs. Since the usability of PUI can be evaluated at all stages of the development process, several goals can be identified. A list of typical purposes is presented below:

- Identify specific UPs;
- Improve PUI performance;
- Predict or estimate final PUI usability;
- Decide on the acceptance of the PUI.

3.2 Select Usability Model

An agreed usability model is used for structuring the usability requirements. The usability model is based on the ISO/IEC 9126-1 standard. It considers the usability from a software product view that allows us to determine the usability of the intermediate artifact in the PUI development process.

3.3 Identify Product Part to be Evaluated

The Usability of a PUI generated by an MDE-compliant method can be evaluated at several stages of the development process: in the PIM by evaluating the Abstract User Interface Model (AUIModel), in the PSM by evaluating the Concrete User Interface Model (CUIModel), in the CM by evaluating the Final User Interface. The product parts to be evaluated depending on the purpose of the evaluation.

3.4 Identify Context of Measurement

The Usability of PUI depends on the context of use in which it will be used. Usability is the quality of use in a context [4]. The context of use is defined by the user, task and environment. The selected context when evaluating the usability should be representative of the important aspect of the actual or the intended context of use.

3.5 Select the Usability Attributes

The selected usability model is used when selecting the attributes that will be considered in the usability evaluation stage. This model is based on the ISO/IEC 9126-1 standard which decomposes the usability into five sub-characteristics: learnability, understandability, operability, attractiveness, compliance. As already mentioned, we only focus on the usability from a software product point of view since it can be evaluated during the development process by inspecting the intermediate artifact.

3.6 Select the Metrics to be Evaluated

With each selected attribute, we associate at least one metric in order to quantify them. The selection metrics activity's must verify the following condition: choosing the minimum number of metrics that reveal the maximum amount of the usability detail for the PUI under study [3]. We propose two types of metrics: objective and subjective since we argue that a robust UEP must aggregate both subjective and objective measurements.

3.7 Establish Rating Levels

The values obtained from the already selected metrics have no meaning since they are numerical values obtained using a calculation formula. These numerical values must be interpreted. We propose to assign a categorical value for the ranges of values obtained for each metrics. Such mechanism is climbed by [6]. The values obtained from the metrics and the rating level established for these values allow us to interpret the degree to which the selected attributes contribute to achieving a usable PUI.

3.8 Define an Aggregation Function

The aim of this activity is to prepare a procedure for the further summary of the attributes and the sub-characteristics. We propose an adaptation of the aggregation function proposed in [7]. We will present such adaptation when developing the case study section.

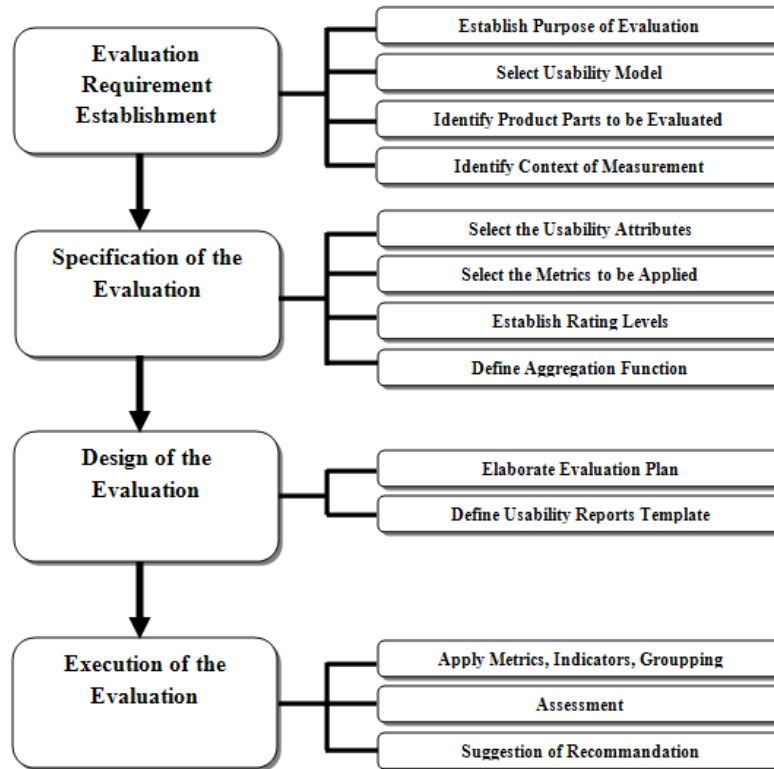


Figure 1: The proposal Usability Evaluation Process.

3.9 Elaborate Evaluation Plan

This activity includes the establishment of the order in which the artifact will be evaluated, the number and type of evaluators and assigning tasks to evaluators. According to [8] the recommended order for artifact evaluation is PIM since this artifact drives the development process. We can then evaluate the PSM and CM artifact. As we mention above, the recommended number of evaluators for an inspection method for usability evaluation is usually three to five.

3.10 Define the Usability Reports Template

This activity defines the template of the usability report that is commonly a list of the UPs detected in the evaluation. We propose the fields presented in Table 2 to describe each UPs.

3.11 Apply Metrics, Indicators and Grouping

The selected metrics should be applied to the product parts already selected according to the evaluation plan. The indicators should be applied in order to identify the artifacts' elements that contribute to the UP. The outcome of this stage is a usability report that collects the UP detected when evaluating the selected artifact. This usability report should clearly specify the source of the UP. After grouping the

Table 3: Usability Report Template.

Fields	Meaning
ID	An identifier of the UP.
Description	A description of the UP.
Related attributes	The attributes from the usability model that is related to the UP.
Level	The categorical value assigned to the numerical values obtained by the evaluation activity.
Recommendation	The suggested changes in order to improve the extent to which contributes the elements which is the source of the UP.

elementary categorical values, the usability of the overall artifact is identified.

3.12 Assessment

The obtained categorical value indicates the extent to which the PUI meets the usability requirements. The evaluator should decide to accept the version of the PUI or to reject it.

3.13 Suggestion of improvement

In case of rejection and based on the usability reports, evaluators should suggest changes in order to improve the

selected artifact that was evaluated. The outcome of this stage is an improvement reports. After applying the suggested changes, it is necessary to re-evaluate the artifact.

4. Instantiation of the proposal Usability Evaluation Process

In order to well structure this section, we will follow the guidelines of the presentation of the case study presented in [9]. In such guidelines, case study is comprised by the following stages: design, preparation, data collection, data analysis.

4.1 Case study design

It is necessary when designing the case study to consider some components. For reason of simplicity, we will consider here only two components: the purpose of study and the research questions.

The purpose of the case study is to show the feasibility of applying our proposal UEP to improve the usability of a PUI that was generated by an MDE-compliant method. The research questions that are addressed by the case study are:

- Does the proposed Usability Evaluation Process contribute to discover sevrals UPs at severals artifacts?
- What are the implications of the detected usability problem for the intermediate artifacts?

We are opted to the conceptual framework proposed in [10]. Such a method identifies four stages in the development process: task model, Abstract UI model, Concrete UI model, Final Model. The object of the evaluation is an Ask for a Credit Card Application. The following scenario illustrates the interaction. the customer is connected to the site of the bank to launch her request of credit card. He has to log in first of all by introducing her user name and password. Then he has to choose her type (private individual or company). Then, he is asked to choose the type of card that she seeks to obtain (Visa, Master Card, etc.) before filling in an information form.

The subjects are all the authors. The preparation stages (establish evaluation requirement, specify evaluation, design the evaluation) are performed by two of the authors while the execution of the evaluation stages are performed by all the authors.

4.2 Preparation of the case study

With regard to the *Evaluation Requirement Establishment* stages of our proposal UEP, the purpose of the evaluation was to identify specific UPs. The usability model that will be used to structure the requirement is an adaptation of the ISO/IEC 9126-1 model. The artifacts to be evaluated were abstract user interface model and the final user interface model. The context of use of the system is the customer

service in a bank agency. Two kinds of users will use the system: the manager of the service and the customer.

As for the *Specification of the Evaluation* stages of our proposal UEP, a set of 6 attributes were selected from the usability model. When selecting the attributes, we consider only those which would be more relevant to the application type and the context in which it is going to be used. For each selected attributes, we identify at least one metrics in order to quantify them. Metrics are then associated with the selected artifacts in which they could be applied. Table 4 and 5 present respectively some calculation formula and the rating level associated to the selected metrics, respectively. The numerical range defined for each metrics has been built from some usability guidelines and heuristics described in the literature, as [6] and [11].

Table 4: Example of metrics.

Name	Number of Font Style Used
Attribute	Font Style Uniformity
Description	The number of font style used in a UI.
Scale type	numerical value
Interpretation	The larger the value of the metrics, the less attractive is the UI.
Artifact	At the PIM and the CM.

Table 5: Examplpes of indicators.

Metrics	VG	G	M	B	VB
TL	<2	$2 \leq TL \leq 3$	$3 < TL \leq 5$	$5 < TL \leq 7$	>7
NFSU	<2	$2 \leq NFSU < 3$	$3 \leq NFSU < 4$	$4 \leq NFSU < 5$	>5
WN	<15	$15 \leq WN \leq 20$	$20 < WN \leq 25$	$25 < WN \leq 30$	>30
MA	<2	$2 \leq MA < 4$	$4 \leq MA < 6$	$6 \leq MA < 8$	>8

The aggregation function that we opted for is an adaptation from that proposed in [7]. Such function aggregates two by two the qualitative values applying a set of rules. The adaptation affect the following rules:

- Combination (G,VB) \rightarrow VB we suggest M
- Combination (B,B) \rightarrow VB we suggest B
- Combination (G,G) \rightarrow VG we suggest G
- Combination (VG,B) \rightarrow VG we suggest M

when adapting these rules we are considering the following hypothesis:

- All metrics that make up attributes, all attributes that make up sub-characteristics and all sub-characteristics that make up usability have the same impact (relative weight).
- Each metrics, each attribute or each sub-characteristics has a positive or a negative effect on the final level of usability.

With regard to the *design of the Evaluation* stage, the evaluation plan was elaborated taking into account the following

order of artifact to be evaluated: PIM and then CM. Three evaluators were selected to perform the execution stage of our proposal. In fact, this number was selected taking into consideration some recommendations (e.g. [12]) which claim that 3 to 5 evaluator can release the most part of UPs. The *Evaluation Report Template* was defined by considering the same fields proposed in the previous section.

4.3 Data collection

Data was collected during the *Execution of the evaluation* stage of our proposal UEP. Metrics are applied to the excerpts of model that are shown in Figure 2. In the PIM model (Abstract User Interface Model AUIM), the value of *Title Length* is 3 (title of the window). The categorical value attributed to this numerical value is G. This leads to the conclusion that the title of each window in the PIM artifact does not exceed the value 3 in order to obtain at least a categorical value equal to G. In the CM model (Final User Interface Model FUIM), the value of the *Number of Font Style Used* (NFSU) is 4. The rating level of this metrics indicates the existence of a Bad UP (UP01). Table 6 shows an excerpts of the usability report that is presented in the UP01.

Table 6: Usability problem UP01.

Fields	Meaning
ID	UP01.
Description	The CM model use four font style: Calibri, Georgia, Arial and Verdana.
Related attributes	Font Style Uniformity.
Level	Bad (four different font style are used).
Recommendation	change the font style property of the captions of the buttons in the abstract UI model. The font style property should be defined at the PIM. We may use a statistic variable that stocks the number of different font style used in each window.

4.4 Analysis of data

The Usability Evaluation Process proposed in this paper contribute in discovering several UPs in several artifacts employed during the early stages of the MDE-compliant method. **Implication of the usability problem for the intermediate artifact.**

Concerning the *Abstract User Interface Model*, the detected problem related to **operability** sub-characteristics can be solved by correcting the AUI meta Model or the transformation rule. In order to increase the attributes that contribute to the **Operability** sub-characteristics (*User Operation Cancellability*, *User Operation Undoability*, *Explicit User Action*), we suggest to add to each UIUnit a Back button, a Cancel button and a Validate button.

Some guidelines recommend a number of navigation that

should not be exceeded. To solve the detected problem related to the *Navigation* attributes, we suggest a statistic variable in each window that stocks the number of a navigation element. Such changes affect the propriety of UIUnit in the AUI Meta Model.

Regarding the *Final User Interface Model*, the detected problems related to the **attractiveness** sub-characteristics can be solved by correcting the AUI Meta Model. We can add some statistic variables defining the number of each metrics contributing to this sub-characteristics through a number (e.g. number of font style used, number of font size used, number of colors used, ...).

We argue that the case study has been useful for us since it allow us to learn more about the benefits and limitations of our proposal UEP and how it can be improved to achieve its purpose.

Several types of problems can be detected using our proposal. These problems are related to several artifacts during the development process. The evaluation process may be a means to discover the limitation in the expressiveness of the artifacts of any MDD approach. The performed changes at several artifacts and the traceability between artifacts allow us to talk about usability that can be achieved by construction [13], at least to some extent.

Our proposal can be applied in another MDD process. We have just to mapp between the metrics and the model elements from the other MDD process. Besides, our proposal can be judged as an accurate process since it takes into account several recommendations like the number of evaluators, the type of evaluators, the evaluation as a group. Such recommendations are judged by the experts as a key factor in the success of any usability evaluation method.

5. Conclusion and Further Work

This paper has presented a Usability Evaluation Process for Plastic UI (UEPP) that integrates usability evaluation during several stages of an MDE-compliant method. The proposal process adapt and extend the ISO/IEC 25000 standard. The UPs detected allow us to extract the expressiveness of the artifacts. The changes performed after evaluation can affect models or the transformation rules. Due to the traceability between models, the usability of an automatically generated UI can be predicted. We are referring to a UI that can be usable by construction [13]. This paper argues the following claim : taking into account usability through the entire development process enables the development of a UI that has better usability. Such mechanism also contributes to the effort reduction at the maintenance stage.

Further work are intended to : propose a usability model that is crafted for plastic user interface; tackle the problem of the aggregation function; study the possibility of automating the

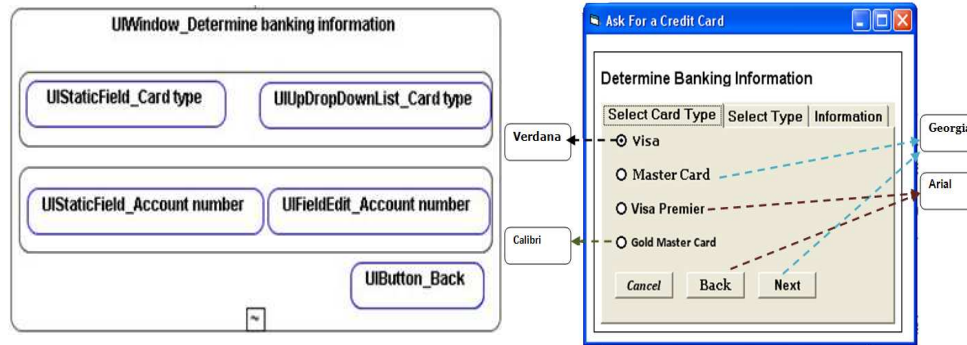


Figure 2: Excerpts of model from PIM and CM artifacts.

large part of our proposal UEP.

References

- [1] A. Fernandez, S. Abrahão, and E. Insfraán, "A web usability evaluation process for model-driven web development," in *CAiSE*, 2011, pp. 108–122.
- [2] L. Olsina and G. Rossi, "Measuring web application quality with webqem," *IEEE MultiMedia*, vol. 9, no. 4, pp. 20–29, 2002.
- [3] M. Y. Ivory and R. Megraw, "Evolution of web site design patterns," *ACM Trans. Inf. Syst.*, vol. 23, pp. 463–497, October 2005. [Online]. Available: <http://doi.acm.org/10.1145/1095872.1095876>
- [4] N. Bevan and M. Macleod, "Usability measurement in context," *Behaviour and Information Technology*, vol. 13, pp. 132–145, 1994.
- [5] H. W. Desurvire, "Usability inspection methods," J. Nielsen and R. L. Mack, Eds. New York, NY, USA: John Wiley & Sons, Inc., 1994, ch. Faster, cheaper!! Are usability inspection methods as effective as empirical testing?, pp. 173–202. [Online]. Available: <http://dl.acm.org/citation.cfm?id=189200.189217>
- [6] J. I. Panach, N. Condori-Fernández, T. E. J. Vos, N. Aquino, and F. Valverde, "Early usability measurement in model-driven development: Definition and empirical evaluation," *International Journal of Software Engineering and Knowledge Engineering*, vol. 21, no. 3, pp. 339–365, 2011.
- [7] J. I. Panach, N. Condori-Fernández, F. Valverde, N. Aquino, and O. Pastor, "Software process and product measurement," J. J. Cuadrado-Gallego, R. Braungarten, R. R. Dumke, and A. Abran, Eds. Berlin, Heidelberg: Springer-Verlag, 2008, ch. Towards an Early Usability Evaluation for Web Applications, pp. 32–45.
- [8] J. I. Panach, N. Condori-Fernández, T. E. J. Vos, N. Aquino, and F. Valverde, "Early usability measurement in model-driven development: Definition and empirical evaluation," *International Journal of Software Engineering and Knowledge Engineering*, vol. 21, no. 3, pp. 339–365, 2011.
- [9] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," *Empirical Softw. Engg.*, vol. 14, no. 2, pp. 131–164, Apr. 2009. [Online]. Available: <http://dx.doi.org/10.1007/s10664-008-9102-8>
- [10] W. Bouchelligua, A. Mahfoudhi, N. Mezhoudi, O. Dâassi, and M. Abed, "User interfaces modelling of workflow information systems," in *EOMAS*, 2010, pp. 143–163.
- [11] J. I. Panach, N. Condori-Fernández, F. Valverde, N. Aquino, and O. Pastor, "Understandability measurement in an early usability evaluation for model-driven development: an empirical study," in *Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement*, ser. ESEM '08. New York, NY, USA: ACM, 2008, pp. 354–356. [Online]. Available: <http://doi.acm.org/10.1145/1414004.1414080>
- [12] N. Bevan, C. Barnum, G. Cockton, J. Nielsen, J. Spool, and D. Wixom, "The "magic number 5": is it enough for web testing?" in *CHI '03 extended abstracts on Human factors in computing systems*, ser. CHI EA '03. New York, NY, USA: ACM, 2003, pp. 698–699. [Online]. Available: <http://doi.acm.org/10.1145/765891.765936>
- [13] E. Iborra, J. V. S. Abrahão, and S. Abrahão, "Usability evaluation of user interfaces gener with a model-driven architecture tool. chapter 2," in *Maturing Usability: Quality in Software, Interaction and Value. HCI Series*. Springer-Verlag, 2007.

SES-based Structure Modeling Method Using the Block Diagram

Suk-hoon Shin¹, Chan-ho Jung¹, Eun-bog Lee¹, Sung-do Chi¹, Seung-jin Han²

¹Graduate School of Computer Engineering, Korea Aerospace University, Goyang-si, Gyeonggi-do, Korea

²Agency for Defense Development, Changwon-si, Gyeongsangnam-do, Korea

Abstract - SES(System Entity Structure) is the well-known structure representation formalism for describing the decomposition and/or taxonomy of the system. It also provides several convenient operations for building the structure model. However, the conventional SES modeling system does not utilize the module diagram which is easy and widely used graphical representation method. To deal with this, we have proposed the advanced structure modeling method by combining the SES structure model with the block diagram representation. The paper will demonstrate how each properties and operations of the SES may be accomplished by using the block diagram.

Keywords: SES, structure modeling, block diagram, modeling tool, system design

1 Introduction

The structural modeling describes the architectural feature of the target system. It refers to define the connection relation between that it is the architectural feature comprising the system hierarchical relation of the sub-model and each sub-model, and etc. The structural modeling modularizes the complex system by the functional and is the administration and revision easy, and the reusability of the module is enhanced.

In the 'Matlab' or 'AnyLogic', and etc., that is the existing system modeling and simulation tool, the typically structural modeling feature is provided. That is the function of the extent which defines the hierarchy structure of the modelling object and connection relation and which it modifies with the reuse of the sub-model[1,2].

The SES is a representation scheme that contains the decomposition, coupling and taxonomy information of a system. The SES contains entities and three types (aspect, specialization and multiple-decomposition) of nodes. One application of the SES framework relates to the design of a system. Here the SES serves as a compact knowledge representation scheme of organizing and generating the possible configurations of a system to be designed. To generation a candidate design, we can use pruning that reduces the SES to a PES (Pruned Entity Structure) [3,4,5,6].

Presently, SES is used in many field overs structure modeling including the modeling & simulation, system design, and etc.

Block diagram is a diagram of a system, in which the principal parts or functions are represented by blocks connected by lines, which show the relationships of the blocks. They are heavily used in the engineering world in hardware design, electronic design, software design, and process flow diagrams [7].

However, the conventional SES tools only deal with tree-type graphic editing facilities. To overcome this limitation, we have suitably proposed the new method for building the structure model by combing the SES formalism with the block diagram representation.

The block diagram is typically used for a higher level, less detailed description aimed more at understanding the overall concepts and less at understanding the details of implementation. Contrast this with the schematic diagram and layout diagram used in the electrical engineering world, where the schematic diagram shows the details of each electrical component and the layout diagram shows the details of physical construction [7].

Therefore, this paper suggests the methodology that it SES-based structure model represent and can utilize the function provided by SES through the block diagram. This paper starts with 1 introduction. It introduces SES in the chapter 2. This paper illustrates the methodology to be proposed in the chapter 3 with the example image. Subsequently, in the chapter 4, it introduces about the user interface of the modeling tool which develops this by applying. A conclusion is provided in the chapter 5.

2 Background on SES

The SES contains entities and three types (aspect, specialization and multiple-decomposition) of nodes as follows; (see Figure 1):

- Entity - An entity represents a real world object that either can be independently identified or postulated as a

component of a decomposition of another real world[3,4,5].

- Aspect - An aspect represents the decomposition out of many possible entities. The children of an aspect are entities representing components in a decomposition of its parent[3,4,5].
- Specialization - A specialization is a mode of classifying entities and used to express alternative choices for components in the system being modeled. The children of specialization are entities representing variants of its parent[3,4,5].
- Multiple Decomposition - A multiple-decomposition is used to represent entities whose number in a system may vary[3,4,5].
- Coupling – A coupling is connection between entities. Each entity has the port and the parent entity has the coupling between ports with the attribute[3,4,5].

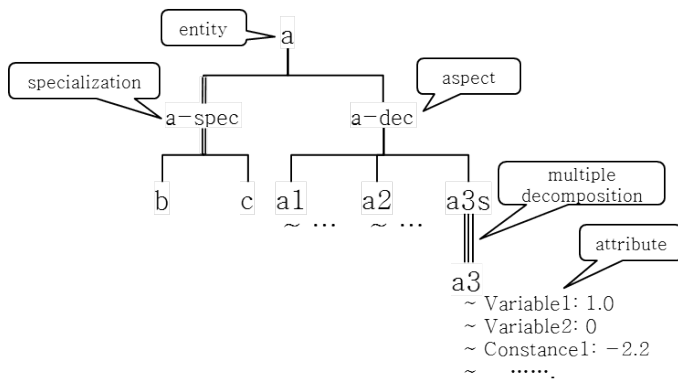


Figure 1. SES example

SES also provides three operations as follows;

- Pruning - Recall that a pure SES is on having no specializations and at most one aspect hanging from every entity. Pruning is required to create such pure SES. The result of pruning is a PES which contains fewer aspects and specializations than the original and therefore specifies a smaller family of alternative models than the latter. Ultimately, pruning terminates in a pure SES which specifies the synthesis of a particular hierarchical model (show pruning of table 1) [3,4,5,8].
- Cutting - Figure 3 shows tool for creating and partitioning SESs. Given an SES, entity 'A' with leaf entity, 'B', we can use operation cutting which (show cutting of table 1) [3,4,8]:
 - Removes the substructure of 'B' from 'A',
 - Reincarnates it in the form an SES, 'B',
 - Allows the user to pruned 'B' as many times as desired.
- Restructuring - The restructuring on SES may be accomplished on the basis of two operations; deepening

and flatting. The deepening is to create a parent entity of the entities, good for grouping. The invers operation of deepening is flatting. The flatting is to remove a parent entity of grouped entities, and its components are coupled to their grandparent entity. The restructuring is provided which automatically does the desired restructuring in such a way that behavioral equivalence is preserved ((show restructuring of table 1) [3]).

3 Proposed SES-based Structure Modeling Method Using Block Diagram

In order to support the convenient means to build the structure model, we have successfully proposed the structure modeling method that combines the SES structure model with the block diagram representation.

3.1 SES Building

SES Building means the set of tasks to create and edit the desired SES. As shown in table1(a), every properties such as aspect (decomposition), specialization, multiple decomposition and coupling can be easily constructed by using the block diagram.

SES building also can be easily developed on the basis of the block diagram as follows;

For the aspect of the SES, the sub-block is arranged inside the upper block and it expresses.

For the specialization of the SES, it arranges in the block expanding the alternative entity to the dotted line and it expresses.

For the multiple decomposition of SES, It marks 'is multiple', in the corresponding object and it expresses. And the input form for the choice the number is provided when pruning.

For the coupling of SES, in the output entity, the arrow is drawn the input entity. In the arrow, the name of the output port and input port is indicated.

3.2 SES Operation

SES operations also can be easily developed on the basis of the block diagram as follows;

For the pruning of the SES, the function and of selecting the alternative of the specialization and function of determining the number of the multiple-decompositions has to be provided by using the block diagram. The pruning of the table 1 (a) is the example of SES pruning with the block diagram. 'X, Y, Z' of Pruning of the Table 1 (a) indicates the substructure of 'A'. 'B is selected' means the select 'B' among alternative of 'A'. it becomes 'B.A' inheriting the 'A'

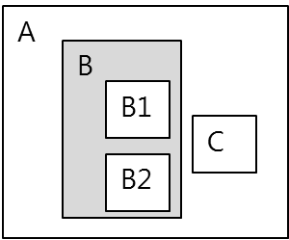
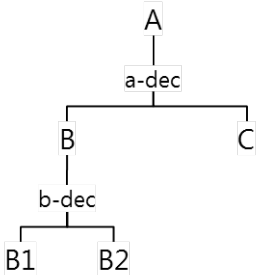
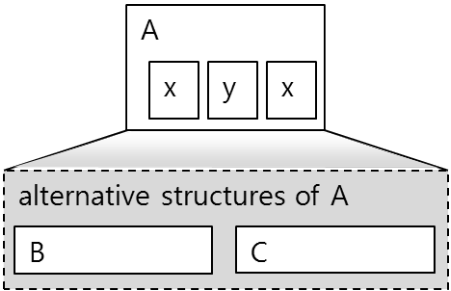
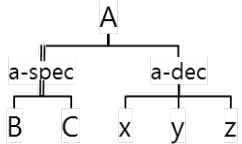
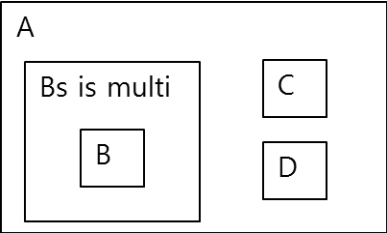
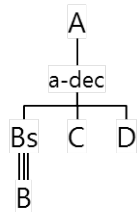
For the cutting of the SES, the structure model base [3,5,6] is needed. The sub-structure which becomes the cutting in any SES of structure model base can be separately stored to the model base. The cutting of the table 1 is the example of SES cutting with the block diagram.

For the restructuring of the SES, there is the function of modifying the coupling automatically. In the deepening, there is the coupling with the parent entity produced newly. In the flattening, there is the coupling removal with parent entity removed and generating the coupling with grandparent entity. The restructuring of the table 1 is the example expressing SES restructuring with the block diagram.

4 Conclusions

SES is the well-organized structural modeling formalism which has been successfully applied many research projects and systems; however, the conventional SES tools only deal with tree-type graphic editing facilities. To overcome this limitation, we have suitably proposed the new method for building the structure model by combing the SES formalism with the block diagram representation. The research is on going to develop the structure modeling tool based on the idea subscribed in this paper.

Table 1. Relation between with SES and block diagram representation
(a) SES building

	Task	Block diagram	SES
SES Building	Aspect		
	Specialization		
	Multiple Decomposition		

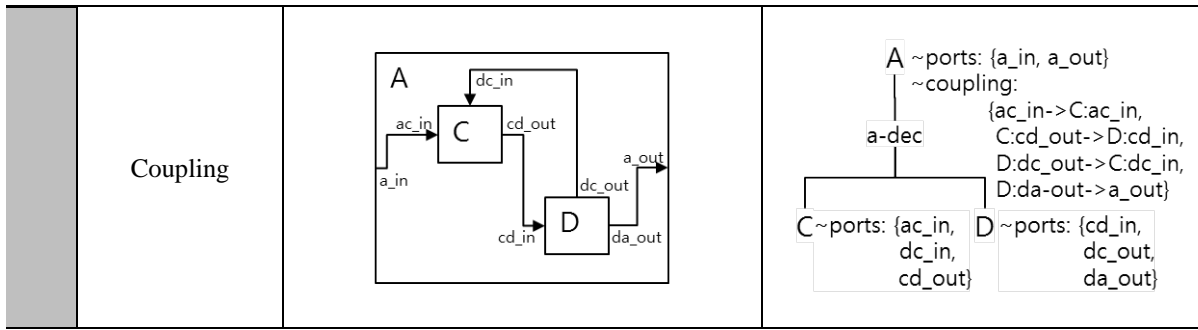
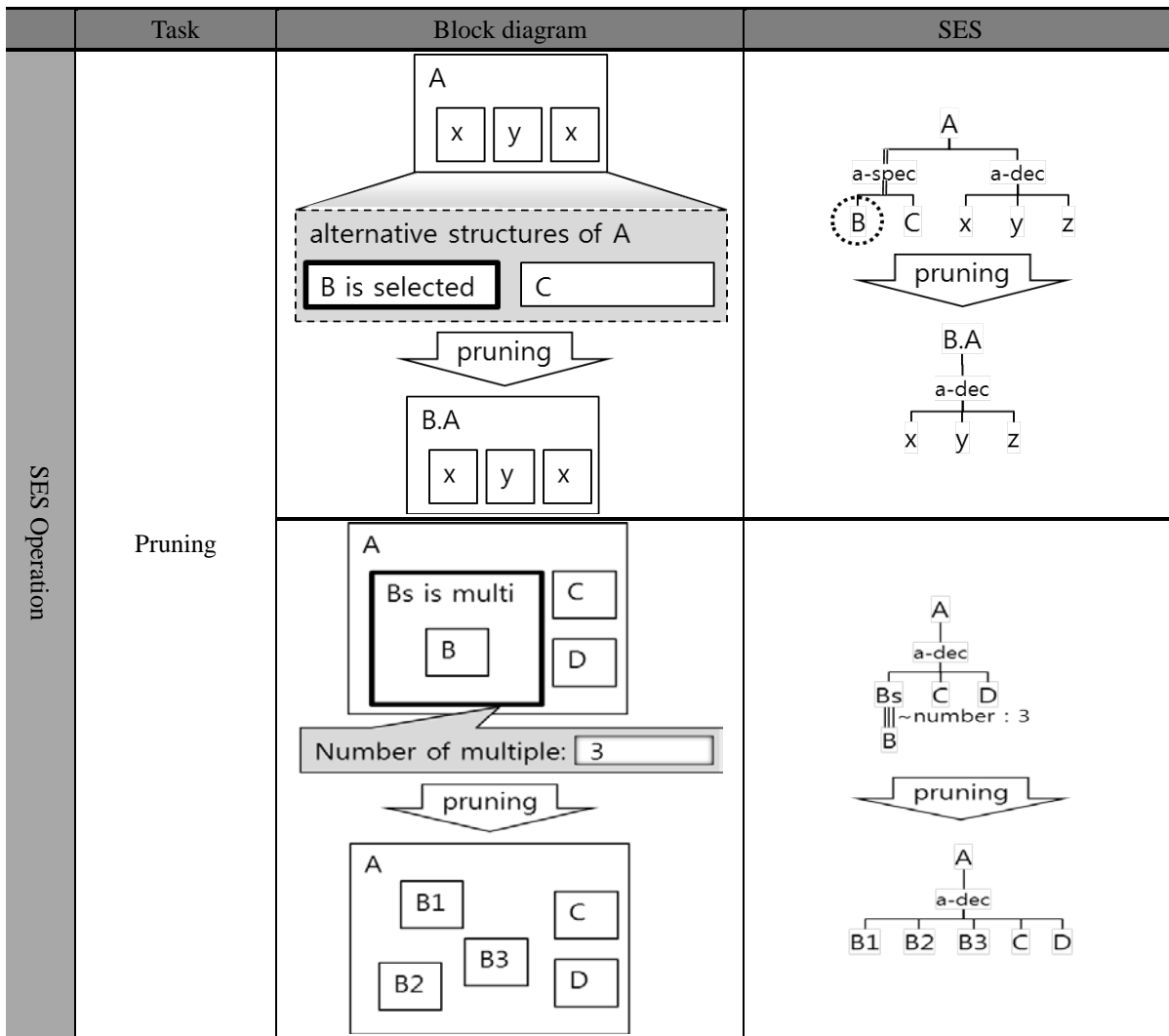
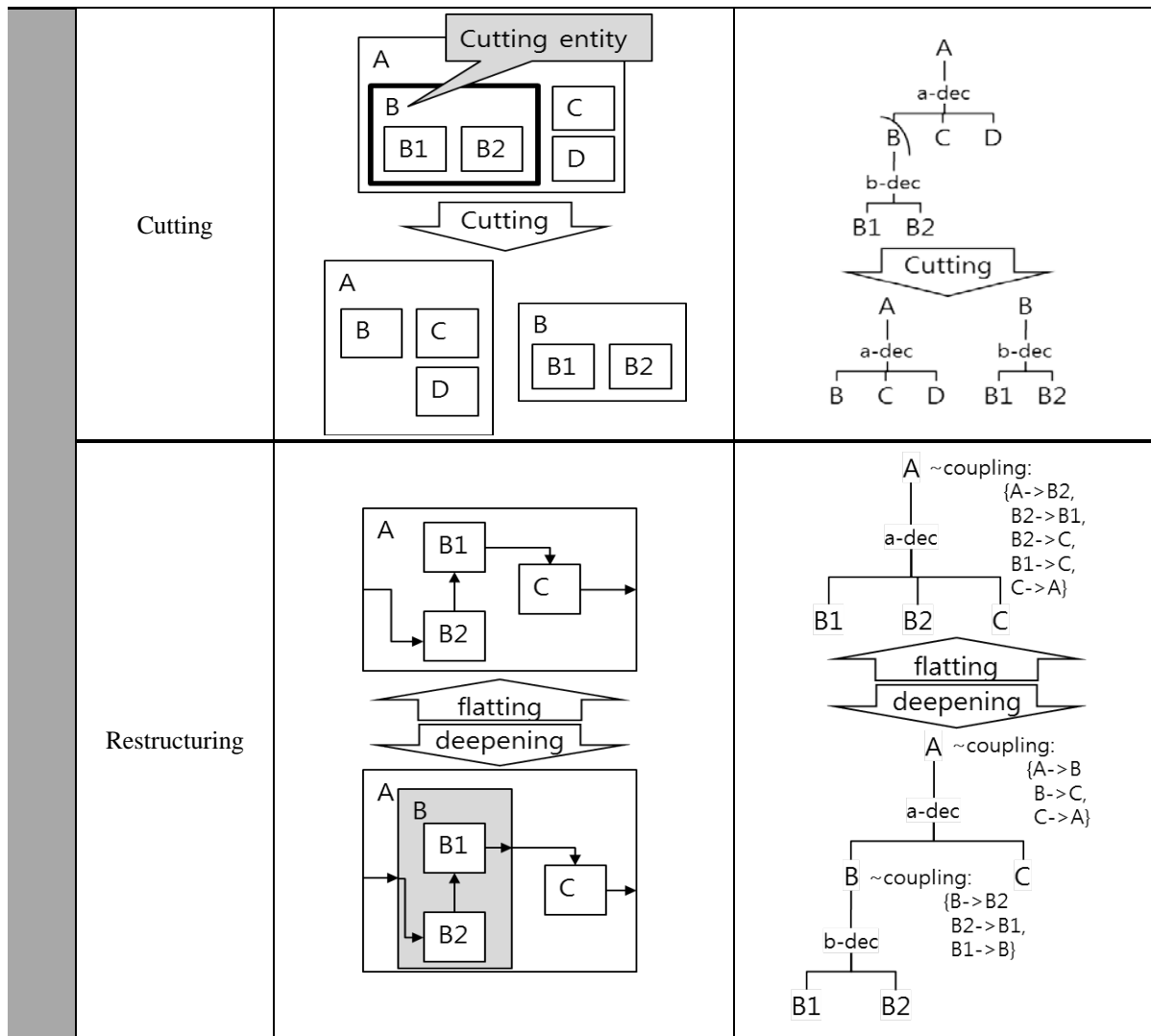


Table 1. Relation between with SES and block diagram representation
(b) SES operation





5 Reference

- [1] <http://www.mathworks.com/products/matlab/>
- [2] http://www.xjtek.com/anylogic/why_anylogic/
- [3] B.P. Zeigler, Object-oriented Simulation with Hierarchical, Modular Models: Intelligent Agent and Endomorphic Systems, Academic Press, 1990.
- [4] B.P. Zeigler, System-theoretic Representation of Simulation Models, IIT Transactions, 19-34, 1984.
- [5] B.P. Zeigler, H. Praehofer and T.G. Kim, Theory of Modeling and Simulation – Integrating Discrete Event and Continuous Complex Dynamic Systems, 2nd edition, ACADEMIC PRESS, 2000.

[6] Jerzy W. Rozenblit and B.P. Zeigler, Representing and Constructing System Specifications Using the System Entity Structure Concepts, 25th Conference on Winter Simulation, WSC '93, ACM, 1993

[7] http://en.wikipedia.org/wiki/Block_diagram

[8] B.P. Zeigler, Multifaceted Modeling and Discrete Event Simulation, ACADEMIC PRESS, 1984.

Acknowledgement

This research was supported by “The Combat Object Modeling Research for the M&S of the Maritime Weapons System”, that is the commissioned research business of ADD(Agency for Defense Development of Korea).

SESSION
AGILE SOFTWARE METHODS

Chair(s)

TBA

Low Fidelity User Interface Prototypes as Agile Refactoring Tools

Michael Wainer

Department of Computer Science
Southern Illinois University Carbondale
Carbondale, IL 62901

Abstract - *Agile development relies on refactoring and iterations to continually refine working code to meet requirements. The goal of refactoring is to produce code which is easy to understand and maintain. Refactoring user interface (UI) code is subject to ordinary refactorings plus others specific to this area. Even as refactoring tools grow more powerful, most programmers do not fully take advantage of automated support for many of the more complicated refactorings. As an alternative to automating source code refactoring tools with more and more sophisticated code transformations, our proposal brings in a more human element considering low fidelity prototypes (screen sketches etc.) as high-level refactoring tools. A beneficial side-effect may also be improved understanding of refactoring and earlier clarifications on important design issues. Suggestions are made for how low fidelity prototypes, already proven so useful in interaction design, can be used to get an earlier start on refactoring as well.*

Keywords: refactoring, prototyping, HCI, agile, tools

1 Introduction

Users have high expectations for their software and in markets where users have a choice, careful consideration of the user interaction design, including the graphical user interface (GUI), is a must. Developing highly usable software requires a considerable effort: practically universal recommendations are to involve users early and often, iterate, and evaluate designs using prototypes.

When developing such software in an agile manner, the use of low fidelity prototypes such as screen sketches and paper prototypes is extremely valuable. Such prototypes are effective not only for customer design evaluations but also as a way of specifying and communicating design ideas among the team. They are fast, cheap and easy to produce and modify. Agile interaction design can be coupled with agile software development by using an interaction design track in parallel with the development track. This dual track approach is discussed in section 2.

A common approach in coding GUI interfaces is to separate the internal data and logic (the business logic) from the particulars of how it is visualized and interacted with. Section 3 explores possible implementation strategies for coding GUIs including an introduction to the Model-View-Controller paradigm. Most interfaces use many components (widgets) which must be properly organized within the application's windows. Developers may be assisted in these tasks by ever more capable visual interface builder tools that are often part of the development environment.

Agile processes produce code which evolves through many iterations. Rather than spending a large amount of upfront time deriving a detailed and unchanging specification, change is recognized as a certainty so design work will continue throughout the project. To make this feasible the code must adapt; not only to add new features but also to stay easy to understand and maintain. Thus refactoring, improving and clarifying the evolving code (without changing its functionality) has become an important agile practice. Refactoring and the related idea of prefactoring with a special emphasis on interface code are taken up in sections 4 and 5.

Given the importance of refactoring when developing an application in an agile manner, section 6 looks at how low fidelity prototypes can be used as refactoring tools. As such prototypes are already available for communicating the interaction design, we show that they can also allow for rapid communication and exchange of ideas regarding refactoring options. An example of this is illustrated by following a sample case introduced in earlier sections. Finally, section 7 summarizes and concludes this paper and gives a look towards future work.

2 Agile Development

We assume an agile development methodology [1] for both GUI and application development as a whole. Agile development expects change and therefore does not put much effort into big upfront design. Instead, it is recognized that the code will have to evolve. Agile development will also favor using lightweight modeling artifacts and documentation as the main emphasis is on working code.

While agile development does not favor big upfront planning, some form of planning is often appropriate. During an iteration, developers will be implementing code derived, in part, from an earlier iteration's interaction design work. As shown in Figure 1, it has been found useful to consider dual tracks of development: one for the interaction design and one for application development [2, 3]. Rather than striving for a complete and detailed interaction design, the interface is designed in enough detail for just the features present in the current development iteration. The interaction design team (which may overlap with the developers) works with prototypes and/or existing code from previous iterations to create a design for the next iteration. Designs are expressed to the developers in an agile manner – often as paper or low fidelity prototypes [2]. An iteration 0 may sometimes be used for initial setup and exploration of internal details which have little dependence on the user interface.

Paper prototypes, screen sketches and navigation flow diagrams have proven to be useful artifacts to communicate the specification for the user interface of applications. These artifacts may be created by hand or by various drawing or modeling applications. They are generally not used to directly generate code but to communicate the design amongst members of the team. Paper prototypes can simulate much of the behavior of a system before any coding. The sketchy nature of low fidelity artifacts invites more appropriate user feedback on early design aspects.

To illustrate more specifically, we use an example of a small Java Swing application, PolygonMaker, which is to be developed in an agile iterative manner. PolygonMaker is an application to assist developers in generating the source code required to specify a polygon for various systems or languages. The user is able to specify the polygon points

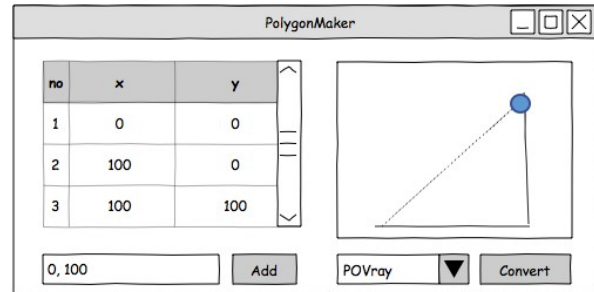


Figure 2: An early design screen sketch for the PolygonMaker example application.

and initiate a process to generate the source representing the specified polygon. Initial choices might be as a Java2D Shape, POVray (a ray-tracing package) prism object, etc. A screen sketch of PolygonMaker's main window is shown in Figure 2.

The user specifies new points in the entry field at the lower left and the polygon is visualized as the table of points on the left and the diagram on the right. The diagram shows the last point entered with emphasis and connects it to the first point with a dashed line to illustrate the resulting polygon. The design is expected to change as the application evolves. The user interface aspects may be under pressure to change in different ways than the underlying application's algorithms and data structures (its “business logic”). The next section discusses a coding approach commonly used to combat this problem as well as the help that can be offered by visual interface construction tools.

3 Implementation Strategy

A commonly accepted principle of good object oriented design is the “Single Responsibility Principle” [1] which seeks to limit the pressures of change exerted upon objects by having objects focus upon a single facet. In this way, they will only have to be modified in response to changes impacting that facet and are kept rather isolated from other possible changes. Like many design principles, while the basic concept seems simple, in practice it can be difficult to apply. In terms of software supporting graphical user interfaces, the Model-View-Controller(MVC) paradigm is commonly used to separate concerns (see Figure 3).

In the Model-View-Controller(MVC) the internal core logic of application entities (business logic) is kept separate from the user interface. Business rules and interface concerns change for different reasons and separating these concerns helps to isolate changes in one from affecting the other. Having a separable model also makes it possible to develop and test the model independently or in parallel with

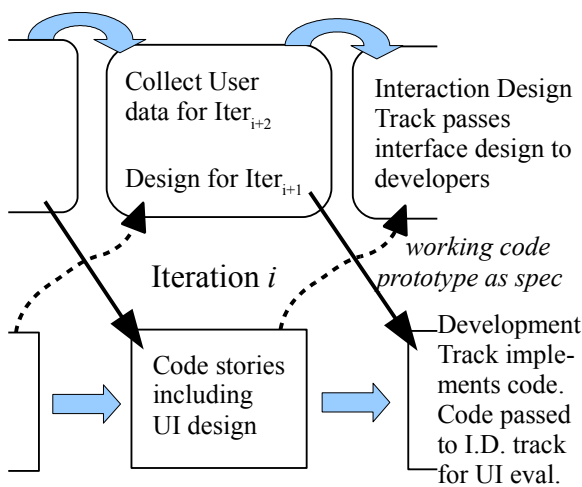


Figure 1: Agile development using parallel tracks for Interaction Design (top) and Development (bottom).

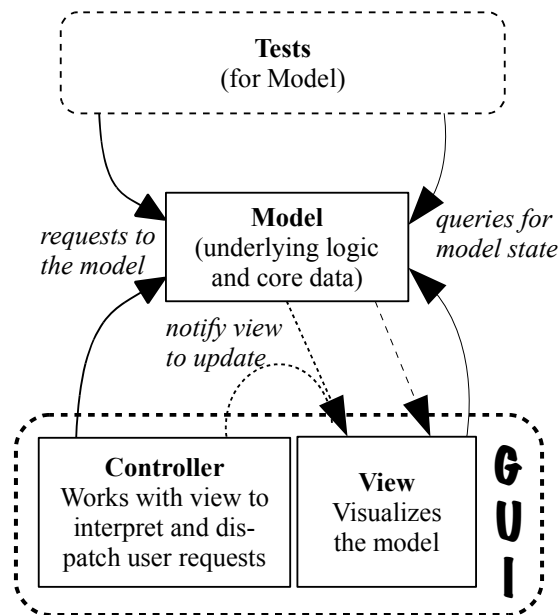


Figure 3: The Model-View-Controller separates core logic and data (the model) from the user interface (view and controller). Tests may also be used to drive and develop the model.

the GUI. Test Driven Development, a common practice in agile development, incrementally builds a suite of tests which drive and direct the development of the model.

Since constructing a GUI involves the specification and creation of its visual aspects, it is only natural that visually based tools can be very helpful. In this paper, *WindowBuild-*

er Pro [4] a visual construction tool for Java Swing interfaces (and other systems as well) is taken as representative of these utilities. These tools usually provide similar interactive design spaces. A design space visually shows the interface under construction and allows changes to be made by dragging components from a palette. A structure view shows the containment hierarchy of the components. Selected components are highlighted in both the design and structure views. A selected component also has properties that are available for inspection and modification in a properties view.

Figure 4 shows a screenshot of *WindowBuilder* working on another screen for the PolygonMaker example application. *WindowBuilder*, now freely available as a plugin for the Eclipse IDE is highly capable and recently acquired and released as open source by Google. While such tools reduce the tedium of coding many details of a GUI, they have drawbacks as well. From unexpected widget sizes and layouts, to failures and delays in parsing, it still can be far easier to sketch a screenshot by hand (especially if custom graphics components are required). Thus even though visual build tools are increasingly powerful and useful in building code, they have not eliminated the need for simple low fidelity prototypes for early design.

It is also interesting to note that the code generated by GUI builders is often quite different from what would typically be constructed manually. Developers may have some control over aspects of the code generated but in general, machine generated code will use uninformative names and create fewer but lengthier methods. The code tends to be easier for the tool to generate and read than it is for humans. This may become an issue especially in the context of an it-

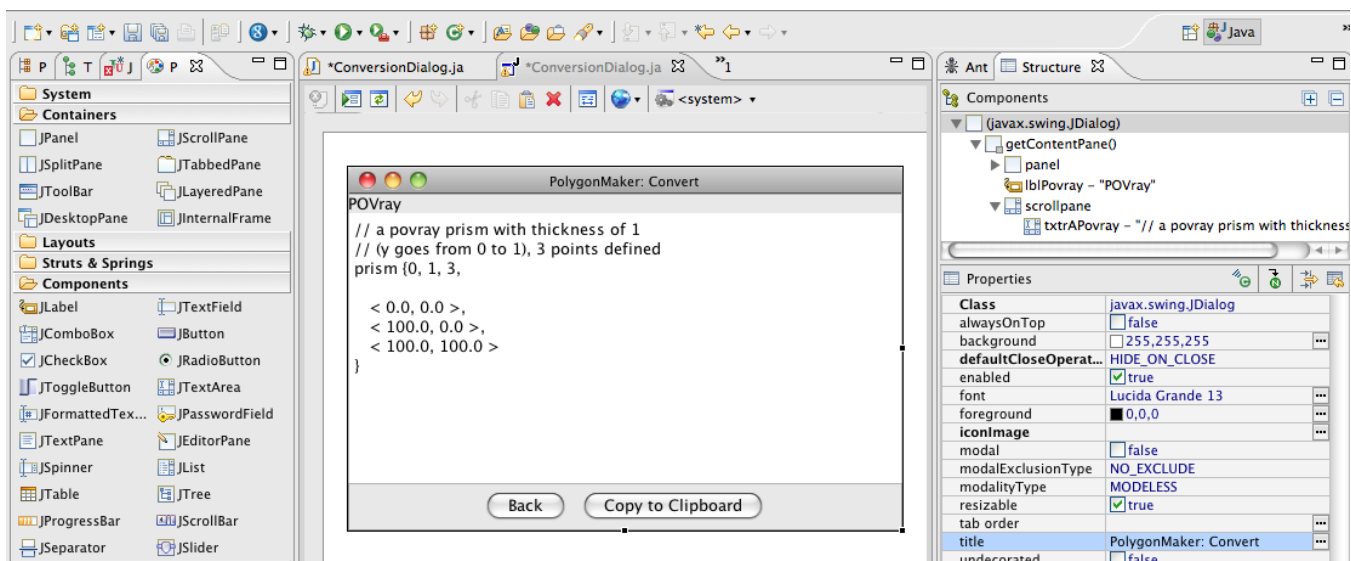


Figure 4: WindowBuilder shown in design mode, while creating a conversion dialog for the example application. PolygonMaker, allows users to enter polygon points and then generates equivalent source code for a variety of systems.

erative development process in which the code is continually revised. In effect, reverse engineering will be needed to interpret and adapt the code generated by the visual build tool[5].

4 Refactoring

Over time as code evolves it often begins to diminish in clarity and quality: methods grow too long, logic gets more convoluted, the original names, data structures and algorithms may become less appropriate. These problem areas in the code are referred to as smells. Many specific code smells have been cataloged along with suggested refactorings to remove them. Refactorings are transformations of the code with the goal of making the code easier to understand and maintain [6]. Specifically, refactorings, while they may be applied to improve design in preparation for new features, do not themselves aim to implement new features. Since agile development methods use continuous design, rather than big upfront design, refactoring is essential to maintain the quality of the code as it evolves.

A very frequent refactoring activity, *rename*, is supported both in visual construction tools as well as through source code editors. Good names can significantly improve the ability of the code to communicate its intent. Naming is so important that Martin in “*Clean Code*”[7] provides seven heuristics devoted to just that consideration alone.

Another frequently used refactoring activity is *extract*. It is often used to reduce the number of lines in methods having the “long method smell”. A method is reduced in size by extracting some of its lines into a new method. The original method then refers to the newly extracted method. Once again naming is important as the new method should have a name which makes its purpose clear. A call to a clearly named purpose often eliminates the need to comment a sequence of statements.

As refactoring becomes more widely practiced and understood, IDEs are increasing support for automated refactorings. Eclipse[8] provides a refactoring menu with many of the refactorings from Fowler[6]. Research is underway to study more about how developers use refactoring tools in their work [9-11]. While IDEs like Eclipse now provide many refactoring options, it has been difficult to determine exactly how refactoring is used in practice[10]. Some studies try to determine refactoring histories by examining repository data while others have used instrumented IDEs to collect and report various types of information. Experiments and interviews with developers are another approach. Overall it seems as though only a handful of the automated refactoring features are widely used. Some attribute this, in part, to the difficulty (poor usability) of using refactoring tools[10-11].

It is logical to assume that certain types of refactorings might be associated with specific types of code. Indeed, GUI code analyzed by participants in Mäntylä and Lassenius[9] refactoring study was noticed to tend toward typical characteristic smells. Furthermore, Marinilli[12] suggests several specific refactorings that are applicable to GUI code. Visual interface construction tools will often directly support some forms of refactorings (even as the code they generate may necessitate others [5]). The complexity of supporting a modern interface suggest a multitude of refactorings which may be possible including: switching layout managers, obtaining text and image content from a resource loading system, internals of event handling, undo and logging. Aside from pure UI issues, the model, or what the application uses as its internal model(s) might also be refactored.

5 Refactoring Early: Prefactoring

Recognizing that code will need to change and that there is value in refactoring, can the practice be advanced by utilizing the knowledge gained from previous software development experiences? That is the idea behind *Prefactoring* as put forward by Pugh[13]. Some of the main concepts emphasized in prefactoring are listed below:

Interface: more focus on what components should do rather than on how.

Abstraction: express ideas with pseudocode.

Separation of Concerns: responsibilities are split between classes, methods etc.

Readability: code should clearly communicate what it does.

There is considerable overlap between these concepts and those supporting good interface design and implementation. *Abstraction* and *interface* manifest themselves in the ideals of expressing interaction design with low fidelity prototypes: offering a high-level way to visually explore what an application should do. *Parameterization* permits an abstraction to consider a generalization of items rather than each individual instance separately. *Separation of concerns* mirrors the MVC paradigm. *Readability* relates to choosing meaningful names and appropriate chunking of ideas. Grouping related interface items together often makes sense for the user (logical groupings, consistency) as well as for the implementation (readability, reusability, reduced duplication).

6 Low Fidelity Prototypes: Refactoring Tools

As software developers prepare to start a new iteration, assume they have available (or can quickly construct) a low fidelity prototype. Consider how that prototype could be used for refactoring (or prefactoring). While not meant to

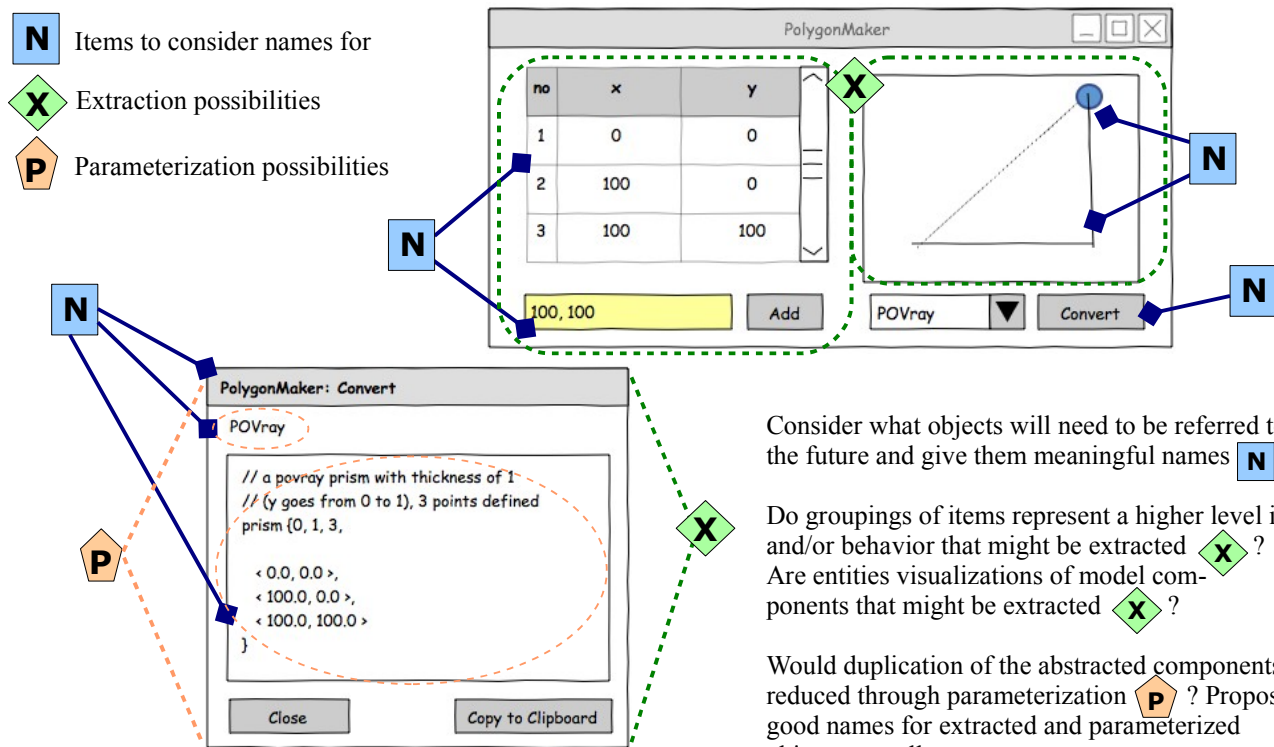


Figure 5: Prototype sketches as refactoring tools.

be an exhaustive list, for clarity, we will focus on refactorings concerned with *naming*, *extraction* and *parameterization*. These refactorings and how they may be explored with low fidelity prototypes are now considered in more detail.

name/rename: interface items may have names visible to the user (labels, titles etc.) as well as internal names. Especially objects that are to be referenced at larger scopes, should have clear names. Naming and renaming can iterate quickly and using the prototype it is easy to involve interaction designers and/or customers in the discussion about appropriate names. Good names are extremely important for creating highly readable code.

extract: a window or dialog may contain many components or areas devoted to different types of information or interaction. The implementation might specifically use separate methods extracted to group together code devoted toward different aspects. Extract may also be useful for breaking down event handling routines. New classes may also be extracted.

parameterize: a window, pane or other object or method may be representative of a family of similar objects. Rather than create each item with distinct code, merge implementations and use parameters to specialize. This can reduce instances of duplicate code.

Consider what objects will need to be referred to in the future and give them meaningful names **N**

Do groupings of items represent a higher level idea and/or behavior that might be extracted **X**?
 Are entities visualizations of model components that might be extracted **X**?

Would duplication of the abstracted components be reduced through parameterization **P**? Propose good names for extracted and parameterized objects as well.

Recall the MVC paradigm discussed earlier. View components may be the first to mind when examining a prototype. In Figure 5, view aspects are clearly the polygon graph and its table of points as well as the view of the generated source code. The graph is likely to be a custom component but it also brings to mind the internal polygon model and what information it contains. A basic polygon model may be composed of a list of vertices but to support graphical editing the application model may add a “selected vertex” data structure as well. In this application, the model is supplemented by additional converter objects to generate source code conversions for polygons. These model classes (or perhaps interfaces), objects and methods all require good names. Models can be tested without the GUI so initial ideas regarding names, method parameters etc. may already exist.

The controller coordinates the events and provides buttons like “Add”, “Copy to Clipboard”, “Convert” etc. In some cases, controls may be enabled/disabled based upon the state of the model (i.e. “Convert” should not be enabled if at least 3 points haven’t been defined). Since such controls need to be referenced to update their status, they require special naming considerations. An alternative implementation strategy might use Swing Actions but again the actions should be carefully named.

To illustrate more specifically, the “Convert” dialog (bottom left of Figure 5) is opened once the user has specified a

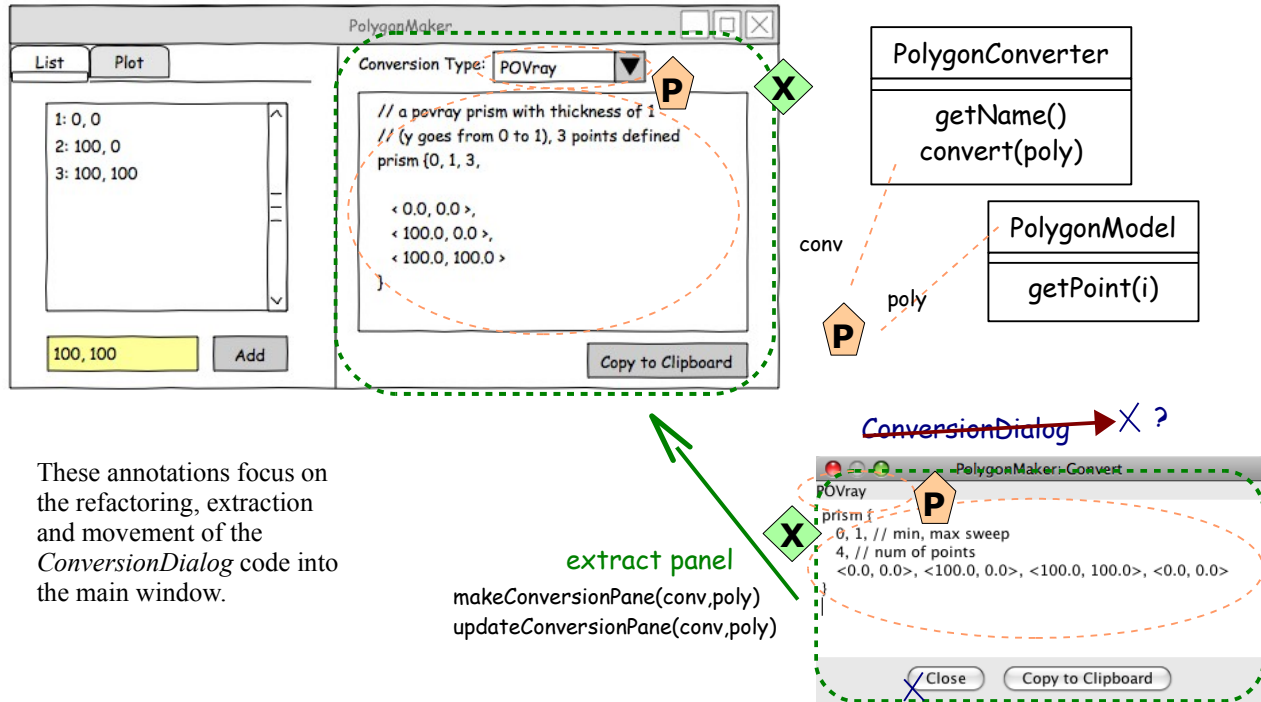


Figure 6: Further design iteration moves the conversion dialog information into a section of the main window.

polygon and clicked the *Convert* button. It is shown for a POVray conversion but no doubt it will be very similar to the conversion dialogs for other source code options. Rather than duplicate essentially the same thing (removing duplication is a common refactoring goal), the dialog can be extracted into a new class and its constructor parameterized. The orange dashed ellipses indicate possible parameter items – the items are also annotated for naming. Deriving good names naturally leads to thoughts about what the data represents and, if it is dynamic, how it changes. Here the data could be passed in directly as a parameter or be generated by other objects (polygon and converter objects).

Of course, once implemented, items may continue to change and new prototype sketches generated by the I.D. team reveal how. Figure 6 shows both the implementation of the conversion dialog along with the design sketch for the next iteration. Screenshots of the working software (and other design diagrams, uml etc.) can be used along side the prototype, to plan the refactoring and to prepare for design changes and new features. Prototype sketches can be scanned and overlaid with notes regarding the current area of focus (refactoring overlays). Figure 6 concerns moving the *ConversionDialog* into a section of the main window. The notes show a refactoring plan to extract the content creation aspect of the dialog to a method that instead creates a pane. The pane can then be used in the main window. To

work with all different types of conversions, parameterization for different type names and source code is maintained. Since the pane will remain open, an update method is also proposed.

Notice that extraction of the code requires new names, decisions of what parameters to use, as well as a determination of where the code should go. Renaming or eliminating the original class is a possibility. Even though the existing code will ultimately be refactored using the refactoring tools of the IDE (and/or manual code refactoring), it can still be helpful to work out a refactoring plan by beginning to refactor using the low fidelity prototype. This can provide a more fluid environment for thought and discussion with a greater sense of context as the existing code is prepared for modification to fit into the new design. A rapid exchange of ideas over the low fidelity prototype may serve to explore and eliminate some refactorings before making any changes to the actual source code.

As new features are considered, the existing code modifications move beyond refactoring but the issues of renaming, extraction etc. become immediately apparent and the process repeats. Prototypes with their refactoring overlays may be saved (scanned and checked into the team repository) for review during retrospectives or whenever needed.

7 Conclusion

Much effort has been placed into making powerful automated refactoring tools available to developers. Research has shown that these tools do not seem to be used as often as might be expected. Some tools suffer from usability issues and there may also be a lack of awareness of complex refactorings among developers. Rather than focusing exclusively on building more powerful automation into refactoring tools, this paper proposes a contrary idea; expand refactoring tools to the more human side by overlaying refactorings onto low fidelity prototypes.

A criticism of prefactoring, and perhaps this method, is that we may fall into the trap of too much upfront planning. That is obviously not the intent. We merely wish to introduce “prototypes as refactoring tools” as one more tool to be used where appropriate. As is the case with simple low fidelity prototypes, they provide a fast and flexible way to consider many ideas quickly and in a way that serves to foster communication among team members. Much of maintaining good clean code involves naming and communicating intent. Informal, light and fast, refactoring with prototypes allows for quick iterations over naming and helps to raise important ideas about what aspects might be extracted and parameterized.

To quote from the Agile manifesto[14]: “Individuals and interactions over processes and tools”. Teammates working together around a whiteboard or table on the initial refactoring for an iteration using prototypes promotes this idea. While there is certainly value in automated refactoring tools, ultimately there is a very human side to crafting code which effectively expresses its intent. Ideally developers will have a broad spectrum of tools to help them succeed in this challenging endeavor.

We plan to continue exploring these ideas. It is expected that refactoring using prototypes will also aid in brainstorming with the team (and customer) and help in story estimation and splitting. Refactoring discussions using prototypes may also raise awareness of refactoring options in general and result in more use of automated refactoring tools as well. We are hoping to collect data on these anticipated effects as well as to determine what if any new notations might be appropriate for annotating refactoring thoughts onto screen sketches etc.

8 References

- [1] Robert Cecil Martin, “Agile Software Development: Principles, Patterns, and Practices”. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2003
- [2] Jeff Patton, “Twelve emerging best practices for adding UX work to Agile development”, 2008 , http://agileproductdesign.com/blog/emerging_best_agile_ux_practice.html
- [3] Lynn Miller. “Case Study of Customer Input for a Successful Product”. In Proc. of the Agile Developers Conference (ADC 2005), Denver, CO, pp 225-234, 2005.
- [4] WindowBuilder (Pro), <http://code.google.com/javadevtools/wbpro/>
- [5] Michael Wainer. “GUI Tools and Generated Code: Refactoring to Reveal Intent”. In Proceedings of the International Conference on Computers and Their Applications (CATA 2011), New Orleans, LA, pp. 108-113, 2011.
- [6] M. Fowler. “Refactoring: Improving the Design of Existing Code”, Addison-Wesley, Boston, MA, USA, 2000.
- [7] Robert Martin. “Clean Code: A Handbook of Agile Software Craftsmanship”, Prentice Hall, 2009.
- [8] Eclipse Project, <http://www.eclipse.org/>
- [9] Mika V. Mäntylä and Casper Lassenius. “Drivers for software refactoring decisions”. In *Proceedings of the 2006 ACM/IEEE international symposium on Empirical software engineering (ISESE '06)*. ACM, New York, NY, USA, 297-306, 2006.
- [10] Mohsen Vakilian, Nicholas Chen, Stas Negara, Balaji Ambresh Rajkumar, Roshanak Zilouchian Moghadam, and Ralph E. Johnson. “The need for richer refactoring usage data”. In *Proceedings of the 3rd ACM SIGPLAN workshop on Evaluation and usability of programming languages and tools (PLATEAU '11)*. ACM, New York, NY, USA, 31-38, 2011.
- [11] Emerson Murphy-Hill and Andrew P. Black. “Breaking the barriers to successful refactoring: observations and tools for extract method”. In *Proceedings of the 30th international conference on Software engineering (ICSE '08)*. ACM, New York, NY, USA, 421-430, 2008.
- [12] Mauro Marinilli, “Professional Java User Interfaces”, John Wiley & Sons, West Sussex, England, pp.191-201, 2006.
- [13] Ken Pugh. “Prefactoring”. O'Reilly Media, Inc., 2005.
- [14] Agile Manifesto, <http://agilemanifesto.org/>

All web sources were accessed March 2011.

Adopting Agile Methodologies to a One-Man Software Engineering Team

Steve Biccum¹, Nasseh Tabrizi²

¹Department of Computer Science, East Carolina University, Greenville, NC, USA

²Department of Computer Science, East Carolina University, Greenville, NC, USA

Abstract - *This paper presents how Agile software development methodologies can be adapted to suit the one man team. The paper follows the development of an application called WebChecker; a Ruby on Rails based software application. Throughout the development of the WebChecker project, special care was taken to try to stay as true to the Agile principles as possible. Both tooling and methodology were tuned for Agile development. The scheduling application that was built for Agile development, and supported all of the Agile artifacts out of the box which made it a natural fit for the project.*

Keywords: *Agile, Management, One-Man Team, Software, Development*

1 INTRODUCTION

Agile [1] and its iterative and incremental [2] nature is clearly more than a fad or development “flash in the pan”, but rather the evolution of some of the established incremental and iterative methods of software development of the past. Prior to the 1990's, software development was completed largely with either a “Waterfall” [3] approach or some flavor of incremental and iterative [4] approach. Waterfall, being a rigid format of where all requirements gathering takes place in the beginning of a project development and Incremental/Iterative development being more tolerant to change by evolving as a project progresses to manage risks. Both approaches have their benefits depending on the project at hand, but both proved to be documentation heavy and somewhat slow to deliver measurable value.

In the mid 1990's [5], small groups of developers began to embrace a more lightweight style of development and came up with several new approaches for developing software. These methods were different from traditional schools of thought in that they embraced the end user point of view of software that adds business value and less of the developer point of view that seeks to develop features to meet a requirement. In 2001, several leaders of the software movement dubbed “Agile” came together in a meeting to seek to find common ground around the various processes that had developed over the last decade. From that initial meeting, the Agile Alliance was formed and the following “manifesto” [6] was released that embodies the Agile way of software development:

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value: Individuals and interactions over processes and tools; working software over comprehensive documentation; customer collaboration over contract negotiation; responding to change over following a plan [7].

When adapting Agile principles to the single man team, it was natural to draw on some of the existing methodologies for somewhat larger teams. Agile SCRUM specifically served as a good starting point and the methodology was customized from there [1]. Although the SCRUM methodology takes its name from a type of meeting among the developers, it still served as a good starting point for ideas. Naturally a “daily standup meeting” that is traditionally held among developers was not required, but other core concepts did prove useful. Agile SCRUM employs the use of user stories, story themes, and of course, the user story backlog to house all of the stories [1]. Beyond that, the project also adopted incremental and iterative scheduling practices that are popular among many flavors of Agile including a story sizing technique based on a metric referred to as story points [1]. Using these pieces of Agile, it was possible to construct a viable one man form of agile development.

2 THE WEBCHECKER SYSTEM

The motivation for the creation of the project called WebChecker was simple, web administrators need to know that expected content is making it to the screens of their clients. Traditional methods of server checking are generally based on checking response headers or some other generic check of page response. In order to verify that a particular page is being served properly, a more sophisticated way of checking must be employed. WebChecker provides a way for savvy web administrators to construct a series of regular expressions that can be used as verification points that prove to the web server administrator that the page did, in fact, load all content properly. Placing this verification on a schedule and adding alerting capabilities such as email, extends the check to be a monitor for any given website. The concept is simple, and while many companies offer some form of checking and verification of site presence, it is not easy to know exactly how these companies verify site availability. With WebChecker, presence is determined directly in a binary way. The verification point will either match or not match. This is similar to the way several web enabled functional

testing frameworks operate, but on a much more targeted scale.

3 AGILE DEVELOPMENT

3.1 User Stories

After defining the scope and nature of the project,

the next step was to start translating the project into requirement terms. In the Agile context, this means defining a set of statements called user stories [8] that capture the bits of business value that when combined form the overall solution. Since Agile is focused from the user perspective, all user stories are written from the user's perspective. This assists the developer in looking at the solution from the user's point of view. A typical story is a single sentence that defines the role of the requester of the requirement, the actual requirement of the solution, and finally and perhaps most importantly, the reason for the requirement.

Although somewhat clunky in syntax, the requirement's reason is most significant because it can lead to other stories by placing customer requirements in context. If the users say they need a feature without properly defining the need being satisfied, it might be easy to miss the underlying need for another feature. For example, if the client requests a feature to track web response times, but the question of "why" was never asked, the fact that the company was actually monitoring responses in order to make a vendor hosting decision might be missed. Tracking response times is simple enough, but the need to group web sites by hosting vendor would most certainly affect how database tables are constructed. Furthermore, presentation of the aggregate response times by vendor would certainly be useful for decision making.

For the Webchecker project, since there was no existing infrastructure, many of the stories were obvious – tool selection, infrastructure decisions, and a high level design of the software. Once past the initial layer of these basics, stories were written that shaped what features would ultimately be implemented including elements of both presentation and collection elements. Most of the stories are written at the onset of a project but as the project progresses, it is quite natural to generate more stories. These new stories as a rule are placed on a queue for later sizing and scheduling. In this manner, the work that is in progress remains in focus and the schedule is maintained.

3.2 Story Points

Once the stories are written, each story must be assigned a unit for size estimation. This unit is based on perceived complexity and is referred to as story points [9]. Story points are a "quick and dirty" estimation technique where developers estimate complexity in terms of similar code that they have developed in the past. A scale is used that helps define a level of complexity that sets it apart from the directly adjacent scale markers. The scale can be a simple Fibonacci sequence of

1,2,3,5,8,13, etc. or, in some cases use terms similar to shirt sizes: extra small, small, medium, large, extra-large. Either is acceptable, but the point is to quantify the level of effort required for a given story. Proper estimation of story points should also take into account the developers experience with the language, code base and the amount of discovery involved with the aspects of the project that the story's implementation would involve. Once quantified, judgments can be made for scheduling, and planning.

For this project, stories were all assigned points just after the stories were written. This allowed for first pass estimation while the story was still fresh in the mind and no additional review was required to gain understanding. Each story was evaluated in terms of what the story would take to fully implement. Depending on the story, this could include estimating database work, interface and functional testing work and, of course, the writing and testing of the unit level code. The project management software that was chosen for this project (discussed later in this paper) provides an easy way to record the story points in terms of shirt sizes (xxs, xs,s,m,l,xl,xxl) with an underlying numerical component that is used for point tracking and project management processes.

3.3 Themes

Once the stories were written and assigned points, stories could be sorted into logical groups called themes [10]. Themes serve a few critical purposes in the Agile software development cycle.

First, the grouping of stories forces the developer to take a step back and take a broader look at the projected work. This high level view allows for sorting and determining interdependencies or competing stories. Interdependent stories typically will dictate some order of implementation while competing may require some stakeholder involvement for resolution. In either case, there is the potential impact on implementation or overall design. Themes also provide a scheduling convenience. Once themes are created, large collections of work can be evaluated for the optimal implementation schedule. Multiple themes can be started concurrently if required.

Finally, and perhaps most importantly, themes provide a goal for each release and a way to clearly articulate that goal. The first release for the WebChecker project, for instance, was to build the infrastructure, select tools, complete the software design, and finally implement the core agent functionality. By simply evaluating the list, even the most passive observer can glean what deliverables would be in place at the end of the first release: a functioning server environment, a set of development tools, a software design, and the first release of the core agent software.

The WebChecker project was broken into stories that were grouped into three distinct themes: infrastructure work, core agent work, and web presentation work. Infrastructure stories relate to building the underlying server hardware and operating system software, as well as tooling research and tooling selection. Core agent work was dedicated to the

components of the system that, as the name suggests, provide the core functionality of checking and alerting functions. The web presentation work's goal was to implement a means for persisting response data for analysis and actually providing data presentation via web based charts and graphs.

3.4 Acceptance Criterion

As noted above, each story was written with an acceptance criterion appended to the end. A criterion was written for every story as it was entered to the tracking system (also known as the product backlog). The idea behind the writing of the criterion is to serve as a quick way to determine the point of "done". In other words, "this story will be complete when the following requirements are met: A,B and C." In the above example, a design artifact must be complete and a unit test generated. In this case the code itself is assumed.

3.5 Incremental and Iterative Scheduling

As mentioned, agile methodologies encourage incremental and iterative development. To achieve this, the project made use of an incremental release and iterative sprint schedule. The release itself has a deliverable at the end that seeks to match the above mentioned themes. Inside each release can contain any number of sprints as determined necessary. In this case, the project was split into three releases with two, two week long sprints inside each release.

As a best practice, the Increments for this project were named using a date format and increment number YYN. Sprints were numbered sequentially within the increment with a dash. For this year, 2011, the first increment will be named 111 and the first sprint will be named 111-1. The second sprint of the first increment is named 111-2. The first sprint of the second increment is named 112-1 and so on.

3.6 Agile Status Checking Backlogs and the Burndown Chart

As lightweight as Agile can be, it is still important to keep a status of the project to determine if a project is behind or on schedule. Agile, or more specifically the Scrum flavor of Agile development, introduces the concept of a "backlog" that represents a list of the known requirements for a given software project [11]. The backlog is the total bucket of work for the project. As the stories are completed, the total number of incomplete story points in the backlog will decrease, the result of which can be traced on a graph. The result is a downward slope from upper left to lower right of the chart as work is completed.

3.7 Project management

Project management software selection was particularly tricky for this project. There are multiple competing applications available to manage an agile project. Since this

was such a small scale and single user type project there were even more choices as there are many enterprise class management applications available for free or low cost when using small teams or small projects. While these were considered, these generally were deemed to be heavy for this project. The core requirements were: lightweight, simple to use, with built in support for the agile artifacts (stories, story points). The selection for the WebChecker project was a single user style, Java freeware planning tool called "Scrumpy" [12]. Scrumpy supports all the agile artifacts and scheduling routines and has some built in charting functions for displaying progress, and is intuitive and effective. One of the most attractive aspects of the tool was the clarity it provides when observing the backlogs and the project backlog.

3.8 Language and Framework

Given the scope of the project, it was clear very early on that Ruby on Rails was a good choice for development. The Rails portion of the configuration allows for rapid development with automated helpers for database configuration. It also uses a rigid Model-view-controller pattern for software development that will lead to better overall system design. Another key advantage of Ruby is the active developer community.

This support allows for a good point of reference in terms of syntax and troubleshooting. As with many software communities, libraries are routinely released as optional packages that extend the base language in various ways. These extensions are bundled into independent libraries that can then be imported for various functionality. In the Ruby community, these libraries are called "gems". These gems provide an easy way to avoid "reinventing the wheel" and in many cases, can speed delivery of working software.

3.9 Development Tooling

An Eclipse based solution was selected as the primary development environment and the search for a suitable Ruby plugin for eclipse became a priority. After some shopping around for different alternatives, Aptana Studio 3 (formerly RadRails 3) was selected as it plugs in perfectly into the Eclipse environment. Ruby on Rails also implements something called Representational State Transfer perhaps better known as REST. The Hyper Text Transfer Protocol relies on 4 core actions: post, put, get, and delete. If you combine these actions with a url object to act on, a consistent syntax emerges that forms the basis for all HTTP create, read, update and delete or CRUD operations. In Ruby on Rails, it means that the Uniform Resource Identifier or URI on the location line represents the object or objects that will be subject of the CRUD operation.

3.10 Design Philosophy

Design is important part of any system and can in many cases, can determine a system's longevity. In this case, as a requirement of each phase a requirement of the realization of the application was a design artifact. In all cases, a basic UML or class diagram was used to reflect the changes in the design as the project was developed. Although, by the end of development, the diagrams became quite complex, it was necessary to make them complete should this code ever be extended or enhanced.

3.11 Testing Conventions

One convention in the Rails community is to avoid repeating yourself. This is especially true in the testing aspects of Rails. Testing takes a lot of time in the development cycle and it is something that should only be performed on code that you personally created and not generated code, or code that is brought into the project via plugins or code that is included for enhancing functionality this includes any gems that are installed. The assumption here is that the creators of any imported code follow good conventions and testing best practices as well and thoroughly test their code before it hits the general public. In accordance with this, you only test the code you develop, not add-ons or functions that Rails provides off the shelf. Following this principle makes the development and testing much lighter from a maintainability standpoint.

4 RELEASE 111

The primary focus of this release was the prework and infrastructure items, and a single release of the WebChecker program. The single release of WebChecker includes testing suite including unit tests and functional tests.

4.2 Sprint 111-1

This sprint was primarily concerned with the infrastructure and prework elements. This includes tool selection, architecture configuration and the initial design of the WebChecker Agent.

4.2.1 Tool Selection

During this release, several tools were decided upon, including planning tools, development environment and the back end database.

Planning Software ScrumPy was the scheduling software that was selected for scheduling and project management. As noted above, ScrumPy is a simple java application that installs on the developer workstation with very little setup. Once installed, the application was intuitive to configure. The work items were entered assigned proper sizing very quickly.

Language Platform As mentioned before, language was an easy choice. Ruby, with its popularity, its open source nature and solid community support was a natural fit. Rails extends

the ruby selection and assists with the persistence mechanisms that will be required in the later releases when the database elements are introduced.

4.2.2 Infrastructure Milestones

The following section details some of the implementation aspects of the infrastructure including the physical server hardware, the Ruby and Rails environment, and the actual http server implementation that is required for serving the rails environment.

Installation of the Server As mentioned above, the sever installation was simple as the hardware was a stock 2.4 GHz Pentium 4 class system with 1GB of RAM and was fully capable of running a modern linux distribution. For this project, Debian 5.0 Linux (also known as "Lenny") as mentioned, was selected for its reputation for stability.

Installation of the Ruby and Rails Environment Ruby and rails proved to be more difficult than initially suspected. This is not a because of Ruby, Rails or the underlying operating system, but due to some constraints external to the scope of this project. These constraints require some versions greater than the built in Debian package system provides. When building the environment and incorporating the constraints, the built in package management system of Debian failed to remove some of the built in Ruby hooks and symbolic links. Since the Ruby and Rails installation for this environment was being built from source, a filesystem cleansing of these links was required to make all subsequent layers work together.

After a manual build for source and installation and some manual symbolic link creation, the system was ready. Ruby has its own package management system that is used to install gems that are essentially add on packages to enhance the ruby environment. Rails and its dependencies are all installed through the gem manager.

Installation of Nginx/Passenger Phusion After some research the Nginx http server, was found to be one of the best http servers for Ruby/Rails around. Passenger Phusion is a proxy and bootstrap system that works well in getting Nginx up and running quickly. The entire package is natively compiled and is a very responsive platform. A test application was raised for verification of the http installation in short order.

Initial Design stages of the project, it was somewhat unclear as to what the initial objects would look like. At this point, the design was a product of what the end objects "should" look like. That is, as responsibilities go, it was clear what roles were going to be in play. The initial Class diagram was developed from this standpoint.

4.3 Sprint 111-2

There was, in fact, a significant amount of discovery that needed to take place as the project evolved. Some of these

items were as simple as Ruby syntax and code structures. Other, more complex items surrounded the libraries that were available for use in the Ruby environment. So there was much evolution of this code over the sprint.

4.3.1 WebChecker Implementation

The first pass with the sprint was to essentially get a functioning prototype up and running. The first implementation was essentially a sequencer that would drive a series of methods within the same class in a rough draft of the core logic. Each of these methods would do their small part of the process using whatever bits of the Ruby libraries that could be used to get the job done. At the end of the exercise, there was, in a sense, a functional script that would do most of the tasks that were required for this iteration. In the next pass, it became apparent what would be required to make this code reusable and more modular. In this step, the methods were ported from the initial prototype into full blown objects. This naturally, required some refactoring of the initially planned objects.

4.3.2 Testing for WebChecker Agent

Acceptance Test At the end of the first sprint, the WebChecker agent was completely developed. This agent was able to be run from the operating systems cron table on a schedule and poll a series of websites. An administrator had the ability to set these web pages (both http and https) and regular expression verification points to ensure that the content of a site was being served properly. Administrators could also specify an email address that could be used for WebChecker to email failure reports. In this increment, all of these settings are contained in comma separated files. While primitive, the agent could be released in this form, as it does deliver business value.

Functional Testing Functional testing for this increment was handled by manipulating the comma delimited input files. Test cases were generated by determining the different input factors that ultimately change the way the program builds the target urls. PageValidator uses mocks as well, this time for network calls. In this test suite, various inputs of the UrlToCheck objects are fed to the unit. Positive and negative paths are tested. There are seven tests that exercise this unit.

5 RELEASE 112

The focus of this release is the enhancement of the core that was built in the first release. Specifically, in the first release, there were some brittle input methods that required special syntax to avoid input errors. In this release a Web based Graphical input mechanism will be created, with a back end database storage system for persistence. Sprint 112-1 is focused on designing and developing the interface, refactoring the core agent code from 111, and integrating the core agent code with the new persistence mechanisms. The next few sections discuss some key concepts that are required about the

nuances of Ruby on Rails development.

5.2 Sprint 112-1

In this sprint, the primary focus was contacts management. Contacts are simple objects that consist of first name (fname), Last name (lname) and email address (email).

5.2.1 Expanded Class Diagram

In the designed class diagram, there are 4 classes that represent the view classes (edit, new_contact, index, and show) along with the contacts_controller class. The Contact class now represents a model of database elements that are persisted in a sqlite 3 database. The Primary focus of this release was to make the management of the core inputs easier. As you recall in the initial release of 111, the inputs to the core agent was handled by a series of comma delimited files. While crude, it was effective but by most standards, susceptible to errors and typos. To fix this, a means to standardize the input methods was required. Development was handled in two separate stories: The contacts management story and the web location management story. While similar in design, each had its own nuances and it was decided to make each story a focus of a sprint.

5.1.3 Design

Previous to this step in the process all development was done in Ruby and not specific to the Rails framework. This was the first real step in the project where the kit that Ruby on Rails provides including the rails generation scripts, Model-View-Controller based architecture, database persistence and restful interactions with the web. Given the conventions that the Rails framework provides, the design was fairly simple. The contacts model would be a table in the database that would hold the string fields that are mentioned above. Scaffolding would create the required elements to provide all Create Review Update and Destroy (CRUD) operations in the view and controller. The rails framework would create the back end tables and model object.

As for integration, the factory pattern that was developed in the first release worked in our favor here, it was trivial to change the getNotificationList method that was already implemented in the factory and change it into a call that would access the database directly. Ultimately, this is the class diagram that represents the classes from this sprint and how it would be wired in with the previous released code.

6 RELEASE 113

This sprint's focus was to enhance WebChecker to collect response data and devise a logical manner to display this data in chart format.

6.2 Design

For this sprint there were two core problems that needed to be tackled: data collection and data representation. The design is therefore divided into separate pieces, data collection and data presentation.

6.1.1 Data Collection

For data collection, further expansion of the database needed to take place. A table to store data about response times needed to be created and integrated with the front end agent that does the gathering. The class diagram was designed and particular interest was the PageValidator class and the response.rb file. The file named response.rb is the data model that represents the underlying table named responses in the database. This response class is also linked to the web_location record of the web site that is being polled. Notice that web_location has been re-factored as well to move the web_response information directly into the response table to further normalize the data. The design is that the PageValidator class will poll each site tracking the required response information, when the poll results are received from the server, the PageValidator makes a call to the Response data object and to the create method on the model object with the the relevant parameters. When the create action occurs, the web_location.id will be passed as well, which is used to link the response to a particular website. When it comes time to display the data, the has_many, belongs_to relationships will join the elements together for grouping calculations, etc.

6.1.2 Data Representation

For representation, there is the temptation to over think the representation and begin to develop complex javascript, dojo or Adobe Flash based solutions. This ultimately leads to a lot of logic placed into the view that can be difficult to test. The approach that was taken was somewhat different : the chart api from Google.

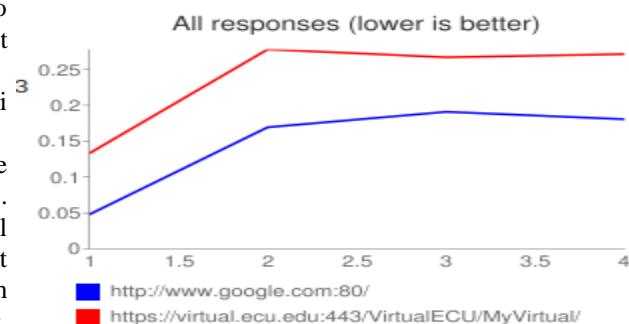
Google has made an API freely available that generates the charts and graphs that are easily integrated into websites. Essentially, what the programmer must do, is integrate a url into an img tag on the page that gets served. When the client displays the page, a request is made to the chart api url from Google with all the relevant data integrated into the img tag. Google's server uses the parameters that are passed via url and creates the chart on the fly. This has the advantage of moving the logic out of the view, and actually out of the application altogether. This does place a dependency on an external entity, but being that Google has a good track record for availability, and the nature of the dependency (few graphical elements), this risk is acceptable.

6.1.3 Implementation

Data Collection Implementation of the data collection for this design was not significantly difficult given that the Rails environment is initialized every time that the agent is run. So,

for this integration, a change was made to the loop, where the polling takes place to add the start_time and stop_time variables along with the addition of single line to create a response record. This single line that has been truncated for readability creates a new record in the response table with all the relevant variables applied. The web_location.id is the binding factor in the database to link the two tables along with the appropriate has_many belongs_to relationships. So now whenever web_location is selected from the database, all of the responses that belong to that same web_location.id will be pulled in as well. This is especially useful for the Data representation phase of this sprint.

Data Representation Implementation To integrate the chart representation, an adapter was implemented that would assist in the url generation that would be placed inside the img tag in the view. This object was naturally written in Ruby, and takes two or three arguments dependent on where the adapter is used. For the index pages, the the controller passes in a title for the chart, a data set to be represented, and the name of the websites that are placed on the chart. In this way, the adapter unfolds all the title, data and names are presents this into a single line url. For the individual pages, the data set is less complex and different type of chart is required. The representation for these types of calls is handled through the chart adapter object as well, where the Title of the chart is passed in along with the data set that will be charted. In either the large collections or the individual representations, the adapter object is passed to the view on the original http get request, and the url method is then called to write the img tag properly. When the page is rendered, Google's server provides the required graphic. See below for a sample chart.



6.1.4 Testing

Data Collection Testing Testing for the Data Collection aspect was handled via unit test in RSpec. Testing was as simple as building a mock object of the WebLocations model, and creating a few expectations on it. This mock was placed in an array and passed to the PageValidator initializer and finally in the test we set an expectation on the Response table to receive exactly one call on the create method. The test consists of fewer than 15 actual lines of code:

Data Presentation Testing The testing for the presentation was a little more involved. Since there are two modes of

operation for the representation, there are a few items that require testing. First a test of each mode was required, then a test to make sure that the `chart_adapter` object that provides the url for the `img` tag is being properly presented to the view.

To test the chart adapter itself a set of tests were created that would exercise the various modes of operation that the adapter was required to provide. To do this, 2 known data sets are constructed for the test- a set that would be used as the representative set for the individual data points. This data would represent the single web site. For the set of web sites, a few more items were created. A set of names, and a set of data points. This represents the multiple sites line graph above.

In the BDD model the test was not overly complex:

*Given these data points,
when put into the chart adapter,
then the response on the url method will be X
(Where X represents the oracle that is hand derived
by the tester.)*

After the chart adapter was fully tested at a functional level, all that was left was the integration testing. In this case, the test that was required was to make sure that the chart adapter object was getting passed to the view properly. Since the passing of objects from the models through the controller is already something required for other objects in the existing system, it was a simple matter of enhancing the existing tests to ensure that the chart object was getting passed to the view for rendering as well.

7 Conclusions

Agile development is typically a team effort. Agile teams can be various sizes but typically, they are comprised of 2 or more individuals. Single members or small sub teams have selected focus but they are part of the larger project effort. This project was unique in that the “team” was a single individual. Rather than simply toss a few stories together, the project and plan was iteratively groomed together. There was cohesion to the stories as they were developed and real meaning was associated with how the stories were grouped, sized and planned. While this approach may seem a bit heavy in nature for a team of one, the structure of it prevented so called “cowboy coding” and ultimately lead to a higher quality of code.

Certain Agile conventions like check point or stand up meetings were deemed unnecessary, other items like the Burndown chart and the large scale written updates at the end of a release proved valuable in the progress of the project. The burndown charts lead to the internal perception that work was progressing at the expected rate. There was a great sense of accomplishment in marking the stories complete and driving the status bar to the right and down.

Large scale written updates near the end of each release were used to document what had transpired. These served as a good verification point and a good sanity check that everything had indeed been completed properly. During Sprint

111-2 for instance, while writing an explanation about the testing procedures, it was deemed that a certain class had no unit tests written for it. This class had been overlooked with regard to unit testing, and the write up procedure actually caught the problem and the tests could be completed before the next set of work was started.

All aspects of the project were completed on time and the project met all of the initial requirements. Agile, with its incremental and iterative nature lent itself to this project nicely. Ruby on Rails was a viable solution for this particular application and the available testing frameworks were more than adequate for the task.

Perhaps most interesting, is that other than the freeware scheduling application, ScrumPy, the entire software stack was comprised of open source solutions. This includes everything from the development environment to the Debian powered server and the other web service components and the core languages. Clearly this is an inexpensive solution that could be extended for cloud based computing and other web based services in the future.

8 References

- [1] Pham, Andrew, and Pham, Phuong-Van 2012 *Scrum in Action: Agile Software Project Management and Development*. Cengage Learning
- [2] Larman, C. and Basili, V.R. 2003. Iterative and Incremental Development: A Brief History *Computer*, vol. 36, no. 6, pp. 47-56
- [3] Department of the US Air Force's Software Technology Support Center, 2003 *Guidelines for Successful Acquisition and Management of Software-Intensive Systems: Weapon Systems Command and Control Systems Management Information Systems Condensed Version Version 3*. Department of the Air Force, Hill AFB, Utah
- [4] Larman, C. and Basili, V.R. 2003. Iterative and Incremental Development: A Brief History *Computer*, vol. 36, no. 6, pp. 47-56
- [5] Larman, C. and Basili, V.R. 2003. Iterative and Incremental Development: A Brief History *Computer*, vol. 36, no. 6, pp. 47-56
- [6] Fowler, Martin and Highsmith Jim, 2001. The Agile Manifesto *Software Development Magazine*
- [7] Larman, C. and Basili, V.R. 2003. Iterative and Incremental Development: A Brief History *Computer*, vol. 36, no. 6, pp. 47-56
- [8] Leffingwell, D. with Behrens, P. 2009 *A User Story Primer*, Leffingwell, LLC.
- [9] Cohn, M. 2005 *Agile Estimating and Planning*, Mountain Goat Software, LLC
- [10] Cohn, M. 2009 *An Introduction to User Stories*. Mountain Goat Software, LLC
- [11] Pekka Abrahamson, Outi Salo, Jussi Ronkainen & Juhani Warsta, 2002 *Agile software development methods Review and analysis*, VTT PUBLICATIONS, VTT, Finland
- [12] ScrumPy, 2011. <http://www.scrumpytool.com/>

Estimating Agile Iterations by Extending Function Point Analysis

A. Udayan Banerjee, B. Kanakalata Narayanan, and C. Mahadevan P

NIIT Technologies Ltd, No.31/2, Rupena Agarhara, Hosur Main Road, Bangalore-560068, India

Abstract - Estimation is critical to software development irrespective of the development methods being used. Waterfall methods work on the concept of signed off requirements, while agile methods are designed to handle changing requirements through increased customer participation and frequent releases. Statistical estimation methods like Function Point Analysis (FPA) are more appropriate in scenarios where requirements are explicitly documented, while agile projects are typically estimated using analogous non standard sizing methods like Story Points.

Organizations outsourcing software development want vendors to adopt agile methods but estimate using standard techniques like FPA and upfront commit to schedules, features, effort. The key characteristic of agile projects which impacts estimation is its iterative incremental lifecycle which includes evolutionary design, requirement refinement, increasing code base and constant code refactoring. In this paper we propose a mechanism for estimating the size of agile iterations in Function Points by extending FPA [11] techniques along with Caper Jones activity scope for software projects [17, 20]. We have applied this technique on three agile projects and observed that there is a linear correlation between effort consumed and the estimated iteration size.

Keywords: Agile, Function Points, Estimation, Iterative Development, Outsourcing.

1 Introduction

As an answer to the challenges of modern software development, different lightweight approaches have been established since the mid 1990s that can be subsumed under the brand Agile Methods [1-2]. They “allow for creativity and responsiveness to changing conditions” [3]. They also emphasize on customer participation, quick reaction to requirements’ changes and continuous releases. These methodologies are gaining in popularity as preferred means for developing software as they allow organizations to deliver software effectively in a changing environment [4].

Agile methods specify that working code should be delivered in small pieces iteratively catering to a sub set of the functionality asked for by the user. With every iteration, users are encouraged to provide feedback, add, remove, change requirements based on which the subsequent code is refined and incremented. The main idea behind this approach is that through emphasis on working code delivered frequently there is a greater chance of delivering usable software which provides business value. [4]

Software estimation is a critical component of software development, irrespective of the development method being adopted. Estimation defines the transformation of requirements, skills, people and equipment into cost and effort [5]. The main software estimation techniques are the following:

- Analogy based: where a new project is estimated based on its resemblance to an existing project,
- Expert opinion: where a group of experts gather together to come to a consensus on how much time is required to build a piece of software,
- Lines of code based: where the estimate is arrived at based on the expected lines of code,
- Bottom up methods like work breakdown structure where each task required for the project is estimated and the sum of it is the total effort for the project,
- Statistical methods like Function Point Analysis which quantify the size of the software rather than estimate the effort directly. These methods use metrics collected from past projects along with mathematical formulae to estimate project costs.

Each method has its advantages and disadvantages which are well researched and documented. Statistical methods offer a scientific approach to software estimation, as compared to the other methods, which are more subjective in nature, with the exception of lines of code based sizing. These methods are more preferred when software development is outsourced by organizations to vendors mainly because they help in quantifying software in a standard way irrespective of the technology being used and such estimations can be independently verified. The main disadvantage of statistical methods is that they require the specifications to be articulated in a detailed manner to provide accurate estimates [6]. Agile development projects on the other hand are characterized by fuzzy or evolving requirements. They are typically estimated using a combination of analogous methods along with expert opinion. Agile processes recommend that estimations are best done by the team executing the project, and revisited each iteration, using a sizing metric evolved by the team. Story Points or Ideal days [7] are most popularly used in this respect. The team looks to past projects or iterations, and draws on its own experiences to produce estimates [7,8]. Caper Jones [9] states that one of the agile weaknesses is a widespread failure to measure projects using standard metrics such as function points.

The purpose of this paper is to propose a mechanism for calculating the size of agile iterations as Function Points accommodating for iterative incremental development by extending Function Point Analysis [10] techniques along with Caper Jones activity scope for software projects [11,12]. While there have been papers published trying to establish a theoretical relationship between Function Points and Story Points [13, 14], we have gone one step further and tried to establish a working model for sizing agile iterations using FPA, in the outsourcing context. Through this we present a standard and consistent approach to sizing agile iterations.

2 Estimating Agile Projects with Story Points

In agile projects the features to be developed are expressed in the form of user stories [15] and one of the popular methods of sizing stories is using Story Points, a subjective unit of estimate derived by agile teams. In this method a team compares a user story to one or more similar stories and gives the story a size in terms of 'Story Points' or 'Ideal Person days'. The number of story points associated with a story represents the overall size of the story. There is no set formula for defining the size of a story [13]. Each team defines story points as they see fit. One team may decide to define a story point as an ideal day of work and another team may define a story point as a measure of the complexity of the story [13]. Story points have emerged as industry best practice for measuring an agile development team's velocity i.e. the number of user stories delivered in an iteration [7].

The stories that may be taken up by a team in an iteration is dependent on the experience of the team, the cohesiveness of the team, the knowledge they have on the product, the technology complexity involved etc. These numbers vary from team to team. Michael Cohn [16] says that it is very difficult to establish a direct correlation between story points and hours and further says that the relationship between story points and hours is typically a distribution centered on a mean (Figure 1). Even for a given team, same story point sized stories may take different times during different points in the release life cycle. A team which has been working for a long time on a specific product may be able to deliver more stories than a newly formed team [13]. Thus while story points provide a way for agile teams to flexibly estimate user stories, it is not always possible to extend these metrics across teams or at an organization level [17].

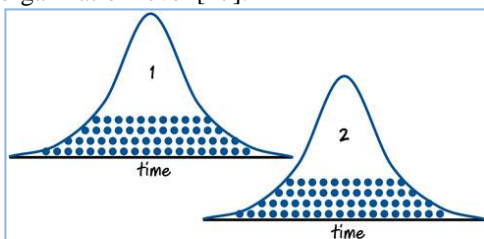


Figure 1: Hours to develop a 1 and 2 point story [16]

3 Challenges of Estimating Iteration Size Using Statistical Methods

In this section we attempt to show how the very nature of iterative development makes it difficult to apply traditional statistical methods like FPA to estimate projects. Each agile iteration is a mix of new stories, refactoring of existing stories, testing and bug fixes to existing stories. It is not executed like a mini waterfall based project where a requirement is executed, completed and signed off with a fixed schedule in a single iteration.

One of the initial activities done at the start of a project or release is Release Planning, where the development team along with the customer or Product owner get together to understand the requirements (product backlog) and estimate roughly the size of the requirements and the number of iterations it will take to fulfill them. Team commits to a probable release date and the best and worst case list of features that may be released by the release date. The team may use Story Points to size the release or use statistical methods like Function Points depending on the extent of clarity they have regarding the requirements.

Iteration Planning and the Definition of Done: Every iteration, the team commits on the number of stories that can be accommodated. While agreeing to develop stories, the team formulates a 'Definition of Done' (DoD) which is a list of activities that will be performed by the team in that iteration towards the selected stories. DoD is a simple list of activities (writing code, coding comments, unit testing, integration testing, release notes, design documents, etc.) that add verifiable/demonstrable value to the product and can be undertaken in an iteration [18]. The team may formulate a definition of done for the release which is a super set of the DoD for an iteration. For example a team may decide that they would leave integration and stress testing (which is in scope for the release) for later iterations and only take up unit testing and functional testing on each story in the current iteration. Based on the activities pending, teams may later visit user stories which were developed in initial iterations to complete and polish them the extent needed to make a formal release.

Hence, the key problem in estimating the size and the effort required in an iteration using FPA is that all activities required to be completed towards a function or user story in a project lifecycle may not necessarily be taken up in a single iteration. The same story may be visited several times over subsequent iterations either to refine it or to add more complexity or undertake additional activities like end to end testing.

Change Management in agile Projects: In order to accommodate change, agile methods recommend that the customer or product owner is able to re-prioritize stories, introduce additional complexity or add new stories into the product backlog. During the iteration planning meeting the team is expected to understand the new prioritized backlog

and decide which items they can take up in the iteration as per the DoD. Hence in an agile project life cycle estimation is something that frequently occurs and is continuously revised and updated.

Evolutionary Design: While agile teams deliver working code in each sprint, they do not generally have a well defined design phase as in waterfall based projects and instead work around evolutionary design practices. Most teams start by having a basic working design which is refined and re-factored as the project progresses. Sometimes more than one design alternative may be tried out, which could cause significant code changes to user stories which are already developed in early iterations [4]. Another reason for design or code refactoring may be driven by business or regulatory needs which may demand adjustments in delivered code. Again this is an aspect affecting estimation of an iteration.

Testing and Bug Fixes: In agile projects the code delivered in a iteration is tested by the product owner and bugs may be recorded which are typically taken up by project teams in subsequent iterations. Most agile teams have dedicated iterations where they do not accommodate any new user stories, but only work on refining and bug fixing on existing user stories in order to make them release worthy. This again adds another dimension to the estimation process.

Complexity is handled by most agile teams iteratively. So a complex user story may be broken up to show a simple working code which is iteratively enhanced with each iteration. An example of this is providing a multi language capable website. In early iterations a single language page may be developed, which is subsequently enhanced and tested for multiple languages..

Hence, upfront committing to a function point count to be delivered in a iteration is very difficult for an team as all these varied dimensions are also to be considered.

4 Estimation in the Outsourcing Context

Outsourcing with off-shoring software projects is a popular trend in organizations whose core activity is not developing software with the chief motive being cost reduction [19]. Most companies outsourcing software use competitive methods to request quotes from vendors and chose the vendor who most closely meets their expectations in terms of cost, quality, skill levels etc. The pre-dominant estimation method preferred in an outsourced scenario is Function Point Analysis (FPA) with the development process being waterfall based and vendors working on the concept of formal signed off requirements.

FPA is very popular when development work is outsourced because it is a standard technique [10] and is considered a scientific approach to sizing software and an absolute metric which can be computed, irrespective of the team executing the project. The organizational productivity benchmark

typically computed in hours per function point (technology specific) is applied to the Function Point count to arrive at the effort and the schedule for a project. It enables companies to verify if the vendor estimates are realistic and within accepted range.

4.1 Agile Estimation in an Outsourcing Context

With agile development methods gaining popularity organizations want to realize the benefits of such methods while continuing with the trend of outsourcing. Organizations have started expecting their vendors to execute projects using agile development techniques.

Given their variable nature it is difficult to fix scope, budget and schedule in agile projects. It is recommended to work with a fixed budget or schedule keeping the scope variable or within a range of possible features that could be delivered [20]. However, these techniques work well only if the projects are in-house developments of the company concerned or if there is a high degree of trust between a customer and a software vendor. In a competitive situation, organizations expect vendor companies to provide an upfront effort estimate and commit to schedules, features and resources even while developing projects the agile way. These estimates may have to be done during contracting and much before a team is assembled to execute the project. Since story points are not counted by scientific methods, they are not accepted as a credible estimation technique in this context and customers typically want function point based estimates.

Offshore vendors assemble bid teams consisting of representative members to help in creating estimates for agile projects during contracting stage either using Function Point Analysis or work breakdown structure or other suitable methods.. Such estimates are at a macro level and may vary significantly once the project execution commences and micro level details are obtained.

A common problem during project execution is the expectation on the team to produce iteration level estimates. It is possible that members of a team assembled for a project and may not have worked with one another before or may not be ones with similar experience or may not have worked in the problem domain or technology [4]. This problem is further complicated when there are multiple agile teams executing a large project. So teams may not be skilled enough to produce estimates and with the absence of organizational metrics or a standard way to estimate the iterative development process, it is very difficult to size agile iterations with team inputs.

Another aspect that agile methods implicitly assume is an atmosphere of trust. They assume that developers truthfully estimate for stories and as iterations progress, take up more and more stories and increase their pace of working and become better at estimating [21, 22]. Secondly customers or product owners are expected to believe in the subjective developer estimates and give the team enough time to achieve a predictable pace of working and delivering user stories.

This is again a difficult situation in a competitive vendor customer relationship.

5 Proposed Solution

Since function point based counts are standard and accepted we propose in this paper to extend the techniques of function point analysis to help in estimating agile iterations. The key considerations being

- Accommodate for increasing complexity and iterative design
- Accommodate for the same story to be worked upon in multiple iterations based on the Definition of Done

We have used FPA method of calculating project complexity using a value adjustment factor [10,23] and the Caper Jones suggested activity scope percentage for software projects [11, 12] to quantify the size of the stories taken up in agile project iteration in the form of Function Points. Through this we propose to provide a scientific basis for computing the size of an iteration, such that the estimates can be verified, validated and defended.

5.1 Function Points and General System Characteristics

Function Points are counted in a two step process. The first step is to classify each feature in the system against one of the major functions i.e. External Inputs / External Outputs / External Queries / Internal Logical Files / External Interfaces and arrive at the function point count, which is called the raw or unadjusted function point [10], [23]. The raw function points are adjusted by computing a value adjustment factor based on the possible impact of a set of fourteen general system characteristics (GSC) of the system to be developed. These factors are listed in for reference in Table 1 [10, 23, 24]

GSC	Description
Data Communications	How many communication facilities are there to aid in the transfer or exchange of information with the application or system?
Distributed Data Processing	How are distributed data and processing functions handled?
Performance	Was response time or throughput required by the user?
Heavily Used Configuration	How heavily used is the current hardware platform where the application will be executed?
Transaction Rate	How frequently are transactions executed

GSC	Description
	daily, weekly, monthly, etc.?
On-line Data Entry	What percentage of the information is entered On-Line?
End -User Efficiency	Was the application designed for end-user efficiency?
On-line Update	How many ILF's are updated by On-Line transaction?
Complex Processing	Does the application have extensive logical or mathematical processing?
Reusability	Was the application developed to meet one or many user's needs?
Installation Ease	How difficult is conversion and installation?
Operational Ease	How effective and/or automated are start-up, back-up, and recovery procedures?
Multiple Sites	Was the application specifically designed, developed, and supported to be installed at multiple sites for multiple organizations?
Facilitate Change	Was the application specifically designed, developed, and supported to facilitate change?

Table 1: General System Characteristics

Each characteristic has associated descriptions that help determine the degrees of influence of the characteristics. The degree of influence ranges on a scale of zero to five, from no influence to strong influence. The IFPUG Manual provides detailed evaluation criteria for each GSC [10]. The GSC is scored based on their influence on the system being counted and this provides the value adjustment factor. The unadjusted Function Point count is multiplied by the value adjustment factor to arrive at the Adjusted Function Point count. The resulting score can increase or decrease the Raw Function Point count by up to 35% [10, 23, and 24].

The GSC can be adjusted to size code complexity by varying the degree of influence of the relevant parameters. For example the FP size of a feature's adherence to performance guidelines may be estimated by varying the degree of influence of the 'Performance' GSC. A re-usable application having the requirement to be configurable or having the ability to be installed in multiple sites (or tested on multiple devices for a mobile application) may be sized by varying the

influence of 'Multiple Sites' and/or the 'Facilitate Change' characteristic.

5.2 Activity Scope as described by Caper Jones

Caper Jones in a paper [11, 12, 25] has described that software projects include many more activities than just coding or programming and has published a list of activity patterns for different kinds of projects. This is a list of around 25 typical activities that are undertaken in software projects and the percentage of effort associated with each activity. He recommends that teams understand which of the most likely activities would be performed in a project and use the activity effort percentage as a guide to estimating software projects. (Ref: Table 2)

Table 2: Caper Jones list of the 25 most applied activities in Software Projects with their % contribution to the estimate

Activity	% Weightage
Requirements	3.84
Architecture	2.25
Project Plan	1.33
Project Management	6.75
Initial Design	3.84
Prototype	4.5
Detail design	4.5
Design Reviews	3.02
Coding	13.5
Unit testing	4.5
Configuration management	0.41
Code inspection	4.5
Formal integration	2.71
Functional testing	4.5
Integration testing	3.84
System testing	3.38
QA	4.5
Field testing	3.02
Independent verification & validation	5.42
Independent third party test	3.38
Acceptance testing	1.94
Installation & training	1.94
User documentation	9.67
Reuse acquisition	1.13
Package purchase	1.63
Total	100%

5.3 Extension of 5.1 and 5.2 to accommodate Agile Iterative Development

Our proposition is to use the concepts in section 5.1 and 5.2 to express the size of an agile iteration in Function Points.

Release Planning: Compute the size of all the stories in a release using Function Points based on the information available and estimate the effort / schedule using the organizational productivity baseline creating the macro level estimate.

Definition of Done for a release: Discuss and come to an agreement on the definition for done (DoD) for a release and map it to the activities as per Caper Jones activity scope. The weights given by Caper Jones are an indicator and they may be adjusted as each team sees fit. Table 3 shows definition of done for one of our reference projects. The percent of the applicable activities in our sample project was 87.57% and we normalized the same to 100%.

Table 3 : Activities as applicable to a reference project for a release

SlNo	Activity Group	Activity	% Weightage	% Applicable	Normalized %
1	Requirements	Requirements	3.84	3.84	4.4
2	Architecture	Architecture	2.25	2.25	2.6
3	Planning	Project Plan	1.33	1.33	1.5
4	Planning	Project Management	6.75	6.75	7.7
5	Design	Initial Design	3.84	3.84	4.4
6	Design	Prototype	4.5	4.5	5.1
7	Design	Detail design	4.5	4.5	5.1
8	Design	Design Reviews	3.02	3.02	3.4
9	Coding	Coding	13.5	13.5	15.4
10	Coding	Unit testing	4.5	4.5	5.1
11	Coding	Configuration management	0.41	0.41	0.5
12	Code Review	Code inspection	4.5	4.5	5.1
13	Integration	Formal integration	2.71	2.71	3.1
14	Testing and QA	Functional testing	4.5	4.5	5.1
15	Testing and QA	Integration testing	3.84	3.84	4.4
16	Testing and QA	System testing	3.38	3.38	3.9
17	Testing and QA	QA	4.5	4.5	5.1
18	Testing and QA	Field testing	3.02	3.02	3.4
19	Independent Testing	Independent verification & validation	5.42	5.42	6.2
20	Independent Testing	Independent third party test	3.38	3.38	3.9
21	User Acceptance	Acceptance testing	1.94	1.94	2.2
22	User Acceptance	Installation & training	1.94	1.94	2.2
23	Documentation	User documentation	9.67	0	0.0
24		Reuse acquisition	1.13		
25		Package purchase	1.63		
		Total	100	87.57	100.0

Sizing an Iteration: In every iteration, an agile team works on new stories and existing stories. The size of an iteration is the size of the quantum of work done in the iteration. We propose to size an iteration as follows:

- Compute the total size of the stories in an iteration in Function Points
- Identify the percentage of activities to be undertaken towards new and existing stories, i.e. the Definition of Done (DoD) for the New stories and DoD for existing stories in an iteration as mapped to Caper Jones applicable activity scope (DoD for the release)
- Apply this percentage to the total size of the stories to arrive at the iteration size.

DoD for New Stories: As explained in section 3, the team may not necessarily undertake all the activities related to a

story in the same iteration. New story refers to the first time a story is worked upon in an iteration.

Table 4: Activities as applicable to new stories in a iteration

Activity Group	% Weight	As Applicable for New Stories (%)
Requirements	4.4	4.4
Base	2.6	
Planning	9.2	9.2
Design	18.1	12.7
Coding	21.0	14.7
Code Review	5.1	
Integration	3.1	1.5
Testing and QA	22.0	11.0
Independent	10.0	
User Acceptance	4.4	
Total		53.5

For example Table 4 depicts the activities that were undertaken towards new stories in an iteration for a reference project. Since the applicable activities are 53.5% of the total activities to be undertaken for the project, the size of the new stories has been measured as 53.5% of the final size of the same stories that were to be delivered as part of the release. The rationale for only undertaking 53.5% of the total work in the specific iterations is as follows.

With reference to Table 4: Only a percentage of the design as delivered at the end of the project was undertaken in the iteration and this was refined in subsequent iterations. This also meant that code towards realizing the design was also spread across multiple iterations with bulk of the initial coding being done in the current iteration. Similarly code review for new stories was formally done in subsequent iterations hence this activity was not sized in the current iteration. Testing for the stories was carried out across iterations with about 50% of the testing activity being undertaken in the current iteration and Independent testing being taken up in the subsequent iteration. The remaining 46.5% of function points remaining towards realizing the same set of stories for the release were spread across the remaining iterations. This number is not a fixed percent but an example to depict the iterative development cycle.

DoD for existing stories: Every iteration team would also be working on stories delivered in earlier iterations, either for refactoring code on account of design evolutions or an account of testing and bug fixes. Again using the Caper Jones activity scope identify the relevant activities applicable for the iteration towards existing stories (DoD of existing stories) and use the percentage to revise the size estimate of the stories.

Table 5: Activity break up for a set of stories across multiple iterations

		Iteration N	Iteration N+1	Iteration N+2	Iteration N+3	UAT
Size using FPA		118.81	118.81	118.81	118.81	118.81
Applicable Size in FP		115.60	115.60	115.60	115.60	115.60
Requirements	4.40%	4.40%				
Design	18.10%	12.67%	2.72%	2.72%		
Coding	21.00%	14.70%	3.15%	3.15%		
Code Review	5.10%		2.55%	2.55%		
Integration	3.10%	1.55%	0.78%	0.78%		
Testing and QA	22.00%	11.00%	3.67%	3.67%	3.67%	
Independent Testing	10.00%		5.00%	1.67%	1.67%	1.67%
Planning	9.20%	9.20%				
UAT	4.40%				2.20%	2.20%
Total Applicable activity	97.30%	53.52%	17.86%	14.52%	7.53%	3.87%
Applicable Size in FP		63.59	21.22	17.26	8.95	4.59

Table 5 shows as an example the activity break up for a set of new stories of total size 118.81 FP spread across 4 iterations and user acceptance testing (UAT). Since only 97.3% of the activities as per Caper Jones scope was applicable, the total applicable size is 115.6 FP. The new stories were taken up in 'Iteration N' and the code was reworked / re-factored across the next 3 iterations before it was released for user acceptance testing. Code Review for the stories were taken up in Iteration N+1 and Iteration N+2 (to accommodate for the review process and rework on account of review comments), while the design evolution took place across 3 iterations with about 70% of design being undertaken in Iteration N. Similarly testing was spread across 4 iterations, with about 50% of the testing happening in Iteration N and the remaining % spread across the other 3 iterations. Table 5 thus gives the size of a set of stories as spread across the multiple iterations in the project. This table is an example and in this manner development teams could calculate the size of new and existing stories in each iteration.

Sizing Code Complexity: As described in Section 5.2: GSC can be adjusted to size the impact of varying code complexity. We propose to size the impact of increasing code complexity as follows:

- In the initial iterations size stories with minimal complexity based on the system general characteristics as applicable for the iteration.
- In later iterations when the same stories have to be enhanced for complexity like for example tuning an application to meet performance criteria, the same story may be sized by varying the appropriate GSC.
- The size impact on account of the enhanced complexity would be a difference between the two sizes.

Base Architecture: Activity towards creating a base architecture for the application will be spread across initial

iterations and the activity percentage associated needs to be accommodated in the sizing for an iteration. Since this affects the whole release and is not dependant on a specific set of stories, its size would be a percentage of the total FP size of the release.

Change Management: Changes can be accommodated again using the same techniques as mentioned above i.e.

- Estimate the total size of the change in FP
- Calculate the spread of the change in FP across iterations based on the definition of done
- Estimate the impact of the change on other user stories in FP.

We have applied these techniques on 3 reference projects that we implemented for a client, who is one of the world's leading provider of technology and services to hotels and hotel chains.

6 Reference Projects

Our reference project's objective was to enable our client to provide their customers with hotel booking capabilities on mobile devices. This product was to be white labeled and used by their customers namely various hotel chains and properties across the world. This application was developed for deployment on Android phones, iPhone and iPad and on mobile browsers.

The development was undertaken as three separate projects on account of the varying technologies and development skills required. The activity scope of these projects included Requirement Analysis, Architecture, Design, Development and Testing including Performance Testing and Usability Testing. The client wanted the application to be developed using agile-SCRUM practices. Each project had its own team for the complete project engagement and undertook all the required software development life cycle activities. The application was to be deployed and tested on multiple devices for each project and it also had to support 3 languages (English / Spanish and Japanese). The applications had to meet stringent performance requirements. The code developed was formally reviewed by the client's technical team. Each project had 6 iterations of 3 weeks each and a user acceptance testing phase for 6 weeks. The teams were a mix of experienced and junior developers and they were working together for the first time.

We first estimated the size of the projects as delivered to the customer in Function Points (Table 6).

Table 6: Size of each reference project with total effort in FP

Project	Size in FP	Effort (Person Months)
Project 1 Android	468.2	31.8

Project	Size in FP	Effort (Person Months)
Project 2 iOS for iPhone	608.69	47.7
Project 3 Mobile Web	457.2	37.5

We applied the techniques outlined in Section 5 to size the iterations of each project. The initial iteration did not have any existing stories to be worked upon. For all the three projects, in the initial two iterations, the architecture and the reference framework for the entire application were put in place and the size of the iterations were adjusted accordingly. The size of the initial iteration was a percentage of the total size of the new stories along with the size of the percent of work undertaken towards creating the base framework. Subsequent iterations had a combination of new and existing stories. All the stories were worked upon in the first 5 iterations and the last iteration i.e. iteration 6 was dedicated towards testing and bug fixing , refining and polishing the code to make it release worthy.

Table 7: Productivity calculation for Project 1 and its variation

Project 1 Android				
	Iteration Size (FP)	Effort	Productivity	Variation
Iteration1	55.94	4.20	12.0	11%
Iteration 2	53.98	4.13	12.2	13%
iteration 3	80.86	5.7	11.2	4%
Iteration4	87.00	5.6	10.2	-5%
Iteration5	91.44	5.5	9.6	-11%
Iteration6	75.07	4.4	9.4	-13%
UAT	23.88	2.3	15.3	
	468.2	31.8	10.8	

We checked to see if the effort expended was comparable to the size computed using our methods. We calculated the productivity of each iteration and checked for the variation. Table 7 shows the size of each iteration (including UAT) and the productivity variation for Project 1 while Table 8 , Table 9 show the same metrics for Project 2 and Project 3.

Table 8: Productivity calculation for Project 2 and its variation

Project 2 iOS				
	Iteration Size (FP)	Effort	Productivity	Variation
Iteration1	55.94	4.6	13.2	8%
Iteration 2	74.4	5.9	12.7	4%
iteration 3	113.50	8.9	12.6	3%
Iteration4	117.30	8.7	11.9	-3%
Iteration5	118.90	8.7	11.7	-4%
Iteration6	97.6	6.9	11.3	-7%
UAT	31	3.9	20.4	
	608.60	47.7	12.2	

Table 9: Productivity calculation for Project 3 and its variation

Project 3 Mobile Web				
	Iteration Size (FP)	Effort	Productivity	Variation
Iteration1	55.94	5.0	14.3	11%
Iteration 2	54.0	4.9	14.4	12%
iteration 3	80.86	6.6	13.0	1%
Iteration4	87.00	6.5	11.9	-8%
Iteration5	85.44	6.5	12.1	-6%
Iteration6	70.07	5.1	11.6	-10%
UAT	23.88	3.1	20.6	
	457.17	37.5	12.9	

As we can see the productivity varies between +/- 15% of the mean showing that there is a direct correlation between effort expended in an iteration and size computed using this method. Another trend which can be clearly observed is that in the initial iterations the team was new and having lesser experience in the technologies and hence they worked with lower productivity as compared to later iterations. We have discarded the effort spent in user acceptance testing in our computation. The graphs below (Figure 2, Figure 3) also show that there was a direct correlation between the effort expended in iteration and the size of the iteration computed using this method in all the reference projects.

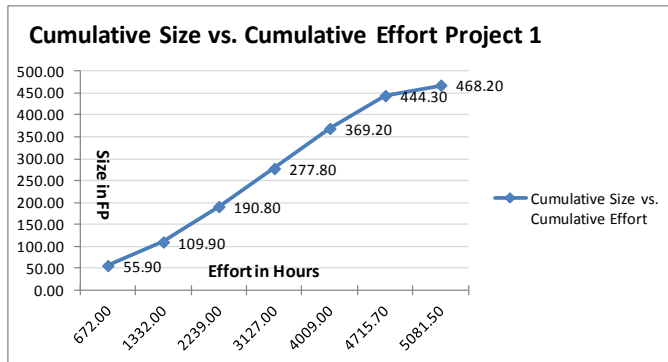


Figure 2: Cumulative Size versus Cumulative Effort for Project 1

There is a slight flattening of the curve at the top which is on account of user acceptance testing being conducted by the client and our role being limited to providing support and undertaking bug fixes.

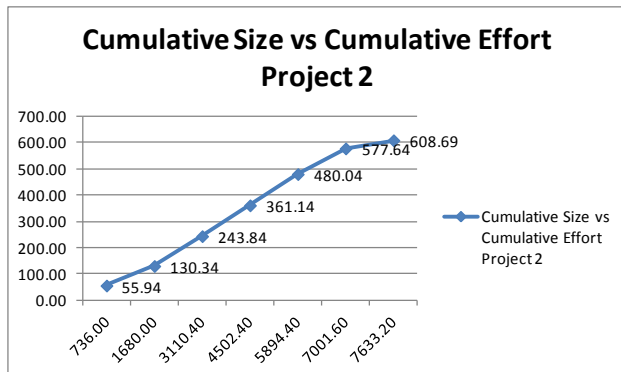


Figure 3: Cumulative Size versus Cumulative Effort for Project 2

7 Conclusion and Future Work

As we can see from the data above this approach to estimating the iteration size in agile projects has promise and needs further exploration. We have applied this method only on projects for mobile devices. We need to extend this idea to larger projects. In order to standardize this technique multiple projects will have to be sized using this method to see if organization productivity benchmarks can be computed reliably and used in new project estimations. The advantage of using this approach is that we can statistically arrive at a method of computing the number of stories which can be realistically taken up in an iteration based on the organization’s productivity baseline. However this idea needs to be verified by applying it on a live project from the outset.

In cases where requirements are not reasonably well documented in early stages, it may be useful to initially size iterations using story point methods and subsequently apply the extended function point technique to validate if the effort to stories ratio is consistent and feasible.

We have not done research on how a team’s morale may be affected due to an estimator making the estimates as opposed to the team as prescribed by the agile manifesto. We need to examine if such estimates will have the required buy in from execution teams, through independent studies.

Another area that needs exploration is the pre-game phase or iteration zero as called by some, where the initial work on a project happens like putting the team together, doing release planning, capturing basic requirements, setting up the infrastructure etc. We have not identified a method of estimating the size of this phase.

Variations of Function Point counting techniques like MK II Function Points [27] have to be further explored to see if they offer alternative methods to size iterative development.

8 References

- [1] K. Beck and C. Andres. “*Extreme Programming Explained: Embrace Change*”. Addison-Wesley, 2nd edition, 2004
- [2] A. Cockburn and J. Highsmith. “*Agile Software Development: The People Factor*”. IEEE Computer, 34(11):131–133, 2001
- [3] M. Doernhoefer. “*Surfing the Net for Software Engineering Notes*”. SIGSOFT Software. Engineering. Notes, 31(1):5–13, 2006
- [4] Experience of Executing Fixed Price Off-shored Agile Project , A. Udayan Banerjee*, B. Eswaran Narasimhan *, C. Kanakalata N *
- [5] Agile Estimation Using Functional Metrics, Thomas Cagley
- [6] Improving Estimations in Agile Projects: Issues and Avenues - Luigi Buglione, Alain Abran

- [7] M. Cohn, Agile Estimation and Planning: Addison-Wesley, 2005.
- [8] Ceschi, M., Sillitti, A., Succi, G. & De Panfilis, S. (2005) Project Management In Plan- Based And Agile Companies. Ieee Software, 22, 21-25.
- [9] Jones, Capers; Applied Software Measurement; McGraw Hill, 2nd edition 1996; ISBN 0-07-032826-9; 618 pages
- [10] <http://www.ifpug.org/>
- [11] Software Cost Estimating Methods for Large Projects, Caper Jones
- [12] Software Cost Estimation in 2002 by Capers Jones, Crosstalk magazine
- [13] Using Function Points in Agile Projects - Célio Santana^{1,2}, Fabiana Leoneo², Alexandre Vasconcelos², and Cristine Gusmão³
- [14] Using function points in XP – considerations, Andrew M. Fuqua
- [15] A User Story Primer – Dean Leffingwell With Pete Behrens
- [16] Michael Cohn – Succeeding with Agile
- [17] The Scrum Papers: Nut, Bolts, and Origins of an Agile Framework - Jeff Sutherland and Ken Schwaber
- [18] <http://www.scrumalliance.org/articles/105-what-is-definition-of-done-dod>
- [19] J Sauer. “Agile Practices in Offshore Outsourcing – An Analysis of Published Experiences”, IRIS 29, Helsingborg, Denmark, 2006
- [20] Fowler, M. & Highsmith, J. (2001) The Agile Manifesto. Software Development, August
- [21] Abrahamsson, P., Warsta, J., Siponen, M. T. & Ronkainen, J. (2003) New Directions On Agile Methods: A Comparative Analysis. Ieee, 244-254
- [22] Levy, J. V. (2003) If Extreme Programming Is Good Management, What Were We Doing Before? Edn, 48, 81-82, 84.
- [23] <http://www.softwaremetrics.com/fpafund.htm>
- [24] <http://www.qpmg.com/fp-intro.htm>
- [25] Software Engineering An Introduction – Fakhar Lodhi
- [26] Jones, C., Programming Productivity, McGraw-Hill, New York, (1986)
- [27] MK II Function Point Analysis Counting Practices Manual - 1998

Domain specific priority based implementation of mobile services- an agile way

Dr. Daya Gupta¹, Rinky Dwivedi² and Sinjan Kumar³

^{1,2,3} Computer Engineering Department, Delhi Technological University
, Delhi, India

¹ daya_gupta2005@yahoo.co.in

² rinkydwivedi@yahoo.co.in

³ sinjan.dtu@gmail.com

Abstract - In this era of globalization mobile companies are spreading their business across the world. This has resulted in a wide range of customers needs and mobile service providers are facing new challenges every day. We have examined the issues involved and found that a dynamic requirement from diverse domains is the most challenging task to be implemented while developing mobile communication software. In order to address this issue we are extending the notion of configurability in mobile communication. We will use traditional methodology for development of basic services because that is stable requirement and will act as the foundation for the further development. We will configure the on-demand services, which we will develop using agile methodology, on this basic component. In this paper we are proposing a Priority based Domain specific Mobile service Implementation (PDMI) process which helps an effective and fast mobile service implementation with high customer's satisfaction.

Keywords: Agile software methods, CASE tool, Domain specific software engineering, Requirement based Mobile software development.

1 Introduction

First generation mobile systems were introduced in the early 1980's. These systems were designed to carry narrow-band circuit switched voice services and were based on analog frequency modulation. The mobile services gained a huge popularity and to fulfill the increasing customer demand second generation (2G) digital mobile systems were developed [7]. The 2G digital mobile systems were introduced in early 90's and are still occupying the market. These systems are based either on GSM (Global System for Mobile Communication) or CDMA (Code Division Multiple Access) technologies. The third generation (3G) mobile systems were introduced after the year 2000 that allows simultaneously use of speech and data services with higher data exchange rates. Presently 4G systems are under development, mostly based on multicarrier modulation such as Orthogonal Frequency Division Multiple Accesses (OFDMA) or Single Carrier Frequency Division Multiple

Access (SC-FDMA). The application of mobile devices certainly improves and supports the lives of all age groups, but in our research we found that there exists a difference in the usage of mobile services for users of different age groups. The PDMI process proposed in this paper helps an effective, quick and customer's requirement based mobile service implementation.

Mobile domains have challenges of addressing dynamic requirements like gaming services, entertainment and lifestyle, etc. The agile methods have dynamic characteristics that help to facilitate the task of mobile software developers and to fulfill a wide range of customer's requirements in real quick time. We use light weight methods for satisfying volatile requirements for implementation purpose. The agile methods are an example of light weight methods which we have used for implementation. In addition to this we propose metric based calculation to establish that agile methodologies are suitable for dynamic requirement in mobile domain. We extend the configuration in method engineering to integrate agile method with traditional method which we are using for basic services.

In the next section we discuss popular agile methods. In section 3 we formulate our proposal for PDMI process. In section 4, we present a set of metrics for analysis of project characteristics that guide us to choose suitable methodology. In section 5, we propose to configure agile methodology with traditional method to develop services required in mobile domain. In section 6, the proposed PDMI process is implemented using view point oriented approach through agile methodology.

2 Agile Methodologies

In recent past agile methodologies are gaining popularity in project development [5, 13]. In this section we review some popular agile methodology and highlighted their underlying process, pros and cons and the guidelines to use them. Agile software development is capable of producing quality software quickly and cheaply and is suited to project with changing requirement without excessive rework.

They do not require long term planning and use the strategy of incremental development through the team work where customer is also involved. As customer is directly involved in the project development so, it minimizes overall risk and adapt to changing requirement quickly.

2.1 Scrum

The scrum method is a general agile method that focuses on managing iterative development and does not adapt specific agile practices.

It has three phases, they are discussed below:-

1. Outline planning phase for designing software architecture derived from general objective of the project. This is the management phase where first general objectives are elicited these are then used to design software architecture.
2. Sprint cycle consisting of a product backlog which is converted into sprint backlog from where product is developed and delivered in increment. The cycle is repeated until the backlog is emptied.
3. Project closure phase which terminates the project and prepares the termination report and user manual.

Once a shippable module is delivered, the product backlog is analyzed for the next priority module. Module reprioritization can also be done if required.

Pros: Scrum focuses on management activities which makes the system more adaptive. It encourages team work enabling the large and complicated project to be developed by multiple teams.

Cons: Scrum projects have poor documentation which may result in dirty programming.

Use: Best suited for large and complex product which requires lot of management activity, cooperation between product developer and product manager.

2.2 Extreme-Programming (XP)

Extreme Programming (XP) is very famous agile methodologies. It takes extreme approach to iterative development and delivers working software frequently [14].

In this, involvement of customer during development is very essential. The development team works in very close environment in presence of customer. There are six phases in XP that are discussed below.

1. Requirement Phase: - In this phase users give requirements as story that are recorded on story cards then these story cards are prioritized.
2. Task gathering: - Here stories are broken into tasks.
3. Plan release phase: - In this phase the most prioritized stories are selected and planned for early release.
4. Development Phase: - The design of the only most prioritized task, which is to be developed, is done in this phase.
5. Release Phase: - The above designed task is developed and released for the use.
6. Evaluation of the system: - working of the released system is evaluated and the next cycle is started.

Pros: It allows developer to focus on coding resulting in fast software development that satisfies the user, does not require highly technical team. Management can deliver the working software for less cost and reduces the risk.

Cons: It requires dedicated effort from client/ user side they should have good knowledge of training on XP for project to be successful. Little documentation constraints the usability of the project.

Use: It is best suitable for project which require fine programming practices and for which testing before development is essential.

2.3 Feature-Driven Development (FDD)

FDD is model driven methodology that releases software in form of features in short iterations. It is suitable for large team and consists of very short phases and delivers specific features in each phase.

Some of the Feature Sets for our project are "Incoming Call," "Outgoing Call," "Messaging," and this comes under "Basic Services" Subject Area. In FDD, we do planning, designing and building of a feature under consideration. It consists of five specific processes in specified order, which is discussed below.

1. Develop an overall model: - Requirements are gathered in top down approach where all subject areas are designed. Subject areas are aggregation of feature set. Feature set are combination of feature. Each feature is task to be performed.
2. Build a list of feature: - Features gathered are compiled to form feature list.
3. Plan by feature: - Planning is done to build a feature.
4. Design by feature: - Proper designing is done for the planned feature.
5. Build by feature: - Actual implementation of the feature is done.

Pros: It focuses on design and code inspection which results in high quality software. It provides excellent documentation and support object oriented programming which are popular among programmers.

Cons: Requires highly skilled team. Delay in release due to design and modeling activity and cost may go up.

Use: Best suited for the development of the software in which focus should be on all the features in detail such as banking, insurance and finance.

3 Priority based Domain specific Mobile service Implementation (PDMI)

We divide mobile services in various groups that act as an input to our PDMI process.

Table1: Services provided by mobile service providers

Basic Services	Incoming and outgoing calls. Messaging (send, receive) a
-----------------------	---

	message service.
Entertainment and lifestyle	Movie reviews and movie tickets. Music & videos, Jokes. Chat Services. Caller tunes. News.
Gaming Applications	Online Games. Games download.
Business Applications	Stock market news. Mobile banking. Finance News.
Traditional Applications	Astrology. Vaastu. Spiritual listening

The basic services are considered as essential for mobile communication, which comes as a default to the customers. The other services are considered as optional to the customers whom they choose as per their requirements. A survey was conducted on 400 people of different age groups wherein the respondents were asked to set a priority for the optional services according to their requirements. The summary of the results are shown in the table 2.

Table 2: Priority of mobile services for various domains

Domain	Mobile Service	Priority
5-15 yrs (Group-1)	Entertainment and lifestyle.	2
	Gaming Applications.	1
	Business Applications.	3
	Traditional Applications	4
16-27 yrs (Group-2)	Entertainment and lifestyle.	1
	Gaming Applications.	2
	Business Applications.	3
	Traditional Applications	4
28-45 yrs (Group-3)	Entertainment and lifestyle.	2
	Gaming Applications.	3
	Business Applications.	1
	Traditional Applications	4
46-62 yrs (Group-4)	Entertainment and lifestyle.	3
	Gaming Applications.	4
	Business Applications.	1

	Traditional Applications	2
63 yrs. Onwards (Group-5)	Entertainment and lifestyle.	3
	Gaming Applications.	4
	Business Applications.	2
	Traditional Applications	1

The mobile service provider will provide basic services to all mobiles. Based on users detail such as age it will provide the optional services in the order of priority. Based on the survey and the present market demand the mobile provider should provide different type of mobiles for different domain (age group). The highly prioritized services will be in-built and the lower priority services will be configured on-demand. Users are free to choose the optional services and can also omit the low priority services. PDMI process will be used for maintaining the priority of services for different age groups.

4 Analysis of project characteristics for choosing suitable methodology

We have chosen 14 project metrics; some of these metrics are taken from COCOMO model [15]. We have divided weight between all the 14 metrics. Weights are assigned in such manner so that the metrics having more support for traditional has given more weight and the metrics having more support for agile has given less weight. We have divided input parameters in three categories. Table below describes the numerical equivalent value for different categories.

Table 3:-Numerical equivalent value for different category.

Category	Low	Medium	High
Value	1	5	9

4.1 Description of different metrics

Complexity:-This metric is the measure of effort required to develop the software. More complex project should be developed using traditional methodology. So, the more weight is given to this metric.

Time to Market: - How fast product is required in the market? If it is required quickly then we should follow agile methodology.

Risk Involved: - Impact of failure of software. If risk is high then we should follow traditional methodology.

Flexibility: - Flexibility is the effort required to modify an operational program. High flexibility has more support for agile methodology.

Modularization of Task: - Is it possible to divide the product into different modules? If its value is high that is, if modularization is easy then we can follow agile methodology and deliver different modules in increments.

Volatility of requirements: - At later stage of development, how much requirements may change? The more value of this metric has more support for agile methodology.

Expandability: - The ease with which changes can be made to the software at later stages. The more value of this metric has more support for agile methodology.

Coupling: - The degree of interdependence between classes. Coupling increases complexity and hence more value of this metric has more support for traditional methodology.

Cohesion: - The larger the similarity between methods, the more cohesive is the class. Low cohesive class is more complex.

Tool Experience: - How much year of work experience the developer has on the tool to be used?

Platform volatility: - How frequently the platform (Operating system for which product is being developed) is changing.

Application Experience: - what is the work experience of the developer on the desired application (application may be java or c etc.).

Programmer's capability: - How much capable the programmer is; for development of the project?

Table 4:- Weight, Input values and their sum of product for decision making

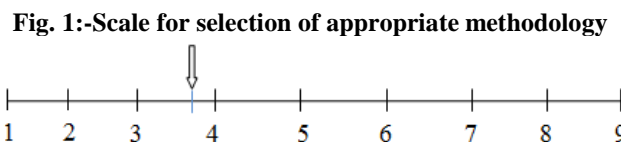
Name of Metrics (Project Characteristics)	Weight	Value of different metrics for Mobile App.	Product of weight and values
Complexity	0.15	Low (1)	0.15
Time to Market	0.08	High (9)	0.72
Risk Involved	0.15	Low (1)	0.15
Flexibility	0.05	High (9)	0.45
Modularization of Task	0.04	High (9)	0.36
Volatility of requirements	0.05	Medium (5)	0.25
Expandability	0.04	Medium (5)	0.20
Coupling	0.15	Low (1)	0.15
Cohesion	0.05	Medium (5)	0.25
Tool Experience	0.04	Medium (5)	0.20
Platform volatility	0.05	High (9)	0.45
Platform Experience	0.06	Low (1)	0.06
Application Experience	0.04	Medium (5)	0.20
Programmers Capability	0.05	Medium (5)	0.25
Total (Sum of Product)			3.84

Calculation:-
$$\text{Total (sum of product)} = \sum_{i=1}^{14} (\text{weight} * \text{Input value})$$

$$\text{Total} = (0.15*1) + (0.08*9) + (0.15*1) + (0.05*9) + (0.04*9) + (0.05*5) + (0.04*5) + (0.15*1) + (0.05*5) + (0.04*5) + (0.05*9) + (0.06*1) + (0.04*5) + (0.05*5)$$

Total=3.84

Output (Total sum of product) will range from 1 to 9. If this value comes between 1 and 4 then it indicates that we should follow agile methodology. If it comes between 6 and 9 then it indicates that we should follow traditional methodology. If value comes between 4 and 6 then we can consider any one or hybrid methodology. This is shown on the scale given below.



For our project the final outcome is 3.84 so we have chosen agile methodology.

5 Configurability in mobile communication domain

The IEEE glossary meaning of the configuration is “The arrangement of a computer system or component, defined by the number, nature, and interconnection of its constituent part” [9, 10]. It has been used to model business processes that are similar to one another in many ways yet differ in some other way. Configurability uses the notion of commonality (commonality that is uniform across a given set of objects) and variability (how members of a family may differ from one another) [11, 12]. We have extended the notion of configurability in method engineering and presented a meta model for configuring situation specific method [16]. It uses the notion of method essentiality and method variability. We apply this notion to configure a suitable method for mobile domain.

According to our proposal the basic services are the essential services and without them, mobile communication will lose its identity. All the other services can be considered as variable services and can be selected as per the requirement of the user. For our PDMI process, method essentiality constitutes a method that provides basic services. We conjecture that traditional methodology will be suitable for it. As evaluated in the previous section for variable mobile services agile methodology is suitable, hence using the method configurability a suitable method will be configured. In this paper we focus on selection of suitable agile methodology and the detail of the method configuration will be dealt in next paper.

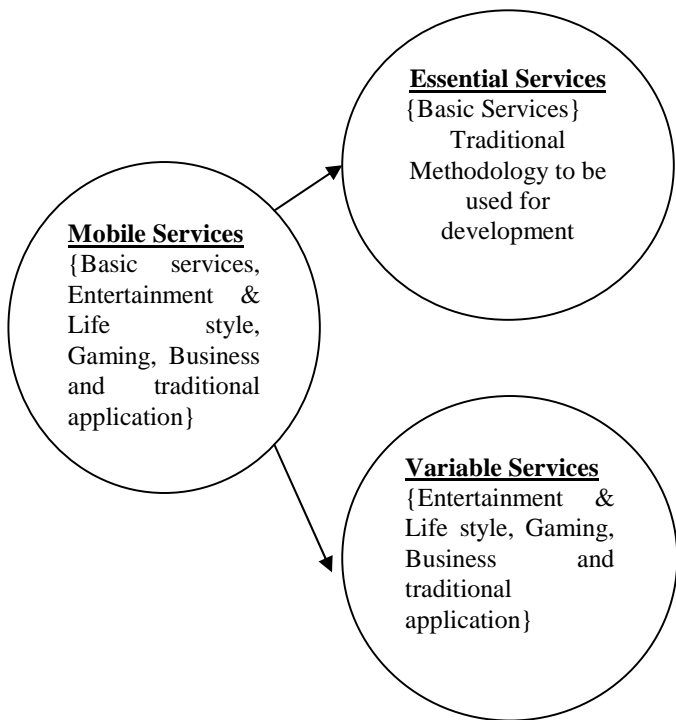


Fig. 2: Configuration for Mobile Services domain.

6 Implementation of PDMI Process through Agile Methodology

As described in section 2 agile methodology comprises of many agile methods like XP, SCRUM, DSDM, FDD, AUP etc. Each agile method [2, 3, 6] has its unique features and is good for one or other task. On the basis of this study we found that Scrum is best suitable methodology for our project, because we are going to develop our PDMI tool which is more lenient to project management.

For our project volatility of requirement is high. That means the requirement may change frequently which require reprioritizing the sprint backlog. Scrum supports reprioritization of backlog. On the other hand, as we have discussed that prioritization is different for different domain (age group) so we have come to a decision that mobile provider should target the entire domain and they should deliver various versions of the same product on the basis of demand in market. It requires a lot of management activity and they have to maintain the different backlog for different domain, so scrum is best suitable methodology for our project.

Scrum supports software development team to a great extent and also gives them [1, 4] full independence to perform. Two major stakeholders in a Scrum project are - Scrum master and Product owner. Scrum master helps the team members use the scrum framework and Product owner guides the team towards building the right product. Scrum is a process skeleton that contains sets of practices and predefined roles. The main actors in Scrum projects are:

Scrum Master- Scrum master manages the whole project, it gets project requirements from actors, decompose the requirements into modules in association with the project manager.

Product Owner-Product owner represents the stakeholders of the project.

Team- A group of about 7 people develops the modules and subsequently the project.

Our PDMI process is based on well known process of View Point Oriented Approach [8]. For each associated actor we create a different view, which describes the functionality of that particular stakeholder.

1. Scrum Master View
2. Sprint Planning Meeting View
3. Development Team View

Table 5: Functionalities of associated actors

View Points	Functionalities
Scrum Master View	<ol style="list-style-type: none"> 1. Identify project domain. 2. Enter project requirements and their priority in association with product owner. 3. Decomposition of project into modules.
Sprint Planning Meeting View	<ol style="list-style-type: none"> 1. Decompose highest priority module into the task and prioritization of tasks. 2. Select tasks to perform in current sprint. 3. Assign the task to the appropriate team on the basis of task complexity and team expertise. 4. Remove task from sprint on its completion.
Team View	<ol style="list-style-type: none"> 1. Pickup the task to perform. 2. Make entry for already burned-down task. 3. Submit the executables.

The idea behind using the agile methodology in our PDMI is, agile methodology believes in releasing functional product at the end of every iteration. The whole process starts with the requirements gathering phase, by the scrum master from the actors, the prioritization of requirements is done by actors in association with the scrum master. The highest priority tasks are forwarded to project development teams for an early release. As the software launches in the market, it starts generating revenues that helps the service providers to develop and implement other priority services.

We have developed a prototype Case tool that supports scrum development adapted to our proposal. Based on the priority of the services it distributes the task between product backlog and scrum backlog.

7 Conclusion

Priority based Domain specific Mobile service Implementation process helps an effective and fast mobile service implementation with high customer's satisfaction. Method configuration process can be used to integrate traditional method for basic services and agile methodology for variable services. This process is an advantageous solution for both service providers and customers. Service providers may start their services with the most basic ones and keep on increasing the add-ons as per the priorities of the customer whereas customers may have effective, quick and requirement based mobile service.

8 References

- [1] K. Schwaber, M. Beedle, *Agile Software Development with SCRUM*, Prentice Hall, 2002.
- [2] K. Petersen and C. Wohlin, *A Comparison of Issues and Advantages in Agile and Incremental Development between State of the Art and an Industrial Case*, Journal of Systems and Software, Vol. 82, No. 9, pp. 1479-1490, 2009.
- [3] Abrahamsson P., Warsta J., Sponen M., Ronkainen J. *New directions on agile methods: a comparative analysis*, Proceedings of the 25th International Conference on Software Engineering, 2003.
- [4] A. Cockburn, *Agile Software Development*, Addison-Wesley, Boston, 2002.
- [5] Nerur, S., Mahapatra, R. and Mangalaraj, G, *Challenges of Migrating to Agile Methodologies*, Communications of the ACM, Vol. 48, No. 5, May 2005, pp. 72-78.
- [6] Chau, T.; Maurer, F.; Melnik, G.(2003). *Knowledge sharing: agile methods vs. Tayloristic methods*, Twelfth IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, 2003. WET ICE 2003. Proceedings.
- [7] Mobile Electronic Transactions Ltd. MeT White Paper on Mobile Transactions (2003); www.mobiletransaction.org.
- [8] Sommerville, I., "Software Engineering" seventh edition 2003. ISBN-8129708671. Pearson Education.
- [9] Coplien J., Hoffman D., Weiss D., *Commonality and Variability in Software Engineering*, IEEE Software, 37-45, 1998
- [10] Weiss D. M., Lai C. T. R., *Software Product-line engineering: A Family based Software development Process*, Addison Wesley, 1999.
- [11] Karlsson, F., Ågerfalk, P.J. (2004) *Method Configuration: Adapting to Situational Characteristics While Creating Reusable Assets*. In Information and Software Technology 46, 2004, 619-633.
- [12] Wistrand, K., Karlsson, F. (2004) *Method Components – Rationale Revealed*. In Advanced Information Systems Engineering 16th International Conference, CAiSE 2004, Riga, Latvia, June 7-11, 2004, Proceedings, A. Persson, J. Stirna, Eds. Springer-Verlag, LNCS 3084, Berlin, 2004, 189-201.
- [13] Miller, G., *The characteristics of agile software processes*. The 39th international conference of object-oriented language and system (TOOLS39), Sanra Barbara, CA, 2001.
- [14] Beck, K. (1999a). *Embracing change with Extreme programming*. IEEE computer 32(10):70-77.
- [15] Chen. Z. *Finding the right data for software cost modeling*. Center for Software Eng., Univ. of Southern California, Los Angeles, CA, USA
- [16] Menzies, T.; Port, D.; Boehm, D. Volume: 22, Issue: 6
- [16] Gupta D. and Dwivedi R. *A Step Towards Method Configuration from Situational Method Engineering*. Software Engineering: An International Journal (SEIJ), INDIA, 2012 (51-5)

SESSION
NOVEL APPLICATIONS AND CASE STUDIES +
EDUCATION

Chair(s)

TBA

Using Visualization Software to Understand Complex Healthcare Interactions in Heterogeneous System Communities

H. Keith Edwards¹, Duane Bender², Paul Brown², and Justin Fyfe²

¹Department of Computer Science, University of Hawaii-Hilo, Hilo, Hawaii, United States

²Mohawk Applied Research Centre, Mohawk College, Hamilton, Ontario, Canada

Abstract – This research examines the role of visualization on the understanding of complex interactions that take place in large-scale communities of interdependent systems that comprise Health Information Exchanges. In particular, this paper uses a survey mechanism to examine the effectiveness of Mohawk College's Visualizer software on human understanding of the transmittal of electronic health record information in such communities of systems. The survey found that health informatics professionals were positively poised regarding the Visualizer's ability to facilitate understanding of vendor products in complex architectures, its ability to assist in interpreting audit messages, and its ability to facilitate the illustration of audit information. Respondents also reported a highly statistical difference in their understanding of the transmittal of electronic health record information when using the Visualizer software as opposed to an architectural diagram.

Keywords: Health Informatics, Visualization, Usability, Audit Messages, Electronic Health Records

1 Introduction

According to Knodel et al., "Visualization is a sound means to facilitate understanding of complex correlations and offers a broad variety of concepts." [5]. As such, Visualization can be used as a tool to understand complicated relationships between different systems that work together in large-scale heterogeneous communities [4].

In healthcare, software components from numerous vendors frequently work together in communities called Health Information Exchanges (HIE's) to provide services to a variety of clients in that community. Clients can be entities such as patients, labs, primary care physicians, specialists, and hospitals. In addition, these communities can also exchange data with systems from different communities.

Tools such as Mohawk's Visualizer software can provide important information about the complicated interactions between the heterogeneous components from the different vendors in these health information exchanges. The Visualizer works by showing the connections between the

components and by animating the exchange of information [2,9] using results derived from health care security audit messages [8].

Figure 1 shows a screen capture of the Visualizer with two Health Information Exchanges that take part in a Cardiologist referral scenario.

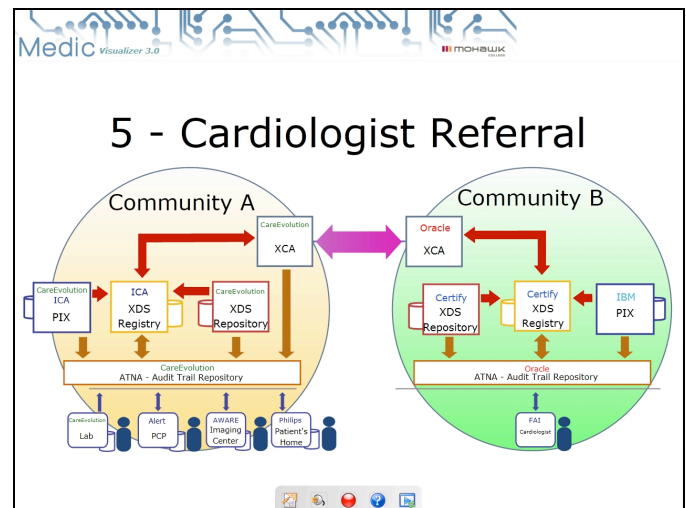


Figure 1: Mohawk's Visualizer

Mohawk College has employed the Visualizer to showcase the interactions in vendor communities at small-scale interoperability showcases such as the COACH conference in Canada as well as at significantly larger gatherings such as the HIMSS conference in the United States [15].

In this research, we want to understand whether the visualization software has an impact on human understanding of the information exchange process in large-scale health informatics communities.

2 Related Work

This section of the paper explores the related work in the area. In particular, we are interested in the application of Visualization to the field of health informatics when it comes

to the transmittal of electronic health record information inside of Health Information Exchanges.

As mentioned in the introduction, Visualization provides a way to understand complex interactions and relationships in real world data. Hence, it is not surprising that Visualization can be employed in applications and environments relating to health informatics.

For example, Hansen et al. developed a system to track how hospital workers moved throughout the hospital in order to track the spread of infections within a hospital environment. The interface for this system employed large data sets as well as an interactive touch screen in order to understand how infection control experts might use such a system to prevent outbreaks of infectious diseases [3].

It is not uncommon for visualization can be combined with data mining techniques. For example, Lavrac et al. use data mining techniques in conjunction with visualization to identify areas in Slovenia that were atypical for availability and accessibility of public health services [6]. Furthermore, visualizations in the public health domain can also be used for resource planning [11] or to integrate diverse epidemiological data with the occurrence of certain types of cancers [12].

Visualization can also be employed from the viewpoint of the physician as discussed in several papers [1, 7, 14]. For example, Mane and Borner look at innovative ways of viewing medical data for the purposes of diagnosis [7]. Roque et al. discusses six visualization systems that mostly target the physician as a primary user [14]. The systems in Roque also make use of a timeline that allows the physician to correlate the events in the patient's history using the electronic health record data.

Finally, patient centered perspectives are also possible such as the design in featured in Rajwan and Kim's article [13]. In this particular work, the authors develop a system design to support the sharing of information between patients and physicians that uses visualization to share complex information requiring a high volume of data.

Mohawk's Visualizer takes a middle ground between these approaches and the tools discussed in Howard et al. [4] in order to display the infrastructure of the software components and how they transmit electronic health record information to one another.

3 Experimental Environment & Design

In order to understand the impact of the Visualization software on human understanding of the interactions involved in Health Information Exchanges, we designed a survey instrument. The survey collected demographic information relating to the participant's organization, role within the organization, age range, and gender. In addition to the demographic questions, we posed six questions concerning

the effectiveness of the Visualizer for displaying information about the Health Information Exchanges. These latter six questions were all constructed using a 5-point Likert scale (strongly agree to strongly disagree) and were worded as follows

- Using only an architectural diagram, it is easy to understand how Electronic Health Record information is transmitted between individual systems in a complex architecture
- Using the Visualizer from MARC-HI enables me to understand how Electronic Health Record information is transmitted between individual systems in a complex architecture
- The Visualizer from MARC-HI facilitates understanding of various vendor products within a complex architecture
- Interpreting standard audit messages and illustrating this information is important
- The Visualizer from MARC-HI facilitates the interpretation of standard audit messages
- The Visualizer from MARC-HI facilitates the illustration of audit information

We distributed the paper-based survey to 35 participants at the 2012 HIMSS Conference in Las Vegas, Nevada who took part in the IHE Interoperability Showcase event. Since surveys were only distributed to visitors to the visualization booth, the response rate was 100%. However, this distribution method introduced a limitation into the applicability of the results, since the participants were all health informatics professionals who had an interest in visualization. Hence, it is important not to extrapolate the survey findings beyond this audience.

4 Discussion & Results

This section of the paper contains the results from the survey. Accordingly, the first subsection presents the demographics of the survey population whilst the second subsection presents the results for the Likert-scaled constructs.

4.1.1 Participant Demographics

The survey design divided the participants into five different age ranges, specifically 18-30, 31-40, 41-50, 51-64, and 65+. As can be observed in Figure 2, the majority of participants fell into the 31-40, 41-50, and 51-64 categories. There were only 2 participants in the 18-30 age range, 2 who did not answer, and no participants over the age of 65. This distribution was not surprising given that the sample came from a population of individuals employed in professional careers.

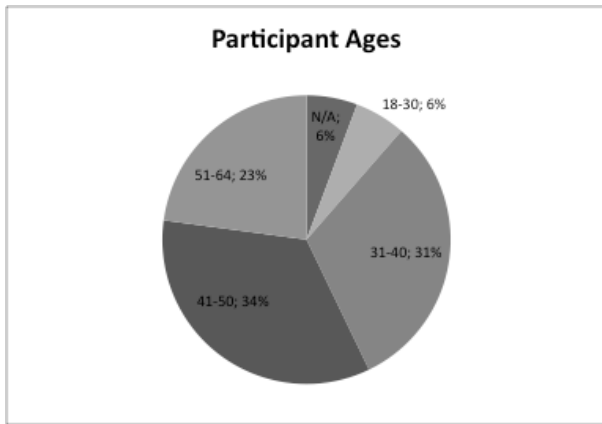


Figure 2: Participant Age Ranges

Likewise, the demographics for participant gender were skewed toward male participants as can be surmised from Figure 3. Given that the conference attracted a large number of computer and technical professionals, this finding is not surprising either.

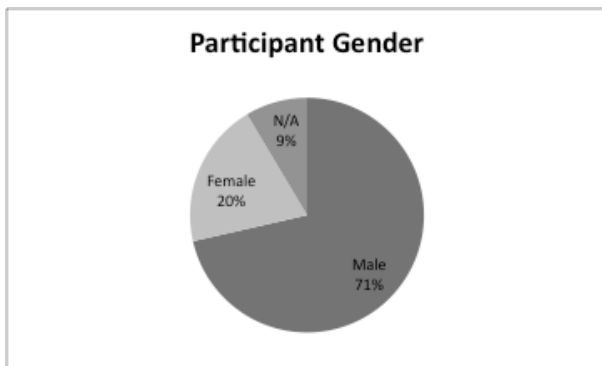


Figure 3: Participant Gender

The final piece of demographic information analyzed here is the participant role in their organization. Figure 4 shows that fully 2/3 of the survey participants were engaged in technical roles with their organization while another 31% worked in management and sales. Again, this finding reflects the general demographics of the HIMSS conference, which was designed to attract professionals in the field of health informatics.

In addition to the participant role, we collected free form reporting of the participant’s organizational affiliation. The vast majority of technical participants worked for technical companies such as Oracle and Optum whilst most management and sales participants worked for healthcare institutions or for universities.



Figure 4: Participant Roles

4.1.2 Survey Question Results

This section discusses the results from the Likert-scaled questions. For the first question, we asked participants to rate their degree of agreement with the statement: *“Using only an architectural diagram, it is easy to understand how Electronic Health Record information is transmitted between individual systems in a complex architecture”*.

Figure 5 shows the distribution of the responses for this question. The mode for the responses was 4 whilst the mean was 3.9¹. In general, participants were positively poised as to the effectiveness of this approach.

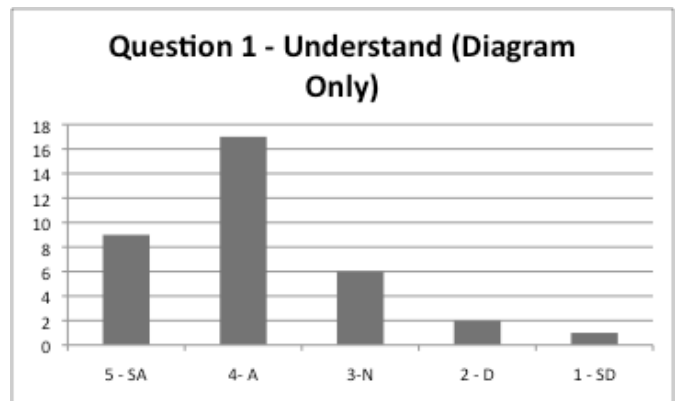


Figure 5: Understand Information Transmittal (Diagram Only)

Question 2 asked participants to rate their degree of agreement with the following statement: *“Using the Visualizer from MARC-HI enables me to understand how Electronic Health Record information is transmitted between individual systems in a complex architecture”*.

¹ Note: Likert-scale constructs are ordinal data, so we report the mean as a measure of central tendency for informational purposes only. All statistical tests we conduct are non-parametric and designed for ordinal data.

As can be seen in Figure 6, the participants were positively poised regarding the effectiveness of the Visualizer with the mode for responses being 5 and the mean being 4.7.

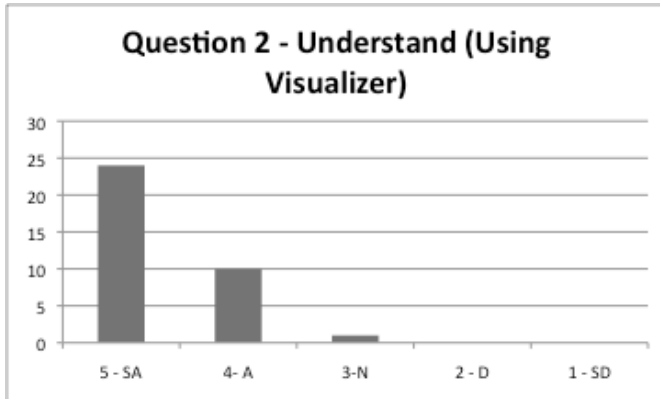


Figure 6: Understand Information Transmittal (Using Visualizer)

Question 3 asked participants to rate their degree of agreement with the statement: “*The Visualizer from MARC-HI facilitates understanding of various vendor products within a complex architecture*”. The results for this question are shown in Figure 7. Again, the mode for all responses was 5, and the vast majority of participants were positively poised about the Visualizer’s ability in this regard.

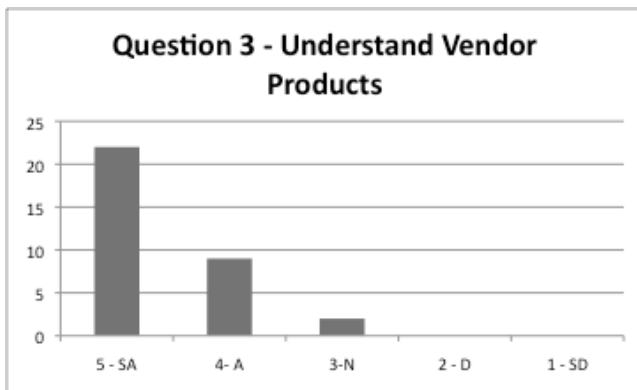


Figure 7: Understand Vendor Products

Question 4 asked participants whether interpreting standard audit messages and illustrating them was important. Again, the participants tended to agree with this statement. 19 reported that they strongly agreed with the statement, 11 agreed with the statement, and 3 remained neutral in this regard. The results for this question are shown in Figure 8.

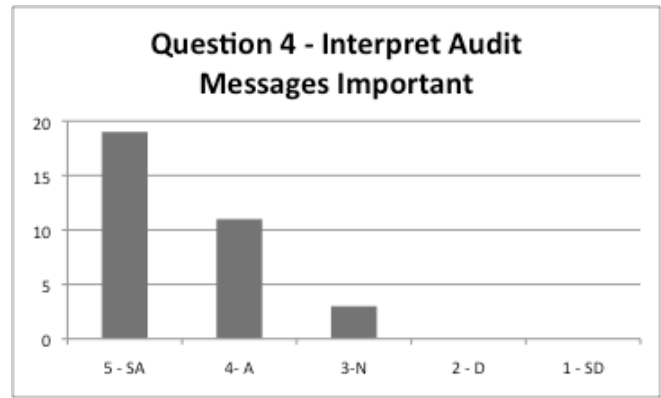


Figure 8: Interpreting Audit Messages is Important

Question 5 asked participants whether they agreed that the Visualizer facilitated the interpretation of standard audit messages. This question was more contentious than previous questions, since 2 participants disagreed and 3 left the construct blank. There were a higher number of neutral responses than the previous questions as can be observed from Figure 9.

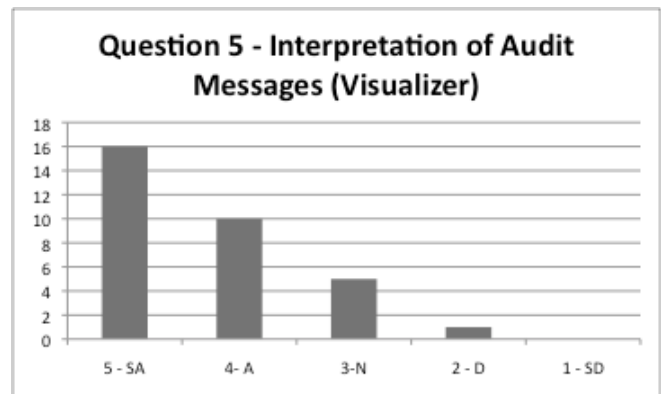


Figure 9: Visualizer Facilitates Interpreting Audit Messages

The final question in the survey asked subjects to rate their degree of agreement with the statement: “*The Visualizer from MARC-HI facilitates the illustration of audit information*”. Figure 10 displays the results from this question. Although there are no negative responses to this question and the respondents are generally positively poised about the subject, there were two blank responses on the survey sheets.

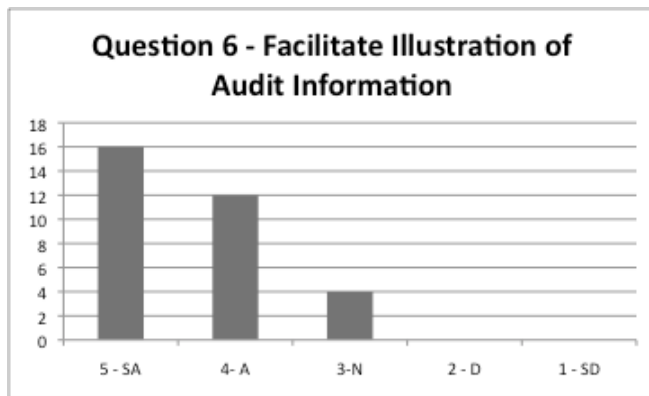


Figure 10: Visualizer Facilitates the Illustration of Audit Messages

In addition to measuring the central tendency for the six Likert-scaled questions, we also ran non-parametric tests such as the Mann-Whitney Wilcoxon and Mann-Whitney U-Test in order to discern whether there were significant differences between the distributions for the responses.

While there were no significant differences between any pair-wise matching of questions 2-6, there were significant differences between question 1 and the rest of the questions.

In particular, a Mann-Whitney Wilcoxon test between questions 1 and 2 generates a z-value of 3.8646 and a p-value of 0.0001. This indicates a highly statistically significant difference between questions 1 & 2. Furthermore, the more robust Mann-Whitney U-Test also yields a p-value of 0.0001. Hence, we can conclude that users have a significantly easier time understanding the exchange of electronic health record information in health information exchange communities with the aid of the Visualizer.

Likewise, the statistical differences between question 1 and the rest of the questions are indicative of the fact that the other questions (2-6) focus on the effectiveness of the Visualizer.

5 Conclusion

Visualization is greatly useful for understanding the complex interactions that take place in large-scale communities of interdependent systems, such as those found in Health Information Exchanges. Mohawk's Visualizer software uses audit repository messages from systems that process electronic health record data to display the flow of information between the systems.

This research developed a survey mechanism to evaluate the effectiveness of this approach on human understanding. The survey was administered to 35 health informatics professionals who attended the IHE Interoperability Showcase display at the HIMSS conference.

The survey found that participants were positively poised regarding the Visualizer's ability to facilitate understanding of vendor products in complex architectures, assist in interpreting audit messages, and facilitate the illustration of audit information. Most importantly, users reported a highly statistical difference in their understanding of the transmittal of electronic health record information when using the Visualizer software as opposed to an architectural diagram.

6 Future Work

A key limitation as to the applicability of this work on a larger scale is the fact that the survey population comes solely from business professionals who attended the interoperability showcase at HIMSS and stopped by the visualization booth. Hence, future work should focus on measuring the effectiveness of this tool and visualization for different populations.

In addition to surveying different user populations, several participants suggested the ability to drill down in the visualization in order to obtain additional information about the various components in each Health Information Exchange. These suggestions work well with the Visual Information Seeking Mantra, which states "Overview first, zoom and filter, then details on demand"[16].

Finally, extending the software to have statistical gathering capabilities can assist in user understanding of the entire process as evidenced by work such as Perer and Shneiderman [10].

7 References

- [1] Social Visualization of Health Messages. In Proceedings of the 42nd Hawaii International Conference on System Sciences (HICSS '09). IEEE Computer Society, Washington, DC, USA, 1-10. 2009.
- [2] Bender, Duane. Edwards, H. Keith. Fyfe, Justin. Brown, Paul. Providing Visualization Software for Large-Scale Health Informatics Communities. Quanta. Volume 1. Issue 1. Spring 2012.
- [3] Thomas E. Hansen, Juan Pablo Hourcade, Alberto Segre, Chris Hlady, Philip Polgreen, and Chris Wyman. 2010. Interactive visualization of hospital contact network data on multi-touch displays. In Proceedings of the 3rd Mexican Workshop on Human Computer Interaction (MexIHC '10), Eduardo H. Calvillo Gámez and Victor M. González y González (Eds.). Universidad Politécnica de San Luis Potosí, San Luis Potosí, S.L.P. México, México, 15-22.
- [4] Stephen L. Howard, James W. Hong, Michael J. Katchabaw, and Michael A. Bauer. 1995. Integrating visualization into event monitoring and analysis in distributed systems management. In Proceedings of the 1995 conference of the Centre for Advanced Studies on Collaborative research

(CASCON '95), Karen Bennet, Morven Gentleman, Howard Johnson, and Evelyn Kidd (Eds.). IBM Press.

[5] Jens Knodel, Dirk Muthig, Matthias Naab, and Dirk Zeckzer. 2006. Towards empirically validated software architecture visualization. In Proceedings of the 2006 ACM symposium on Software visualization (SoftVis '06). ACM, New York, NY, USA, 187-188.

[6] Nada Lavrac, Marko Bohanec, Aleksander Pur, Bojan Cestnik, Marko Debeljak, and Andrej Kobler. 2007. Data mining and visualization for decision support and modeling of public health-care resources. *J. of Biomedical Informatics* 40, 4 (August 2007), 438-447.

[7] Ketan K. Mane and Katy Borner. 2007. Computational Diagnostics: A Novel Approach to Viewing Medical Data. In Proceedings of the Fifth International Conference on Coordinated and Multiple Views in Exploratory Visualization (CMV '07). IEEE Computer Society, Washington, DC, USA, 27-34.

[8] G. Marshall. Security Audit and Access Accountability Message XML Data Definitions for Healthcare Applications. Request for Comments 3881. Network Working Group. September 2004.

[9] Mohawk Applied Research Centre. The Visualizer: Illustrating Interoperability via Visualization of Audit Messages. Technical Report. April 2011.

[10] Adam Perer and Ben Shneiderman. 2008. Integrating statistics and visualization: case studies of gaining clarity during exploratory data analysis. In Proceedings of the twenty-sixth annual SIGCHI conference on Human factors in computing systems (CHI '08). ACM, New York, NY, USA, 265-274.

[11] Aleksander Pur, Marko Bohanec, Nada Lavrac, Bojan Cestnik, Marko Debeljak, and Anton Gradisek. 2007. Monitoring Human Resources of a Public Health-Care System Through Intelligent Data Analysis and Visualization. In Proceedings of the 11th conference on Artificial Intelligence in Medicine (AIME '07), Riccardo Bellazzi, Ameen Abu-Hanna, and Jim Hunter (Eds.). Springer-Verlag, Berlin, Heidelberg, 175-179.

[12] Shweta Purushe, Georges Grinstein, Mary Beth Smrtic, and Helen Lyons. 2011. Interactive Animated Visualizations of Breast, Ovarian Cancer and Other Health Indicator Data Using Weave, an Interactive Web -- based Analysis and Visualization Environment. In Proceedings of the 2011 15th International Conference on Information Visualisation (IV '11). IEEE Computer Society, Washington, DC, USA, 247-252.

[13] Yair G. Rajwan and George R. Kim. Medical information visualization conceptual model for patient-

physician health communication. In Proceedings of the 1st ACM International Health Informatics Symposium (IHI '10), Tiffany Veinot (Ed.). ACM, New York, NY, USA, 512-516.

[14] Francisco S. Roque, Laura Slaughter, and Alexandr Tkatenko. 2010. A comparison of several key information visualization systems for secondary use of electronic health record content. In Proceedings of the NAACL HLT 2010 Second Louhi Workshop on Text and Data Mining of Health Documents (Louhi '10). Association for Computational Linguistics, Stroudsburg, PA, USA, 76-83.

[15] Rowe, Jeff (editor). Interoperability: Making the most of IT connections. HIMSS Daily News. February 20, 2012.

[16] Shneiderman, B. The Eyes Have It: A Task by Data Type Taxonomy for Information Visualization. In Proc. Visual Languages(1996), 336-343.

[17] Siegal S, Castellan Jr. N.J. (1988) Nonparametric statistics for the Behavioral Sciences 2nd. Ed. ISBN 0-07-057357-3 0. McGraw Hill Book Company New York (Mann Whitney Wilcoxon p128-137) (Robust Rank Order Test also called Mann Whitney U Test p. 137-144).

Training Effective Developers

M. Barjaktarovic

Department of Mathematics and Computer Science, Hawai'i Pacific University, Honolulu, Hawai'i, U.S.A.

Abstract - *Many employers today complain that they "cannot find enough talented developers." Everyone wants to hire only good programmers. How can higher education produce good programmers? This paper presents the results of our experience teaching coding to diverse audiences, from out-of-major freshmen to more senior majors. We have discovered through experience that, if students approach programming as an exercise in memorizing seemingly obscure and random programming language syntax, they struggle with coding. We found that by teaching them the formal basics of the syntax, students learn very quickly (literally in minutes) and retain the knowledge well. Their coding skills improve greatly. The language we use in this paper is Java, but any other language can be substituted.*

Keywords: design, testing, programming, curriculum, teaching

1 Introduction

Often, programming is taught through examples and memorization and without a formal approach. It would be equivalent to learning a foreign language by remembering commonly used phrases instead of understanding the rules of the grammar first. Although it is possible to learn this way, this approach is inefficient. Most of us understand that when it comes to learning foreign languages, but for some reason we do not treat programming languages in the same way.

Students, at the start of their studies, usually do not have the "big picture" and might not realize that there is actually a very logical ordering to Java syntax which makes learning Java much easier. We tell students that most high-level imperative programming languages tend to have the same rules except with slightly different syntax. Why that is so might not be immediately obvious and forces them to relate what they have learned in mathematics and basic computer organization.

All programming languages are similar to some degree because they languages are based on mathematics. Mathematics is always the same because computers are digital devices and work by processing binary data. Therefore, every computer operation boils down to processing binary numbers. The consequence of this very physical fact is that programming language syntax can do only what binary numbers can do. That limits what kinds of problems we can program.

In other words, every computer program is really a mathematical solution typed up in an appropriate syntax. This simple realization demystifies programming. A computer program is what we would do on paper in mathematics, except that we can use only characters that can be typed into a text editor, and we must follow certain rules dictating how to write out the solution. Those rules are called syntax.

In programming, we use the word "syntax." In math, we use the word "formula." Students are familiar with the concept of mathematical formulas and understand that a formula is an abstract placeholder into which we substitute values. (If they are not familiar with that concept, it has to be reviewed before we get into the discussion of syntax.) Programming language syntax is, in essence, one big formula.

One of the possible formalized ways to represent syntax is Backus-Naur form (BNF). BNF syntax is simply a way to write anything down formally. For example, [.] means that the item inside the [] is optional, + means that the item before the + repeats at least once, etc. BNF is used in many fields, for example computer networking [3]. It is important to write syntax into a formal way in order to assure clear communication. For example, compiler writers must have a common specification from which to write their compilers. Different compilers for the same programming language should parse in the same way. A compiler is really just a text parser, i.e. a compiler is just a program itself that reads source code and parses it based on syntax rules.

Java syntax rules, written in BNF, can be easily seen on the Web, e.g. on [7]. This is the complete syntax of the entire Java language. We do show those rules in class in an elaborate attempt to demystify programming by pointing out its obvious structure and thus clearly manageable nature. Students are always surprised to see the complete Java syntax in BNF form, having learned that "it is all there is to it." It demystifies the programming. It opens their minds and makes programming a lot more accessible.

We do not teach BNF syntax to students because it is more advanced than a beginning programmer needs to know. However, we do use a very simplified version of it when teaching programming. In our experience, it makes learning programming much faster and easier and we are surprised that this approach is not more commonly used in textbooks. We suspect that the missing link is the mathematical prerequisite [4], which could be overcome with a short review [1].

2 Syntax as a Formula: Java

In order to teach the concept of syntax as a “formula,” we chose some basic qualities for major components of Java code.

Each `< >` element marks a placeholder where we have to substitute a value into it. So, `< >` is akin to “x” in a formula such as “`x+2=x-2`”. It is necessary to plug in a value for x, just like we have to plug in a value for inside the `< >`. If a word is not enclosed in `< >`, it means that it is required by the rules of the syntax. In our lecture notes, we also like to highlight similar items in different colors, to further emphasize the patterns. Lines starting with `//` are comments.

A simplified formula for Java classes is shown below:

```
<accesstype> class <classname> {
    //fields
    <accesstype> <datatype> <fieldname>;

    //constructor
    <accesstype> <classname>
        (<datatype> <variablename>, ...) {
        .... statements ending with ;
    }

    //if there is no constructor specified, Java puts in:
    //public <classname>()

    //constructors can be overloaded, i.e.
    //there can be many constructors but with different
    arguments

    //methods are optional, but
    //we should have get and set for each private field
}
```

A simplified formula for Java methods is shown below:

```
<accesstype> <datatype> <name>
    (<datatype> <name>, ...) {
    .... statements ending with ;
}
//methods can be overloaded, i.e. methods can
//have the same name but different arguments
}
```

Each class has to be in a file named `<classname>.java`

Possible values for variables inside `<>`s are:
`accesstype` \in {public, private, protected}

`name` \in {...any possible combo of letters and numbers that starts with a letter and excludes special words...}

`datatype` \in {primitive, non_primitive}
`//primitive data types are passed / stored by value`

`primitive` \in {int, float, double, boolean, character}

`non_primitive` \in
 {array of any data types, //passed by reference
 object, //passed by reference
 list, stack, queue, //abstract data types

 }

Datatypes include the special case: void, as a possible datatype for “return” statement.

It is safer to briefly mention to the students that the above is a notation for sets. A set consists of elements in { }. For example, $S = \{1,2,3\}$ means that set S consists of elements 1, 2 and 3. $x \in S$ means that element x belong to set S.

Students also appreciate hearing this in ways they can relate to, such as GUI: each `< >` is like a Java drop-down menu, and the choices for the drop-down menu are all members of that set.

2.1 Basic Java class formula

This is yet another way that we try to use when showing Java syntax. It is a very simplified version of Java since students are in an early programming class.

A simplified Java class can be described as:

```
AccessIdentifier class Name {
    //access can be private or public
    Members of class (called fields)
    //can have private or public access
    Methods of class
    //can have private or public access
    Special case of method, called constructor
}
```

Constructor is a special case of method because it can only be used with the word **new** to create new objects. “Regular” methods can only be used on existing objects, e.g. to get values or to change values of the fields.

One of the critical aspects of teaching this is emphasizing that each class and each method is a “box” and thus it has some “formula” or algorithm associated with it. Therefore we need to be very clear as to what the boxes want as input, what output they produce, who may invoke them, and where and how they may be invoked.

For example, a constructor is a box that can only create brand new objects; it takes zero or some parameters and produces a brand new object; and can be invoked only with “new.” For example, a method is a box that can be called only by

existing objects; it takes some parameters and modifies the object or reports some info on the object.

The concept of objects and methods as boxes is useful for teaching testing and design. Agile developers are in favor of test driven design (TDD) [TDD]. It helps to visualize code as flow charts [2].

Another concept we emphasize, in order to make sure to clearly tie the prerequisite mathematical knowledge, is that a class is a concept, an abstraction, a way to group together related items. In other words: a class is a formula, i.e. a “cookie cutter” that is then used to make actual objects, i.e. “cookies”. Until we make objects based on the class “formula”, there is nothing tangible to work with. We create new cookies with the “new” directive and a constructor.

3 Examples

Concepts are best illustrated with examples that are familiar. We tried illustrating the concepts with the traditional example of making cookies and then proceeded to work with rectangles and restaurant orders. These examples were concocted by discussing them with students in class and getting their feedback. It was rather challenging to find the restaurant analogy, but students do understand it. Code was designed to illustrate the basic concepts as well as some basics of software engineering: documentation, variable names, sensible parceling into modules, and so on.

First we start with the high-level design in pseudocode.

3.1 Cookie Pseudocode

```
class Cookie {
    String topping; // topping ∈ {vanilla, chocolate, fruit}
    float calories; // calories ∈  $\mathbf{Z}^+$ 
    float weight; // weight ∈  $\mathbf{R}^+$ 
    float size; // size ∈  $\mathbf{R}^+$ 
    ... constructors...
    ... methods ...
}
```

```
main() {
//call one of the Cookie constructors to create
// a brand new chocolate cookie called cc
// Cookie cutter is the class
// Act of applying the cookie cutter to the dough is the
// calling the constructor
```

```
Cookie cc = new Cookie("chocolate");
```

```
//The only way to create a brand new object is to
// call the constructor
```

```
//call one of the Cookie constructors to create
```

```
//a brand new vanilla cookie called cv
```

```
Cookie cv = new Cookie("vanilla");
```

```
//change your mind; make the vanilla cookie be chocolate
```

```
cv.setTopping("chocolate");
```

```
//It is not possible to change a cookie unless it already exists
//i.e. methods work only on existing objects
}
```

3.2 Rectangle Pseudocode

```
class Rectangle {
    int length; // length ∈  $\mathbf{Z}^+$  to make it simple
    int width; // width ∈  $\mathbf{Z}^+$ 
    //MS Word rectangles also have color, fill, border, etc.
    ... constructors...
    ... methods ...
}
```

```
main() {
//call one of the Rectangle constructors to create a
//new Rectangle object of size 20x30.
```

```
Rectangle r1 = new Rectangle(20,30);
```

```
//make another rectangle
```

```
Rectangle r2 = new Rectangle(50,50);
```

```
//are the two rectangles equal?
```

```
if (r1.equals(r2)) {
    print "rectangles are equal";
}
```

3.3 Restaurant Order Pseudocode

```
class RestaurantOrder {
    String carb; // carb ∈ {rice, bread, noodles}
    String protein; // protein ∈ {chicken, beef, tofu}
    String veggie; // veggie ∈ {broccoli, beans, squash}
    String drink; // drink ∈ {soda, juice, water}
    int quantity; // quantity ∈  $\mathbf{Z}^+$ 
    float price; // price ∈  $\mathbf{R}^+$ 
    float tax; // tax ∈  $\mathbf{R}^+$ 
    ... constructors...
    ... methods ...
}
```

```
//Alice, Bob and Charlie walk into the restaurant.
```

```
//They see the menu on the wall; i.e. the menu is the class
```

```
//So they order based on the menu;
```

```
// i.e. they create new objects based on the class
```

```
//What is Order physically? A plate of food and the bill.

main() {
    //Alice orders for herself, i.e. she creates a new order:

    Order alice = new Order("chicken", "rice", "beans",
"water");

    //Bob orders for himself:

    Order bob = new Order("beef", "bread", "none",
"soda");

    //Charlie wants to order the same thing as Alice
    //so we are using copy constructor

    Order charlie = new Order(alice);

    //Bob changes his mind and decides to try broccoli
    //Since Bob's order already exists, the only thing
    //we can do with it is call the set method.
    //Calling the constructor would make a new order.

    bob.setVeggie("broccoli");

    //Alice asks Bob to repeat what he ordered for veggie

    String bob_veggie = bob.getVeggie();

    //Charlie decides to try the veggie that Bob is having

    charlie.setVeggie(bob.getVeggie());
OR
    Charlie.setVeggie(bob_veggie);

    //Charlie changes his mind,
    // and wants to eat the same thing as Bob

    charlie.copycat(Bob);

    //Alice clones her order for her friend Dianne

    //alice.clone();
    //this method returns an Order,
    //so you can do whatever you want with it
    //but cannot do: diane = alice.clone().
    //Why?
    //here are some possibilities with the clone:
```

```
float diane_tax = alice.clone().getTax();

    //Every time you call "clone" it will make a new object
    //This is not the best way to code. It is a lot better to
    //use the copy constructor.
    }
}
```

3.4 Implementation

So far we worked with the design. The next step is to actually implement our design. Therefore, students get to experience the design process and then the implementation, thus realizing that we think on different levels of abstraction. The classes are implemented as follows:

```
class Cookie {
    String topping;
    float calories;
    float weight;
    float size;

    //constructors
    public Cookie (String topping) {
        this.topping = topping;
    }

    //methods
    public void setTopping(String topping) {
        this.topping = topping;
    }
    public String getTopping() { return topping; }
}

class Rectangle {
    int length;
    int width;

    //constructors...
    public Rectangle( int len, int wid) {
        length = len;
        width =wid;
    }

    //methods:
    //getLen(), getWidth(), setLen(.), setWidth(.)

    public void equals(Rectangle r) {
        if (this.length == r.getLength() &&
            this.width == r.getWidth()) {
            return true;
        }
        else {
            return false;
        }
    }
}
```

```

class RestaurantOrder {
    String carb;
    String protein;
    String veggie;
    String drink;
    int quantity;
    float price;
    float tax;

    //constructors...

    //This is a "typical" constructor
    public Order(String carb, String protein, String veggie,
String drink, int quantity, float price, float tax) {
        this.carb=carb;
        this.protein=protein;
        this.veggie=veggie;
        this.drink=drink;
        this.quantity=quantity;
        this.price=price;
        this.tax=tax;
    }
}

```

4 Copy Constructor

The copy constructor can be rather elusive to students if they do not understand the basics concepts of classes. We give an in-class exercise to attempt to write a copy constructor and often we see the clone version of the copy constructor. Thus we decided to formally teach it and clarify the confusion. Therefore, we call the copy method "the copycat method" in order to distinguish it from the cloning version of copy.

We will continue using the example of restaurant class.

```

///////// Copy constructor //////////
//Input: An existing order
//Output: A brand new order based on existing order

public Order(Order existingOrder) {
    this.carb= existingOrder.getCarb();
    this.protein=existingOrder.getprotein();
    this.veggie=existingOrder.getveggie();
    this.drink=existingOrder.getdrink();
    this.quantity=existingOrder.getquantity();
    this.price=existingOrder.getprice();
    this.tax=existingOrder.gettax();
}

```

The copy constructor creates a new object. However, we might not want a new object. We might want to change an existing object.

If we want to copy an existing object into another existing object, we need to use the copycat method. The copycat method is the only way to safely copy.

Therefore, the copy constructor and copycat method have the same body, but they can have different calls. The copy constructor creates brand new objects, and the copycat method modifies existing object.

```

///////// Copycat method //////////
//Input: An existing order
//Output: none. Copycat method will change the object
// that called it; it should change all fields to
// match the object we are copying from

public void copycat(Order existingOrder) {
    this.carb= existingOrder.getCarb();
    this.protein=existingOrder.getprotein();
    this.veggie=existingOrder.getveggie();
    this.drink=existingOrder.getdrink();
    this.quantity=existingOrder.getquantity();
    this.price=existingOrder.getprice();
    this.tax=existingOrder.gettax();
}

```

Sometimes students write the copy method as a clone method, not as a copycat method. The clone method is not useful because the object returned is not easily accessible. Every time we call the clone method, we create a new object yet we can change its properties only once. If we do want a new identical object, we should use the copy constructor. .

```

///////// Clone method //////////
//Input: none
//Output: a brand new object cloned off this object

public Order clone() {
    Order o = new Order(this.carb, this.protein, .... );
    return o;
}
}

```

We tell students that the copy method is meant to be a copycat method. We use an example of fashion. For example, if Brad Pitt or Angelina Jolie are current fashion role models, then we try to copycat them and we make ourselves look like them. In other words, we invoke the copy method, i.e. copycat method. The entire fashion industry works on this principle. The copy method in the clone version would mean that Angelina Jolie would clone herself into her twin so there would be two identical people. However, we would not be able to easily access the twin. This is not useful. We should use the copy constructor instead.

4.1 Money Counterfeit Example

The basic difference between the copy constructor and the copy method can be easily be demonstrated by using a

money counterfeiting example. We can model printing new money, changing existing bills to look like higher-value bills, and also creating new bills by looking at existing ones. Below is the pseudocode that illustrates this example.

```
class Money {
    int bill ;

    getBill {
        return bill ;
    }
    setBill( int amount) {
        bill = amount ;
    }

    //constructor: prints a new bill
    Money (int a) {
        bill = a ;
    }

    //copy constructor: prints a new bill, same as another
    Money (Money m) {
        bill = m.bill ;
    }

    //copycat method: changes the bill we have
    void counterfeit(Money higherbill) {
        bill = higherbill.bill ;
    }

    //clone method: prints a new bill, the same like we
    have
    Money clone( ) {
        Money m = new Money(bill)
    }
}

main() {
    Money low = new Money(5) ; //print a $5 bill
    Money high = new Money(20) ; //print a $20 bill
    Money low2 = new Money(low) ; //print another $5
    bill

    low.counterfeit(high) ; //makes the $5 bill into $20 bill
    low.clone(high).setBill(100) ; //clone into $100 bill
    //However, $100 bill is not accessible anymore
}
```

5 Aggregate Classes

Aggregate classes seem quite new and complicated to students until we review the concept that Java syntax is a formula and that we can have fields of any data type. Therefore, it is possible to have fields that are objects. This

very quickly clarifies aggregate classes and makes learning them quite fast.

The discussion about copy constructor and copy method also makes learning aggregate classes easier since students understand the copying of objects. Writing the equals method is a natural extension.

6 Conclusions

This paper presents the results of teaching programming using a formalized approach, by treating syntax as a formula. Students learn rather quickly. For example, we have taught a 1-credit basic programming class to mechanical engineering freshmen at the University of Hawai'i at Manoa. Within one semester, students were able to write subprograms to multiply matrices. For example, aggregate classes in basic Java classes take a very short time to cover when taught using this approach.

7 References

- [1] Milica Barjaktarovic. "Teaching Mathematics and Programming Foundations Early in Curriculum Using Real-Life Multicultural Examples." The 2012 International Conference on Frontiers in Education: Computer Science and Computer Engineering (FECS'12), July 2012.
- Milica Barjaktarovic. "Teaching Design and Testing in Computer Science Curriculum." The 2012 International Conference on Frontiers in Education: Computer Science and Computer Engineering (FECS'12), July 2012.
- [2] Kent Beck. "Test Driven Development (TDD)." Addison-Wesley Professional; 1st edition, 2002.
- [3] D. Crocker, P. Overell. "Augmented BNF," RFC 5234. <http://tools.ietf.org/html/rfc4234>. January 2008
- [4] Nancy Eickelmann. "Interview to profile ACM Fellow David Lorge Parnas for SIGSOFT SEN"; SEN Vol. 24 Issue No. 3, May 1999. <http://www.sigsoft.org/SEN/parnas.html>
- [5] Paul Jarley, Dean of Lee Business School. "Where are all IT Students." Posted on February 20, 2012. <http://business.unlv.edu/dean/where-are-all-the-it-students/>
- [6] Mike Subelsky, Open Society Institute Baltimore. "Baltimore Should Become a Software Leader." <http://www.audaciousideas.org/2011/08/baltimore-should-become-a-software-education-leader/> Posted on August 2, 2011.
- [7] Source Forge. "BNF for Java Project." <http://bnf-for-java.sourceforge.net/>

Infrastructure Needed for Distance Learning in Developing Countries Through Multimedia Technology Using Mobile devices

Sagarmay Deb

Central Queensland University, Sydney, New South Wales, Australia

Abstract - *Although the developments of multimedia technology and internet networks have contributed to immense improvements in the standard of learning as well as distance learning in developed world, the developing world is still not in position to take advantage of these improvements because of limited spread of these technologies, lack of proper management and infrastructure problems. Unless we succeed in solving these problems to enable people of developing countries to take advantages of these technologies for distance learning the vast majority of the world population will be lagging behind. In this paper we explore how to develop appropriate infrastructures for the use of mobile technology to provide distance learning in an efficient way using advanced multimedia tools. We recommend the use of mobile and multimedia technology to reach this vast population of under-developed countries to impart quality learning in an effective way.*

Keywords: Distance learning, infrastructure, mobile technology, multimedia technology, developing countries

1 Introduction

The concepts of distance learning are prevalent in developing countries for last few decades and it is very much in vogue in developed countries [1], [4]. In developing countries it started like many other countries did with correspondence courses where printed learning materials used to be despatched to the students at regular intervals and students were expected to read the materials and answer questions. The basic philosophy was teachers would be physically away from the students and have to conduct the teaching process from distance [2].

With the development of computer industry and internet networks during the last three decades things have changed and global communication has reached an unprecedented height [1]. With these developments immense scopes have come to the surface to impart learning in a much more efficient and interactive way. Multimedia technology and internet networks have changed the whole philosophy of learning and distance

learning and provided us with the opportunity for close interaction between teachers and learners with improved standard of learning materials compared to what was existing only with the printed media. It has gone to such an extent to create a virtual class room where teachers and students are scattered all over the world. Although some of these facilities are expensive still the developed world is in a position to take advantage of these facilities to impart much better distance-learning to students residing in the developed countries. But for developing countries the story is different as computerization and network connections are still very limited compared to the developed world. In this paper we focus our attention on defining the problems infrastructures that is needed for using these technologies for much more improved and extensive distance-learning and suggest how we could possibly reach these vast majority of people from the developing countries with the improved quality of distance-learning provided by multimedia and internet networks through viable and affordable infrastructures that could be created in those setup.

Section one gives an introduction of the area. Section two presents the advancements in infrastructures developing countries are making to make use of mobile technologies. Section three presents the issue of usage of mobile technology with advanced multimedia tools in distance learning in developing countries with appropriate infrastructures. We put our concluding remarks in section four.

2 Analyses of Works Done

The open-universities which started functioning by late sixties and early seventies of last century, reaching off-campus students delivering instruction through radio, television, recorded audio-tapes and correspondence tutoring. Several universities particularly in developing countries still use educational radio as the main instructional delivery tool [1].

Print, audiocassettes, and pre-recorded instructional television (lectures) are the lowest-cost technologies for small numbers of students (fewer than 250), while radio requires 1,000 students or more to achieve comparable per-student costs. Computer conferencing is a low-cost approach to providing interactivity between teachers

and students, but live interactive broadcasts and video conferencing are still very high-cost technologies, regardless of the number of students enrolled [17].

Although distance education has been around for more than two hundred years and has been shown to be effective in a variety of settings, the introduction of technology and its application across global boundaries introduces new trends, issues, and challenges. How, for example, does one judge the quality of a degree earned over the Internet? When should technology be used? And which technology is best? Should countries use programs offered by foreign institutions rather than developing their own? These and many other questions are confronting education policymakers and practitioners around the world. Careful analysis, evaluation, and research will be needed [17].

Globalization, accreditation, and competition. Employers and universities are now drawing both staff members and students from all corners of the globe. Consequently, they face new challenges in evaluating course work done at, and degrees earned from, unknown institutions in other countries. While accreditation has typically been controlled by individual countries, the globalization of distance education has created a whole new challenge in accreditation and certification of learning. For example, the Global Alliance for Transnational Education (GATE) has been formed to carry out the formidable task of creating a global certification and review process for education delivered across borders [17].

Globalization raises other issues for countries. For example, instructional programs broadcast from abroad have heightened fears about the contamination of cultures and values. Competition between local and foreign education providers is another issue. While competition is usually good for the consumer, in that it often raises quality and reduces prices, local institutions typically resist foreign competition and, in some countries, are trying to block outsiders from operating in local markets [17].

Quality and effectiveness. Some developing countries are reluctant to adopt programs originating elsewhere, despite their reputed quality, choosing instead to develop their own; unfortunately, many lack the expertise needed to produce high-quality materials and support structures. Considerable time and expense are required to produce quality programs, and countries with limited resources may put programs together that are inadequate [17].

Technology. Making sound investment decisions about technology is a major challenge facing educational policymakers and planners. New technologies offer options to both expand educational opportunity and improve quality, but inappropriate decisions regarding whether to use technology or what type of technology to use can be costly and can impede the success of a

distance education program. Unfortunately, the information needed to make such decisions is limited. Care should be taken to avoid allowing the novelty of technology to drive decisions regarding the most appropriate delivery mode for distance education programs, overshadowing the more important decisions regarding curriculum and instructional quality. If a country's conventional education or teacher training program is not effective, using a new technology to deliver that education or training will not make it any more effective [17].

Affordability. Distance education programs need sound financial planning and management to ensure sustainability. In many cases, developing countries find that funds are not available to continue a distance program after donor funds are terminated, so it is important that initial investment be accompanied by adequate funding for recurrent expenditures. A related problem arises when the per-student cost of adding distance education or other education technology is large relative to a country's average per-student financial allocation for that educational level. For instance, if a distance mathematics program using computers consumes financing equal to 50 percent of the country's per-student budgetary allocation, its financial future is likely to be bleak, despite high putative benefits. When such a situation is encountered, the country would be best advised to opt for pilot programs that test less expensive alternatives than to do away with the distance education program entirely [17].

It is useful to consider distance learning not in terms of a technical problem but as essentially similar to traditional learning. Distance learning can be seen as comprising the following functions that broadly correspond with the TrainX methodology. They are: Course design, Development of materials, Evaluation, Course delivery, Evaluation.

Providing support for students can be difficult with distance learning. This can be exacerbated in developing countries where trainees may lack other kinds of support and be widely dispersed. The lack of face-to-face support and human interaction can lead to student isolation and high dropout rates.

Providing support to distance learning trainees – either through online forums, telephone or mail, requires proper coordination of human resources and/or the provision of additional face-to-face training opportunities.

Ongoing support is provided by using local networks and trainers and through a 'training of trainers' programme. Distance learning activities use regularly scheduled chat sessions to provide opportunities for feedback and questions as well as regular e-mail contact or telephone support where appropriate.

In order to promote a wider distribution of expertise and knowledge and so that the beneficiary countries feel some ownership over the training, regional pedagogical committees are established to oversee the training and distance learning activities [20].

In Africa there is a need for utilizing resources to effectively develop and use ICT solutions. There are a lot of previous experiences in different African countries like Sierra Leone and Nigeria, they use many methodologies of the distance learning from corresponding courses, CD-ROM, Internet, TV and World Wide Web. Sudan As in many African countries, poor network infrastructure is a main challenge, in addition to lack of awareness and commitment of teachers and institutions. The Sudan Open University is the first initiative in this area. It is prime initial phase of development, depends on printed material, lectures in CD Room, cassette, TV, Radio and new one video conference (between the main center and one state for one time).SOU(Sudan Open University) has electronic library on line and now enrolled 93,000 students since 2003 in (Educational, Administration, Computers collages) .Their future plans include develop a video conference capabilities to enable access to their online library for their students [18].

Even though India has shown considerable progress, the full development and large scale adoption of e-Infrastructure-enabled distance learning still face several challenges, including:

- Lack of course content,
- High cost of production of high quality e-learning material,
- Lack of satisfactory quality applications in certain areas (like genomic sciences).

Tackling these challenges requires:

- More training for greater expertise in development and delivery of e-learning solutions,
- Additional investment in research and application and product development,
- More reliable communication infrastructures with higher bandwidth,
- Further development and use of standards
- Higher availability of adequate IPv6 applications [19].

3 How To Develop Proper Infrastructures For Distance Learning

With the extended application of information technologies (IT), the conventional education system has crossed physical boundaries to reach the un-reached through a virtual education system. In the distant mode of education, students get the opportunity for education through self-learning methods with the use of technology-mediated techniques. Efforts are being made to promote distance education in the remotest regions of developing countries through institutional collaborations and adaptive use of collaborative learning systems [2].

Initially, computers with multimedia facilities can be delivered to regional resource centers and media rooms can be established in those centers to be used as

multimedia labs. Running those labs would necessitate involvement of two or three IT personnel in each centre. To implement and ascertain the necessity, importance, effectiveness, demand and efficiency, an initial questionnaire can be developed. Distributing periodical surveys among the learners would reflect the effectiveness of the project for necessary fine-tuning. After complete installation and operation of a few pilot tests in specific regions, the whole country can be brought under a common network through these regional centers [2].

In developed economies, newer versions of technology are often used to upgrade older versions, but in developing economies where still older versions of technology are often prevalent (if they exist at all), the opportunities for leapfrogging over the successive generations of technology to the most recent version are that much greater [3].

In the conventional view, (i.e. as seen by technology developers and donors), developing countries passively adopt technology as standard products which have been developed in industrialized countries and which can be usefully employed immediately. However, successful use of IT requires much more than mere installation and application of systematized knowledge. It also requires the application of implied knowledge regarding the organization and management of the technology and its application to the contextual environment in which it is to be used. This implied IT knowledge often represents experience with the deployment of previous technology accumulated over time, such experiences contributing towards the shaping of new technology [3].

In addition to purely technological issues, the development of appropriate human resources skills are required, i.e. extensive training of the people who are going to use (and train others how to use) the resources. Training is seen as particularly important as this is not technology just a few people to benefit from, but for many. As Pekka Tarjanne, Secretary General of the ITU, made clear at Africa Telecom '98, "communication *is* a basic human right" (original emphasis). Nelson Mandela, at Telecom 95 in Geneva, urged regional co-operation in Africa, emphasizing the importance of a massive investment in education and skills transfer, thereby ensuring that developing countries also have the opportunity to participate in the information revolution and the "global communications marketplace"[3].

Canada's International Development Research Centre (IDRC) runs a number of developing country projects that involve technology leapfrogging. The Pan Asian Network (PAN) was set up to fund ICT infrastructure and research projects in developing countries across Asia. Individuals, development institutions, and other organizations should all be able to use the infrastructure so as to share information [3].

PAN works with Bangladesh's world famous grassroots Grameen Bank. One service here is a "telecottage", where network services can be obtained. The technology and the material will be tailored to meet the needs of Grameen's typically poorly educated clients. One of PAN's objectives is gender equity. Women, who constitute some 95% of Grameen's borrowers, will be prominent among PAN users in Bangladesh [3].

PAN is also responsible for linking Laos to the Internet. The Science, Technology and Environment Organization (STENO) of the Laos Government invited some Laotian IT professionals living and working overseas to return home and share their experiences with their colleagues in the country. STENO collaborated with PAN in designing an 18-month long project to build the necessary infrastructure for a dial-up e-mail service. Among the pioneer users were "researchers working on agriculture and aquaculture projects; journalists managing national news agencies and newspapers; lawyers consulting on international legal issues; travel agents planning business trips; computer resellers tracking down suppliers and obtaining pricing information; and about 20 others in both the public and private sectors" [5].

Presentation of course materials through multimedia in remote locations where in villages there could be school structures where those presentations could be made is feasible. Of course learning materials must be self-explanatory and not boring. Using multimedia facilities like videos, audios, graphics and interesting textual descriptions, it is possible to reach the remote locations of the world where computer technology has not reached yet. As the areas not covered by computer and internet technology is still profoundly vast in the world this approach seems to be very constructive and should be pursued.

Wherever possible distance learning through multimedia should be imparted through internet as internet and networks are the vehicles of multimedia. But since bandwidth connection is still very limited in vast areas of Asia, Africa and Latin America it would still take long time to reach major part of the population of the above-mentioned regions with multimedia and web.

Mobile technology offers a very hopeful way to reach the vast population of the developing countries as it does not require bandwidth connections. We have to develop distance learning using multimedia through mobile technology. This seems to be the most viable way to reach billions living in the rural areas of the developing countries. Hence considerable research efforts must be dedicated to this line. Instructions could be sent through emails to mobiles of the distance learners. Also relevant website addresses could be transmitted to their emails and they could then visit those sites of distance learning through the internet of their mobiles.

In his book, Mayer (2001) declares that while learning from the text-only books results in the poorest retention and transfer performance, learning from books that include both text and illustrations and from computer-based environments that include on-screen text, illustrations, animations and narrations results in better performance [10].

Similar to e-Learning, mobile technologies can also be interfaced with many other media like audio, video, the Internet, and so forth. Mobile learning is more interactive, involves more contact, communication and collaboration with people [14].

The increasing and ubiquitous use of mobile phones provides a viable avenue for initiating contact and implementing interventions proactively. For instance, Short Message Service (SMS) is highly cost-effective and very reliable method of communication. It is less expensive to send an SMS than to mail a reminder through regular postal mail, or even follow-up via a telephone call. Further, no costly machines are required (which is clearly the case in terms of owning a personal computer). Besides SMS, distance learners can use mobile phones/ MP3 players to listen to their course lectures, and for storage and data transfer. New technologies especially mobile technologies are now challenging the traditional concept of Distance Education [12]. Today the more and more rapid development of the ICT contributes to the increasing abilities of the mobile devices (cell phones, smart phones, PDAs, laptops) and wireless communications, which are the main parts of the mobile learning. On the other hand for the implementation of mobile learning it is necessary to use a corresponding system for the management of such type of education [13].

The use of mobile technologies can help today's educators to embrace a truly learner-centred approach to learning. In various parts of the world mobile learning developments are taking place at three levels:

The use of mobile devices in educational administration

Development of a series of 5-6 screen mobile learning academic supports for students

Development of a number of mobile learning course modules [11].

Research into the current state of play in Europe indicates:

1. There is a wide range of roles for mobile technologies supporting the learner in many ways ranging from relatively simple use of SMS texting to the more advanced use of smartphones for content delivery, project work, searching for information and assessment. Some proponents of mobile learning believe that it will only „come of age“ when whole courses can be studied, assessed and learners accredited through mobile devices.

2. Although books are now being downloaded onto mobile devices, the authors believe that to support the learning process a great deal of thought has to be given to the structure of the learning and assessment material. However, it is true that for some, mainly at higher education level, mobile phones offer the opportunity to access institutional learning management systems. This provides greater flexibility to the learner without any new pedagogical input.

3. Costs are coming down rapidly; new first generation simple mobile phones will not be available on the market from 2010. All mobile phone users in Europe will be using 3 or 4G phones within the next two years. A welcome associated step is a move towards some form of standardization by the mobile phone companies as exemplified by the shift to common charging devices over the next two years.

4. The value which is put on possession of a mobile phone, especially by young people is surprising and the data on ownership suggests that this will be a ubiquitous tool for all very shortly and that it will be well cared for: there is evidence that ownership of devices brings responsible use and care.

5. Large scale educational usage in schools currently depends on government investment but in higher and further education it is safe to assume that all learners will have their own devices. Institutions will need to advise potential students on the range of devices most suitable for the curriculum, as they do currently with regard to computers. The convergence between small lap tops and handheld devices will continue until they are regarded as different varieties of the same species of technology.

6. There is a great potential for educational providers to work with large phone companies, both to reduce costs and to co-develop appropriate software [6].

Bangladesh Open University (BOU) is the only national institution in Bangladesh which is catering distance education in the country. It has extensive network throughout the country to provide readily accessible contact points for its learners. After passing of 15 years since its inception, BOU has lagged behind in using technologies. In consideration of its limit to conventional method in teaching, a project was undertaken to test the effectiveness and viability of interactive television (TV) and mobile's Short Message Service (SMS) classroom and explore the use of available and appropriate technologies to provide ICT enabled distance tuition. In this project, the mobile technology's SMS along with perceived live telecast was used to create ideal classroom situation for distance learning through the Question Based Participation (QBP) technique. The existing videos of BOU TV programs were made interactive using this technologies and technique. The existing BOU TV program and interactive version of the same were showed to same learners of BOU to evaluate its effectiveness. It is found from the study that this interactive virtual classroom significantly perform well in teaching than BOU video programs (non-interactive) which is used at present [7].

Another paper presents and discusses NKI (Norwegian Knowledge Institute) Distance Education basic philosophies of distance teaching and learning and their consequences for development of a learning environment supporting mobile distance learners.

For NKI it has been a major challenge to design solutions for users of mobile technology who wish to study also when on the move. Thus, when students are mobile and wishing to study, the equipment and technologies they use will be in addition to the equipment used at home or at work. The solutions must be designed in ways to allow both users and non-users of mobile technology to participate in the same course. This means that we have looked for solutions that are optimal for distributing content and communication in courses, independent on whether the students and tutors apply mobile technology or standard PC and Internet connection for teaching or learning. The learning environment must efficiently cater for both situations and both types of students. The solutions were developed for PDAs. During the time of the development and research the technologies have developed rapidly. Mobile phones are including PDA functionalities and vice versa. In principle the aim of developments is to design solutions that can be used on any kind of mobile devices.

The paper builds on experiences from four European Union (EU) supported projects on mobile learning: From e-learning to m-learning (2000-2003), Mobile learning – the next generation of learning (2003-2005), Incorporating mobile learning into mainstream education (2005-2007) and the ongoing project, The role of mobile learning in European education (2006-2008).

Most NKI courses are not designed to function as online interactive e-learning programs, although some parts of the courses may imply such interaction with multi-media materials, tests and assignments. The courses normally involve intensive study, mainly of text based materials, solving problems, writing essays, submitting assignments and communicating with fellow students by e-mail or in the web based conferences. This means that most of the time the students will be offline when studying. From experience we also know that the students often download content for reading offline and often also print out content for reading on paper. All aspects and functions of mobile learning in the NKI large scale distance learning system is clearly an additional service to the students [8].

Mobile Assisted Language Learning (MALL) describes an approach to language learning that is assisted or enhanced through the use of a handheld mobile device. MALL is a subset of both Mobile Learning (m-learning) and Computer Assisted Language Learning (CALL). MALL has evolved to support students language learning with the increased use of mobile technologies such as mobile phones

(cellphones), MP3 and MP4 players, PDAs and devices such as the iPhone or iPad. With MALL, students are able to access language learning materials and to communicate with their teachers and peers at any time anywhere [9].

3.1 Current Limitations of Mobile Technology

Every technology has some limitations and weaknesses, and mobile devices are no exception. They have shown some usability problems. Kukulska-Hulme summarized these problems as follows: 1) physical attributes of mobile devices, such as small screen size, heavy weight, inadequate memory, and short battery life; (2) content and software application limitations, including a lack of built-in functions, the difficulty of adding applications, challenges in learning how to work with a mobile device, and differences between applications and circumstances of use; (3) network speed and reliability; and (4) physical environment issues such as problems with using the device outdoors, excessive screen brightness, concerns about personal security, possible radiation exposure from devices using radio frequencies, the need for rain covers in rainy or humid conditions, and so on. It is important to consider these issues when using mobile devices and designing the learning environment [15]. We expect mobile producers would take care of these problems in the near future.

3.2 How To Overcome Any Limitations in the Spread of Distance Learning

As we discussed in Section 2 some countries could be concerned if they buy study materials from overseas it would have an adverse effect on their cultures. But these countries can always sit with the people preparing those courses overseas and guide them how to design the materials which would not have any adverse effect in their countries cultures or societal setup. The pedagogical considerations also could be settled this way. Even in a particular country with multi-cultural setup there could be multiple versions of the study materials available. It would be upon the governments of those developing to formulate a policy on distance education according to the requirements and existing setups and affordability. There should be schools set up on rural areas with at least couple of computers in each campus where students can watch and learn geography, mathematics and so on based on multimedia approach of text, audio, video and graphics. At the post school level there could be some small software centres where some IT trained persons could be deployed to help and guide people going for distance learning through mobile technology. This would solve the problem of learning being isolated and would lower down the number of dropouts.

Looking at how rapidly new mobile products are improving, with advanced functions and numerous

applications and accessories available these days, the technical limitations of mobile devices may be a temporary concern. Also, the use of mobile technologies in education is moving from small-scale and short-term trials or pilots into sustained and blended development projects [16].

Most developing countries do not have an extensive infrastructure to support M-Learning, and this makes it more complicated to implement it in these countries. However, this developing world still maintain similar needs for M-Learning as developed countries do. Ken Masters (2004) proposes that the lack of infrastructure should be no reason for developing countries to delay implementing M-Learning. It is essential, that if the need exists, institutions within these developing countries should establish and commence mobile learning efforts as soon as possible [21].

Users in developing countries have the same need for M-Learning to be mobile, accessible and affordable, as those in developed countries do. The very significance of M-Learning is its ability to make learning mobile, away from the classroom or workplace. These Wireless and mobile technologies enable learning opportunities to learners who do not have direct access to learning in these places. Many learners in developing countries have trouble accessing the internet, or experience difficulty in affording technology that enables learning in an E-Learning environment. Mobile devices are a cheaper alternative compared to traditional E-Learning equipment such as PC's and Laptops [22].

However, to fully utilize this potential it is imperative to explore the factors that determine mobile telecommunications development in the developing world[23]. Delivering mobile services on open hardware and open software not just practically make sense but can also lower the cost and thus increase the possibility of offering sustainable services in the future.[24] While the benefits of open-source software are proven, it is important to conduct a broader study to investigate the potential role of relatively new copyleft approach for custom hardware, as supporting mobile learners in their own socio-cultural contexts of developing countries is a significant challenge[25].

Mobile learning cannot be imparted to a learner until he or she attains certain qualifications and age. Also socio-economic situations of the society concerned would dictate the growth of mobile technology as we know in many societies young population have to enter the work force at a very early age. This shows socio-economic development is very important for providing distance learning in developing societies.

Also there could be problems of deploying qualified teachers in rural setup of developing countries. But this could be overcome if properly trained teachers are deployed to make the curriculum and to monitor and support distance learning from the resource centre setup

in urban areas. They could even make occasional visit to rural areas for providing face to face learning support.

4 Conclusion

In this paper we studied the problems of infrastructures in imparting distance learning through multimedia in developing countries. We suggested guidelines which the developing countries can adapt to spread education through distance learning in their countries using mobile technology a viable and affordable media through which distance learning could be imparted to billions of people in an efficient way. We presented some examples of achievements in this field in this paper where we can use telephone, photography, audio, video, internet, eBook, animations and so on in mobile and deliver effective distance education in developing countries. More research needs to be carried out to improve the infrastructures required for spreading distance learning among billions in developing countries through mobile technology and gearing up multimedia technology to be easily transported to those locations.

5 References

- [1] K Passerint and M J Granger. "Developmental Model for Distance Learning Using the Internet, Computer & Education"; 34, (1), (2000)
- [2] H Rahman. "Interactive Multimedia Technologies for Distance Education in Developing Countries - Introduction, Background, Main focus, Future trends, Conclusion"; <http://encyclopedia.jrank.org/articles/pages/6637/Interactive-Multimedia-Technologies-for-Distance-Education-in-Developing-Countries.html>, (2000)
- [3] R Davison, D Vogel, R Harris, N Jones. "Technology Leapfrogging in Developing Countries – An Inevitable Luxury?"; Journal of Information Systems in Developing Countries (2000)
- [4] S Ruth, J Giri. "The Distance Learning Playing Field: Do We Need Different Hash Marks?"; http://technologysource.org/article/distance_learning_playing_field/ (2001)
- [5] S Nhoybouakong, M L H Ng, R Lafond, <http://www.panasia.org.sg/hnews/la/la01i001.htm>, (1999)
- [6] "Using Mobile Technology for Learner Support in Open schooling"; www.col.org/sitecollectiondocuments/
- [7] M S Alam, Y M Islam. "Virtual Interactive Classroom (VIC) using Mobile Technology at the Bangladesh Open University (BOU)"; wikieducator.org/images/4/45/PID_563.pdf
- [8] A Dye, T Rekkedal. "Enhancing the flexibility of distance education through mobile learning" The European Consortium for the learning Organisation. ECLC – 15th International conference, Budapest, May 15-16 (2008)
- [9] Mobile Assisted Language Learning, en.wikipedia.org/wiki/Mobile_Assisted
- [10] R. E. Mayer. "Multimedia learning"; Cambridge. Cambridge University Press (2001)
- [11] "Implications of Mobile Learning in Distance Education for Operational Activities"; http://wikieducator.org/images/c/c6/PID_624.pdf
- [12] M. Yousuf. "Effectiveness of Mobile Learning in Distance Education. Turkish Online Journal of Distance Education-TOJDE"; October 2007 ISSN 1302-6488, 8, (4), Article 9, (Oct 2007) <http://www.google.co.in/search?hl=en&q=%22Effectiveness+of+Mobile+Learning+in+Distance+Education%22&meta=> Retrieved on 31.3.2008, (2006)
- [13] E Georgieva. "A Comparison Analysis of Mobile Learning Systems. Paper presented at International Conference on Computer Systems and Technologies- CompSysTech; " (2006), <http://ecet.ecs.ru.acad.bg/cst06/Docs/cp/sIV/IV.17.pdf> Retrieved on 31.3.2008
- [14] G N Vavoula. "D4.4: A study of mobile learning practices. MOBIlearn project deliverable. The MOBIlearn project website"; http://www.mobilelearn.org/download/results/public_deliverables/MOBIlearn_D4.4_Final.pdf (2005)
- [15] A Kukulska-Hulme. "Mobile usability in educational context: What have we learnt?"; International Review of Research in Open and Distance Learning, 8(2), 1-16 (2007)
- [16] J Traxler. "Defining, discussing, and evaluating mobile learning: The moving finger writes and having write..."; International Review of Research in Open and Distance Learning, 8(2), 1-12 (2007)
- [17] M Potashnik, J Capper. "Distance Education: Growth and Diversity"; <https://tojde.anadolu.edu.tr/fdmarchnws.htm> (1998)
- [18] H M Elnour. "Distance Learning in Sudan – The Potential and Challenges"; [//www.appropriatetech.net/files/Distance_Learning_in_Sudan.pdf](http://www.appropriatetech.net/files/Distance_Learning_in_Sudan.pdf) (Year Unknown)
- [19] "Position statement on e-Infrastructures for Distance Learning: Opportunities and Challenges for

the Indian Society”;
www.beliefproject.org/.../indian.../Indian%20Position%20Statement

[20] “Strategy for implementing a Distance Learning (DL) process in UNCTAD for strengthening training capacities in international trade in developing countries”; United Nations Conference on Trade and Development Geneva (2004)

[21] “Offline mobile learning From Wikipedia - the free encyclopedia”;
http://en.wikipedia.org/wiki/Offline_mobile_learning

[22]. Template:M-learning: Developing Countries

[23] G Ping, R Adnan. “Analysing the Mobile Telecommunications Market in a Developing Country: A Socio-Technical Perspective on Pakistan, Centre for Development Informatics”; Institute for development policy and management, SED. (2009)

[24] S Shrestha, J Moore, J Abdelnour-Nocera. “Offline Mobile Learning for ICT4D”, IADIS International Conference Mobile Learning 2010, (March 2010)

[25] S Shrestha, J. Moore, J. Abdelnour-Nocera. "Low-cost hardware for ICT4D: what's right and what's left?"; IEEE Multidisciplinary Engineering Education Magazine, 6, 1 (2011).

A Multi-attribute Decision Making Approach for Resource Allocation in Software Projects

A. Ejnoui¹, C. E. Otero¹, and L. D. Otero²

¹Information Technology, University of South Florida Lakeland, Lakeland, Florida, USA

²Engineering Systems, Florida Institute of Technology, Melbourne, Florida, USA

Abstract—For most software companies, the production of reliable software within the planned time schedule is of paramount importance. Many times, inadequate resource allocation can lead to high costs and low quality in software products. Hence, it is critical to put in place well-planned processes for personnel assignment that take into consideration the skill sets of the personnel with the objective of reducing costs and training time as well as increasing product quality. To this end, this paper presents a novel methodology that considers multiple project-specific skills for assigning human resources to software projects. This methodology takes into account the existing capabilities of personnel to determine the best fit based on the required skills for the task. Because personnel selection is essentially an imprecise task, this methodology uses fuzzy sets to represent the problem of personnel assignment as a fuzzy multi-attribute decision problem. This problem is solved by ranking personnel candidates based on the expected value operator of fuzzy numbers. A sample case study is used to show the methodology's capabilities.

Keywords: : Expected Value Operator, Fuzzy Sets, Software Engineering, Resource Allocation, Multi-Attribute Decision Problem.

1 Introduction

Task assignment decisions in software development environments are critical because they influence the performance of workers and quality of products [1]. As documented in [2], the U.S. Department of Defense (DOD) spent nearly 8 billion dollars in 2004 to rework software. This large financial figure serves as evidence that quality-related issues continue to be a major struggle for software companies. Furthermore, “evidence reveals that the failure of software development projects is often a result of inadequate human resource project planning” [3]. In [4], Linberg stated that only about 16.2% of software projects are on time and within budget. A major contributor to this problem is the inefficient allocation of resources that may result in schedule overruns, decreased customer satisfaction, decreased employee morale, reduced product quality, and negative market reputation. The inevitable consequence is a decrease in potential profit for companies.

Despite all the research and advances in the field, software development is still very challenging due to its unpredictable nature. The fast pace at which new technologies and techniques are being developed today to improve the design and development of products increases the demand for specialized individual skills in the workforce. Most of the time, candidates with the required skills to work on specific tasks are not available, and decision makers are forced to assign resources to tasks based on subjective measures [1]. Therefore, further studies of the processes and techniques for personnel management are necessary to provide better solutions in terms of quality, cost, and schedule. Often, assigning resources is not certain and can be very fuzzy.

This paper proposes a fuzzy multi-attribute decision making approach for allocating human resources to task assignment in software engineering. The approach uses a fuzzy function ranking to provide a unified metric representative of the suitability between the complete set of skills available from candidates and skills required for tasks [5]. As such, decision makers can quantitatively assign resources to tasks even when the most desirable skills are not available from the existing workforce. The approach is extensible to consider a wide variety of project specific capabilities, such as years of experience, level of perceived expertise on a particular language, operating system, domain knowledge, etc. Moreover, managers can use this methodology as a tool to increase the efficiency of resource allocation.

This paper is divided into seven sections. Section 2 describes the literature related to resource allocation in software projects. Section 3 briefly describes the proposed methodology for personnel assignment. Section 4 presents a detailed coverage of ranking of fuzzy variables using the expected value operator. Section 5 introduces concepts related to how credibility spaces are used to compute the expected values of fuzzy numbers. Section 6 presents results of the approach on a small case study. Finally, section 7 concludes this paper.

2 Related Work

In recent literature regarding the assignment of software developers to tasks, Acuña et al. stated that software managers typically make assignments based on “their experience, heuristic knowledge, subjective perception, and

instinct” [1]. In [6], Duggan et al. developed a multi-objective optimization model for software task allocation based on genetic algorithms. The competencies of developers were modeled using a categorical variable with five levels. Each competency level was associated with an expected productivity per day, and an expected number of defects per unit of productivity. Other studies have developed procedures for allocating personnel to software tasks based on the assessment of behavioral competencies [1, 7].

In [3], the authors proposed selecting resources using the CRD method and the Taguchi’s parameter design approach. The CRD was used because it focused on resource scheduling rather than activity scheduling to represent human-resource workflow and tasks’ precedence. The Taguchi’s parameter design was used to obtain a scheme that would optimize the selection of engineers for tasks under dynamic and stochastic conditions. The Taguchi’s parameter design approach is based on the concept of target value [8]; that is, any deviations from the target value will result in additional costs. Deviations are attributed to controllable and uncontrollable factors. The aim is to achieve the optimal levels of the controllable factors while minimizing the variation caused by the nuisance (uncontrollable) factors [9].

In [3], the skill levels of resources were estimated as an average number of software lines of code (SLOC) per day. The authors commented on the importance of including in their model stochastic factors affecting the selection of resources. Specifically, emphasis was placed on the stochastic behavior of tasks complexities, since they are very difficult to measure or even estimate, causing most of the variability of calculated project completion times.

Other methodologies used to evaluate staffing alternatives include the Analytical Hierarchy Process (AHP) and linear programming (LP). In [10], Ho-Leung used AHP to tackle the problem of human resource substitution, considering several organizational, client, and application attributes. Even though the proposed model did not consider the relationship between known and required skills, the author commented about the importance of developing faster methods for human resource substitution. In [11], the authors proposed an LP assignment model to match resources to tasks when optimum skill sets are not available. Their model takes into account existing capabilities of candidates, required levels of expertise, and priorities of required skills for the task.

While the approaches described above use parameters that are quantified as crisp values, most often qualitative criteria cannot be precisely defined particularly when performing capability assessment in skill-based environments. Because these criteria tend to be imprecise, they have been modeled using fuzzy sets. One of the earliest attempt in this direction used fuzzy expected values to represent the capability of an employee [12]. Others addressed this problem by modeling competency levels using fuzzy variables [13-15]. In addition,

fuzzy relationships have been integrated in expert systems to match employees with specific tasks [16, 17].

From the reviewed literature, it is evident that there is much room for improved personnel assignment methodologies in software projects. The literature shows that the most common measure of the ability of a software developer is an estimated value of SLOC per day. Furthermore, this estimated SLOC value is usually a function of developers’ experience level. To the best of our knowledge, a readily available methodology that considers complete set of capabilities of candidates, levels of skills required, and priorities of required skills for tasks is nonexistent in the software development literature.

3 Proposed Approach

To properly make resource allocation decisions in software engineering projects, decision-makers must follow a decision-making process that takes into consideration the fundamental efficiency metrics present in specific projects. The creation of such process is achieved as follows. First, experienced project leads must identify the particular skill-set required for a particular project. Then, from the pool of available candidates, each candidate is assessed using a small set of skill levels based on previous performance, educational background, or combination of both. In the proposed approach, we suggest five skill levels: None, Novice, Proficient, Advanced, and Expert. After each candidate is evaluated in all required skills, these skill levels are used as input to the proposed fuzzy multi-attribute decision problem to compute the expected value of a weighted fuzzy function.

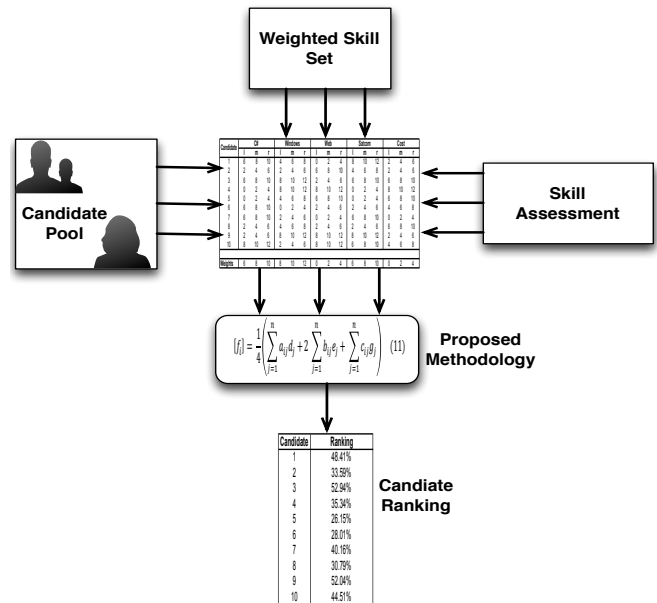


Figure 1. Proposed approach.

Finally, the candidates with the highest expected value of this function get selected for the software project.

4 Preliminaries

This section introduces basic concepts in credibility theory that are critical in developing the approach proposed in this paper. Most of these concepts have been published in [5, 18, 19].

4.1 Fuzzy Variables and Credibility

Let Θ be a nonempty set and $P(\Theta)$ its power set. Each element in $P(\Theta)$ is called an *event*. The credibility of an event A , denoted by $\text{Cr}\{A\}$, is a number that represents the credibility that A will occur.

Definition 1 Let Θ be a nonempty set, $P(\Theta)$ the power set of Θ , and Cr a credibility measure. Then the triplet $(\Theta, P(\Theta), \text{Cr})$ is called a *credibility space*.

Definition 2 A *fuzzy variable* is defined as a (measurable) function from a credibility space $(\Theta, P(\Theta), \text{Cr})$ to the set of real numbers.

Theorem 1 (Credibility Inversion) Let ξ be a fuzzy variable with membership function μ . Then for any set B of real numbers, we have

$$\text{Cr}\{\xi \in B\} = \frac{1}{2} \left(\sup_{x \in B} \mu(x) + 1 - \sup_{x \in B^c} \mu(x) \right) \quad (1)$$

where B^c is the complement set of B . The proof of this theorem can be found in [18, 19].

4.2 Expected Value

Although there are many ways to define an expected value operator, we choose to focus on the most general definition of this operator [5, 18-21]. This definition is applicable to both continuous and discrete fuzzy variables.

Definition 3 Let ξ be a fuzzy variable and θ a real number. Then the expected value of ξ is defined as

$$E[\xi] = \int_0^{+\infty} \text{Cr}\{\xi \geq \theta\} d\theta - \int_{-\infty}^0 \text{Cr}\{\xi \leq \theta\} d\theta \quad (2)$$

provided that at least one of the two integrals is finite.

Definition 4 Let $A = (l, m, r)$ be a triangular fuzzy number characterized by its grade membership function as:

$$\mu_A(x) = \begin{cases} 1 - \frac{(m-x)}{l}, & (m-l) \leq x \leq m \\ 1 - \frac{(x-m)}{r}, & m \leq x \leq m+r \\ 0, & \text{otherwise} \end{cases} \quad (3).$$

The values l , m , and r are respectively the left, middle and right spreads of A .

Theorem 2 Based on Definition 3, the expected value of a triangular fuzzy number $A = (l, m, r)$ can be calculated as follows:

$$E[A] = \int_0^{+\infty} \text{Cr}\{\xi \geq \theta\} d\theta - \int_{-\infty}^0 \text{Cr}\{\xi \leq \theta\} d\theta = m + \frac{r-l}{4} \quad (4).$$

The proof of this theorem can be found in [22].

Theorem 3 Let ξ and η be independent fuzzy variables with finite expected values. Then for any numbers a and b , we have

$$E[a\xi + b\eta] = aE[\xi] + bE[\eta] \quad (5).$$

This property is called the linearity of expected value operator of fuzzy variables. The proof of this theorem can be found in [18].

4.3 Ranking of Fuzzy Variables

Contrary to the set of real numbers, fuzzy variables do not have a natural order in a fuzzy world. As such, several approaches were devised to rank fuzzy variables [23]. One approach is based on the expected value operator of a fuzzy variable.

Definition 4 (Expected Value Criterion) Let ξ and η be fuzzy variables with finite expected values. We say $\xi > \eta$ if and only if $E[\xi] > E[\eta]$ where E is the expected value operator of a fuzzy variable.

This definition can be readily applied to triangular fuzzy numbers.

5 Expected Value Method

This section formulates the problem of resource allocation in software projects as a fuzzy multi-attribute decision problem. Then, it proposes a solution to this problem based on the expected value of fuzzy numbers in the problem formulation.

5.1 Problem Formulation

In making decision to allocate resources in software projects, we assume a set of candidates $C = \{C_1, C_2, \dots, C_m\}$ and a set of skills required to complete a project $S = \{S_1, S_2, \dots, S_n\}$. We assume that the evaluation of each candidate with regard to each skill has been completed by a project manager resulting in a fuzzy decision matrix $A = [\xi_{ij}]_{m \times n}$ where each fuzzy number ξ_{ij} represents the skill level of candidate C_i in skill S_j . In fact, each skill can be viewed as a fuzzy variable characterized by its membership functions based on a set of linguistic concepts defining the level of expertise in each skill. We also assume a set of weights $W = \{w_1, w_2, \dots, w_m\}$ that represents the weights of the skills in S . This formulation is known as the *fuzzy multi-attribute decision problem (FMADM)* where the skill set S represents the attributes in the decision matrix A .

5.2 Matrix Normalization

In most FMADM problems expressed in matrix form, normalization is necessary in order to transform the matrix and weight vector numbers to comparable values. In our case, normalization is based on the expected value operator [5]. For each fuzzy number ξ_{ij} in A , transform this number as follows:

If ξ_{ij} is a benefit:

$$\eta_{ij} = \frac{\xi_{ij}}{\sqrt{\sum_{i=1}^m (E[\xi_{ij}])^2}} \quad (6)$$

for $i = 1, 2, \dots, n$ and $j = 1, 2, \dots, m$.

If ξ_{ij} is a cost:

$$\eta_{ij} = \frac{p_j - \xi_{ij}}{\sqrt{\sum_{i=1}^m (E[\xi_{ij}])^2}} \quad (7)$$

for $i = 1, 2, \dots, n, j = 1, 2, \dots, m$, and $p_j = \max_{1 \leq i \leq m} \sup\{x_{ij} \mid \mu_{ij}(x_{ij}) > 0\}$. Note that both expressions use in their denominators the expected values of the fuzzy variables of each column or attribute. Also, note that the μ_i are the membership functions of the fuzzy variables representing the attributes. The obtained normalized matrix is $B = [\eta_{ij}]_{m \times n}$. In addition to the decision matrix, the set W can be normalized as follows:

$$\omega_j = \frac{w_j}{\sum_{j=1}^n E[w_j]} \quad (8)$$

for $j = 1, 2, \dots, n$. The final normalized weight vector is $\omega = [\omega_1, \omega_2, \dots, \omega_n]$.

5.3 Expected Value Approach

Given a normalized matrix of fuzzy numbers and a normalized weight vector, a simple additive weighting approach can be used to compute the following m fuzzy variables as follows [5]:

$$f_i = \sum_{j=1}^n \omega_j \eta_{ij} \quad (9)$$

for $i = 1, 2, \dots, m$. Each fuzzy variable f_i can be viewed as the real-value function associated with each candidate. A utility value function $E[f_i], i = 1, 2, \dots, m$, based on the expected value operator can be devised to rank the m fuzzy variables. Assuming that the fuzzy variables have triangular membership functions, this utility $E[f_i]$ can be computed using equation (4). In this case, the real-valued function f can be computed as follows if we assume $\eta_{ij} = (a_{ij}, b_{ij}, c_{ij})$ and $\omega_j = (d_j, e_j, g_j)$:

$$f_i = \left(\sum_{j=1}^n a_{ij}d_j, \sum_{j=1}^n b_{ij}e_j, \sum_{j=1}^n c_{ij}g_j \right) \quad (10)$$

for $i = 1, 2, \dots, m$. In this case, the utility function of f_i can be computed as:

$$E[f_i] = \frac{1}{4} \left(\sum_{j=1}^n a_{ij}d_j + 2 \sum_{j=1}^n b_{ij}e_j + \sum_{j=1}^n c_{ij}g_j \right) \quad (11)$$

for $i = 1, 2, \dots, m$.

6 Case Study

This section presents results of a resource allocation case study using the proposed approach. The case study assumes a scenario where 10 candidates are available. The identified required skill set involves knowledge of the C# language, Windows platform, web programming, and knowledge of the satellite communications domain. In addition, cost is identified as a decision making unit. Skills are modeled using the fuzzy sets shown in Figure 2 while weights are modeled using the fuzzy sets shown in Figure 3. The first set shows five skill levels: None, Novice, Proficient, Advanced, and Expert. On the other hand, the second set shows five levels of importance: Not Important, Somewhat Important, Moderately Important, Important, and Very Important. Costs are modeled in five levels of severity similar to the levels used in the weights, but their membership functions are identical to the ones shown in Figure 2. Using synthetic data, the skill assessment matrix is presented in tabular form as shown in Table 1. The weights are shown in the bottom of Table 1.

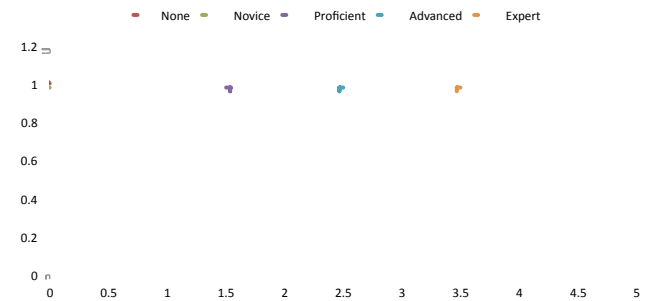


Figure 1. Fuzzy set of skill levels.

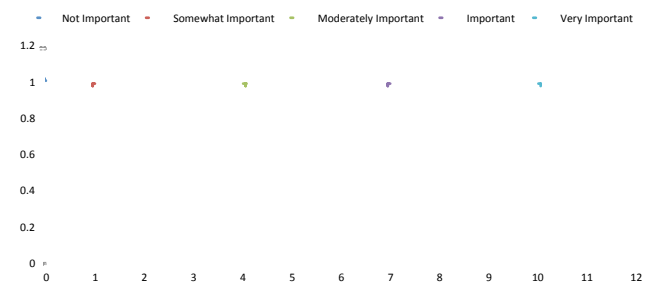


Figure 2. Fuzzy sets of weights.

Table 1. Skill assessment matrix for the case study.

Candidate	C#			Windows			Web			Satcom			Cost		
	l	m	r	l	m	r	l	m	r	l	m	r	l	m	r
1	0	0	1	2.5	3.5	4.5	0	0	1	1.5	2.5	3.5	0	1	2
2	2.5	3.5	4.5	0	0	1	0	0	1	2.5	3.5	4.5	0	1	2
3	2.5	3.5	4.5	0	1	2	2.5	3.5	4.5	3.5	5	5	0	0	1
4	0	0	1	3.5	5	5	1.5	2.5	3.5	1.5	2.5	3.5	1.5	2.5	3.5
5	2.5	3.5	4.5	0	0	1	2.5	3.5	4.5	1.5	2.5	3.5	1.5	2.5	3.5
6	0	0	1	3.5	5	5	2.5	3.5	4.5	0	1	2	2.5	3.5	4.5
7	1.5	2.5	3.5	2.5	3.5	4.5	0	1	2	3.5	5	5	0	1	2
8	3.5	5	5	3.5	5	5	0	1	2	1.5	2.5	3.5	2.5	3.5	4.5
9	0	0	1	0	0	1	0	0	1	1.5	2.5	3.5	1.5	2.5	3.5
10	2.5	3.5	4.5	1.5	2.5	3.5	0	1	2	0	1	2	2.5	3.5	4.5
Weights	7	9	11	7	9	11	0	0	2	4	6	8	1	3	5

Table 2. Normalized matrix.

Candidate	C#			Windows			Web			Satcom			Cost		
	l	m	r	l	m	r	l	m	r	l	m	r	l	m	r
1	0	0	0.099	0.337	0.471	0.606	0	0	0.196	0.218	0.363	0.508	0.618	0.441	0.265
2	0.247	0.346	0.445	0	0	0.135	0	0	0.196	0.363	0.508	0.653	0.618	0.441	0.265
3	0.247	0.346	0.445	0	0.135	0.269	0.489	0.685	0.88	0.508	0.725	0.725	0.618	0.618	0.441
4	0	0	0.099	0.471	0.673	0.673	0.293	0.489	0.685	0.218	0.363	0.508	0.353	0.176	0
5	0.247	0.346	0.445	0	0	0.135	0.489	0.685	0.88	0.218	0.363	0.508	0.353	0.176	0
6	0	0	0.099	0.471	0.673	0.673	0.489	0.685	0.88	0	0.145	0.29	0.176	0	0.176
7	0.148	0.247	0.346	0.337	0.471	0.606	0	0.196	0.391	0.508	0.725	0.725	0.618	0.441	0.265
8	0.346	0.495	0.495	0.471	0.673	0.673	0	0.196	0.391	0.218	0.363	0.508	0.176	0	0.176
9	0	0	0.099	0	0	0.135	0	0	0.196	0.218	0.363	0.508	0.353	0.176	0
10	0.247	0.346	0.445	0.202	0.337	0.471	0	0.196	0.391	0	0.145	0.29	0.176	0	0.176
Weights	0.292	0.375	0.458	0.292	0.375	0.458	0	0	0.083	0.167	0.25	0.333	0.042	0.125	0.208

The columns l, m, and r represent the left, middle and right values of each fuzzy number representing a skill level in the table. Using equations (6)-(8), the assessment matrix is normalized as shown in Table 2. Normalization is based on the expected values of the triangular numbers in the matrix as equation (4) shows. This table is used to compute the ranking of the candidates based on equation (11). As seen from this particular scenario, candidates 8, 7, and 3 are the top three candidates that fit the required skills. These candidates display rankings of 52.83%, 50.74%, and 44.87% in terms of skills for the project at hand. In fact, candidate 8 displays expert skills in C# and Windows, as well as proficient skills in satellite communication.

Table 3. Candidate ranking.

Candidate	Ranking
1	34.22%
2	33.43%
3	44.87%
4	37.47%
5	27.92%
6	32.07%
7	50.74%
8	52.83%
9	14.23%
10	32.73%

7 Conclusion

The research presented in this paper develops a systematic approach for planning resource allocation in software projects based on multiple criteria. Specifically, it presents a methodology based on the expected value operator for ranking fuzzy numbers. This concept is used to rank fuzzy numbers representing skill levels for each candidate. Through a small case study, the approach is proven successful in providing a way for analyzing resource assignment for application-specific projects.

There are several important contributions from this research. First, the approach is simple and readily available for implementation using a simple spreadsheet. This can promote usage in practical scenarios, where highly complex methodologies for resource allocations are impractical due to schedule and budget constraints. Another important contribution from the approach presented in this research is the ability to consider numerous decision-making factors in the decision-making process. For example, beside the skills presented in the case study, the approach can be easily extended to incorporate project-specific factors, such as expected availability, level of clearances, etc. Finally, the results provided by this approach can be used by program managers to tailor scheduling goals to make them more realistic. Depending on the overall skill rankings of all candidates in a team, program managers can determine if either more resources need to be allocated for the task, or

scheduling requirements need to be relaxed to complete the project.

8 References

- [1] S. T. Acuña, N. Juristo and A. Moreno, "Emphasizing human capabilities in software development," *IEEE Software*, vol. 23, no. 2, pp. 94–101, March 2006.
- [2] U.S. General Accounting Office (GAO), "Defense Acquisitions: Stronger Management Practices Are Needed to Improve DOD's Software Intensive Weapon Acquisitions," Document number GAO-04-393, 2004.
- [3] H. Tsai, H. Moskowitz, H. Lee, "Human resource selection for software development projects using Taguchi's parameter design," *European Journal of Operational Research*, vol. 151, no. 1, pp. 167-180, November 2003.
- [4] K. R. Linberg, "Software developer perceptions about software project failure: a case study," *The Journal of Systems and Software*, vol. 49, no. 2-3, pp. 177-192, December 1999.
- [5] Z. Ling, "Expected value method for fuzzy multiple attribute decision making," *Journal of Tsinghua and Technology*, vol. 11, no. 1, pp. 102-106, February 2006.
- [6] J. Duggan, J. Byrne and G. Lyons, "A task allocation optimizer for software construction," *IEEE Software*, vol. 21, no. 3, pp. 76 – 82, May-June 2004.
- [7] S. T. Acuña, N. Juristo, "Assigning people to roles in software projects," *Software-Practice and Experience*, vol. 34, no. 7, pp. 675 – 696, June 2004.
- [8] R. K. Roy, *Design of Experiments Using the Taguchi Approach*, John Wiley & Sons, Inc., 2001.
- [9] P. Ross, *Taguchi Techniques for Quality Engineering*, McGraw-Hill, 1996.
- [10] R. Ho-Leung, "Using Analytic Hierarchy Process (AHP) Method to Prioritise Human Resources in Substitution Problem," *International Journal of the Computer, The Internet and Management*, vol. 9, no. 1, 2001.
- [11] L. D. Otero, G. Centeno, A. Ruiz-Torres, and C. E. Otero, "A Systematic Approach for Resource Allocation in Software Projects," *Computers and Industrial Engineering*, vol. 56, no. 4, pp. 1333 – 1339, May 2009.
- [12] S. Petrovic-Lazarevic, "Personnel selection fuzzy model," *International Transactions on Operation Research*, vol. 8, no. 1, pp. 89-105, January 2001.
- [13] A. Golec and E. Kahya, "A fuzzy model for competency-based employee evaluation and selection," *Computers & Industrial Engineering*, vol. 52, no. 1, pp. 143-161, February 2007.
- [14] S. B. Yaakob and S. Kawata, "Workers' placement in an industrial environment," *Fuzzy Sets and Systems*, vol. 106, no. 3, pp. 289-297, September 1999.
- [15] G. S. Liang and M. J. Wang, "Personnel placement in a fuzzy environment," *Computers and Operation Research*, vol. 19, no. 2, pp. 107-121, February 1992.
- [16] A. Drigas, S. Kouremenos, S. Vrettos, J. Vrettaros, and D. Kouremenos, "An expert system for job matching of the unemployed," *Expert Systems with Applications*, vol. 26, no. 2, pp. 217-224, February 2004.
- [17] L. D. Otero and C. E. Otero, "A fuzzy expert system architecture for capability assessments in skill-based environments," *Expert Systems with Applications*, vol. 39, no. 1, pp. 654-662, January 2012.
- [18] B. Liu, "A survey of credibility theory," *Fuzzy Optimization and Decision Making*, vol. 5, no. 4, pp. 387-408, October 2006.
- [19] B. Liu and Y. K. Liu, "Expected value of fuzzy variable and fuzzy expected value models," *IEEE Transactions on Fuzzy Systems*, vol. 10, no. 4, pp. 445-450, August 2002.
- [20] B. Liu, *Theory and practice of uncertain programming*, Physica-Verlag Heidelberg, 2003.
- [21] Y.-K. Liu and B. Liu, "Expected value operator of random fuzzy variable and random fuzzy expected value models," *International Journal of Uncertain, Fuzziness and Knowledge-Based Systems*, vol. 10, no. 4, pp. 195-215, August 2002.
- [22] Y. Chen, R. Y. K. Fung and J. Tang, "Fuzzy expected value modeling approach for determining target values of engineering characteristics in QFD," *International Journal of Production Research*, vol. 43, no. 17, pp. 3583-3604, September 2005.
- [23] A. Gonzalez, "A study of the ranking function approach through mean values," *Fuzzy Sets and Systems*, vol. 35, no. 1, pp. 29-41, March 1990.

Measuring the Impact of Security and Reliable Messaging on the Transport Layer for Medical Messaging Applications

H. Keith Edwards¹, Justin Fyfe², Duane Bender², and Suseelan Vigneswaran³

¹Department of Computer Science, University of Hawaii – Hilo, Hilo, Hawaii, United States (hedwards@hawaii.edu)

²Mohawk Applied Research Centre, Mohawk College, Hamilton, Ontario, Canada (Justin.fyfe1@mohawkcollege.ca, duane.bender@mohawkcollege.ca)

³Department Of Electrical and Computer Engineering, University of Waterloo, Waterloo, Ontario, Canada (s3vignes@uwaterloo.ca)

Abstract - *In the realm of medical informatics, having secure and reliable messaging is essential in order for medical information systems to gain acceptance in clinical practice. In addition to these system characteristics, medical practitioners work in a time sensitive context and expect that implemented systems will be able to provide the performance necessary to deliver information in as expeditious a manner as possible. The research in this paper looks at the overhead of enabling security (the ws specification) and reliable messaging on the transport layer for medical messaging systems. Our results show that the factor of reliable messaging along with the interaction of the number of users and message type play statistically significant roles in the variance of the response time. Furthermore, there is a tendency toward lower response times (11/12 cases) and lower standard deviations (10/12 cases) for experiments where security and reliable messaging were both disabled compared to when both factors were enabled. However, this latter finding is not always true at a statistically significant level (7/12 cases) given the noise in the experimental environment.*

Keywords: Security, Electronic Medical Records, Performance Analysis.

1 Introduction

Security and reliable messaging are important to the transmission of electronic medical records in the context of the modern health organization. This paper looks to measure the impact of reliable messaging and security on the transport layer of medical information systems. The rest of this paper is organized as follows. Section 2 presents the related work in the field. Section 3 provides an overview of the system architecture and the experimental design for this research while section 4 provides the results of the experiment and a discussion of their impact. Section 5 presents the conclusions of this work. Finally, section 6 presents ideas for future work.

2 Related Work

There are three main areas of inquiry related to the research discussed in this paper. First, there is the area of standards for electronic health records, which includes the formatting of these records as well as techniques for their transmittal and retrieval. A second important area in the literature are societal factors and concerns, which relate mainly to issues of privacy and misuse whilst examining how appropriate security and policy can help to mitigate these concerns. The final area of related research looks at implementations of electronic health records and examines aspects such as their organizational impact (e.g. changes in workflow) as well as their efficiency.

2.1 Electronic Health Records: Standards, Formatting and Messaging

The first area of related research concerns the different standards available in the domain of electronic health records and ways to format electronic health records. One of the most popular standards in the arena of eHealth is HL7 v3. According to its creators, HL7 v3 is “the world’s leading standard for the electronic interchange of healthcare information.” [9]. Amongst the key concepts of HL7 v3 are:

- Use of Unified Modeling Language
- The Reference Information Model (RIM) – The RIM is the fundamental model in HL7 and is comprised of about 70 different classes based primarily with entities, roles, participation, acts, role links, and act relationships serving as the fundamental classes.
- Domain Message Information Model (D-MIM) – The D-MIM serves as messaging model for a particular healthcare domain.
- Refined Message Information Model (R-MIM) – When the D-MIM is further refined to a more specific set of classes, attributes, and relationships needed for a particular group of messages, a R-MIM is produced as a result.
- Hierarchical Message Description (HMD) – Once an R-MIM is developed, “it can be used to derive a model for a specific reference area.” The derived model has a

different structure to “make it more appropriate for the development of messages.”

- Message Type – An HMD can be further constrained to meet the needs of a particular message resulting in a Message Type. [10].

In addition to these elements, HL7 has the clinical document architecture (CDA). According to Nelson, “the CDA is a document markup standard that specifies the structure and semantics of a clinical document (such as a discharge summary, progress note, procedure report) for the purpose of exchange” [14].

HL7 is a tremendously popular standard in the field of eHealth. In a literature review relating to integration of patient data between systems, Cruz-Correia et al. reviewed 3124 articles that were published over the period from 1994-2004 and found that HL7 was the predominant messaging standard indicated in these papers [7].

Of course, HL7 is not the only standard in the field and there are other standards such as OpenEHR. Furthermore, there are several academic constructs for the structure of electronic health records such as Kukafka et al.’s redesign of EHRs to make them more useful to public health officials [13] or Ammon’s architecture for a knowledge based electronic health record [2].

2.2 Societal Factors and Concerns

The literature in this sub-category looks at societal concerns for health records such as privacy, misuse, and security. Several studies [8,15,17] use ethnography as a tool to understand how health records are used in practice. The findings in these types of studies can be used to structure electronic health record so that they are efficiently adopted by health care practitioners and produce a minimal amount of social inertia when incorporated into the larger organization.

For example, Osterlund [15] undertook a 15-month ethnographic study exploring the work practices of doctors and nurses in documenting patient care. The central finding in this study is that documents have spatial and temporal properties as well as participants. The author hypothesizes that medical personnel can be resistant to large scale information systems, since these disrupt neatly constructed and maintained places for collaboration. He also explains that since documents represent spaces, this accounts for recording the patient history in multiple different documents.

Osterlund also notes in a 2004 article that “In the process of getting admitted, typical patients have their histories taken more than 15 times and these will be recorded in more than 34 information systems, some electronic, some paper-based.” [16]

In the spirit of Osterlund, Hardstone et al. [8] is another ethnography paper on the use of health records in a clinical

setting. This paper presents an ethnography study of community mental health teams in the United Kingdom and the ways in which health practitioners in these contexts create and use health records. As in Osterlund, the researchers find that there is a collaborative nature of the work and that documents have spatial and temporal properties. In addition, the researchers found that practitioners used verbal communications and hand-written notes as drafts of their findings before formalizing these findings in electronic medical records. The researchers argue that pure electronic records could impact the ability of teams to revise their findings since electronic records are viewed as final drafts.

Patients also use electronic health records so understanding their use in this context is important. Enrico Maria Piras and Alberto Zanutto [17] provide a detailed look at how patients use Personal Health Records. The researchers in the article conducted numerous semi-structured interviews with patients in order to see how they kept and used personal health records.

Amongst the findings here are that no one teaches patients how to organize the documents, but that doctors have particular requirements for the amount of order and detail in the records. Second, patients interact with the records by annotating, highlighting, and integrating the records. Third, patients have three general areas where they interact with records:

1. Crossroads (e.g. refrigerator)
2. Archive (e.g. folder)
3. Archive in Use (e.g. diabetic records).

A final finding is that patients have emotional attachments to records (e.g. pregnancy book). In each case, the researchers provide the implication for system design to deal with these findings, e.g. support for annotations and highlighting.

In addition to concerns as to how medical records are employed in actual practice, security of medical information is of paramount importance, particularly in regard to legislation such as HIPA (Health Insurance Privacy Act) and other initiatives throughout the world.

Angst and Agarwal [3] look at individual attitudes toward the adoption of electronic health records in the wake of privacy concerns. In order to do this, the researchers designed a partial factorial design and surveyed 366 participants concerning their attitudes toward EHR adoption. The researchers found that positively framed arguments can lead people to the adoption of electronic versions of their health records.

In addition to having people adopt electronic health records, securing the information and detecting intrusions after adoption is important. Sokolova et al. [21] developed a text analysis tool to detect personal health information (PHI) in heterogeneous text documents. The researchers tested the

system on a variety of data and were able to eliminate most non-PHI documents such as works of fiction and music. The system managed to detect most files with PHI, but yielded a couple of false positives for a health care worker resume and for a blank insurance form.

2.3 Implementations, Case Studies and Performance

A final area of related work concerns actual implementations and case studies related to the implementation of systems that support electronic health records. As one might expect, there are numerous implementations in various jurisdictions around the world. Hence, we present a representative sample of the literature on the subject.

Early papers such as Hooda et al. [11] in 2004 focused solely on viewing and editing electronic health records. However, the scope and scale of implementations have evolved greatly in the past decade. Some of the newer implementations focus on new paradigms such as context-aware mobile systems [20] and the sharing of virtual composite electronic health records from multiple distributed sources [12].

A second category of papers looks at implementations that span large organizations and even nation-wide efforts. For example, Venoit [23] examines the use of an electronic health records system in the veteran's administration hospitals for the treatment of patients with diabetes. Reidl [19] also examines the impact of an implementation across multiple organizations by examining the implementation of an electronic health records system in three oncology clinics in Austria. Nationwide efforts such as Britain's National Programme for Information Technology (NPfIT) [4] and Denmark's BEHR [6] are examined have also been examined by researchers. While accidental problems are different across all these implementations, recurring themes from large-scale software engineering efforts such as requirements and workflow definitions [18,22] are common to all these efforts.

There are quite a number of implementations such as [1,24] that focus on security. Yee et al. [24] focuses on creating a HIPAA compliant personal health record on a flash device, while Adams [1] looks at employing a traffic light metaphor (red, yellow, green) that allows patients to control access to various aspects of their medical records, e.g. setting one aspect to green to allow everyone to see it.

Cruz-Correia et al. [7] conducted an extensive literature review relating to integration of patient data between systems. The authors reviewed 3124 articles that were published over the period from 1994-2004. They categorized the articles by projects and then proceeded to look at general trends. The authors found that HL7 was the predominant messaging standard and that message-based projects were also more common than middleware solutions. The authors also found

that web-based services were becoming more common during this time period as were web-based solutions. The authors found that quite a few of the studies were missing a discussion on error detection and discussions of impact evaluation for the systems.

There has also been little study of performance analysis for electronic health record systems. However, Barua et al. [5] develop a cloud computing architecture for security and patient-centric access control known as EPSAC. Their system "provides an architectural model of an eHealth care system". The authors also demonstrate how their architecture provides for secure communication and has a patient-centric access control policy. The research here is mainly focused on the secure architecture. However, the researchers do evaluate the computation time of the encryption/decryption functions and simulate the protocol against a DoS attack. The performance analysis here is mostly analytic and simulation based whereas our work is solely empirical for standard security measures and reliable messaging in an industrial implementation context.

3 System Architecture & Environment

In this section, we provide an overview of the system under consideration, its implementation environment, and the experimental design.

3.1 System Architecture & Information Flow

This section presents an overview of the system architecture and will provide an extensive discussion of the transport layer. Figure 1 shows the overall architecture of the system with its four separate subcomponents. In particular, the system under examination has subsystems consisting of a client, an application-programming interface (Everest), the Windows Communication Foundation (WCF), and the service. In the case of our experiment, the service merely provides an echo in order to avoid any confounding effects on the outcome variables due to differentiated services.

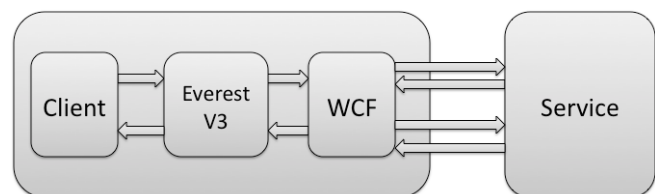


Figure 1: System Architecture

The message flow begins with the client sending a pointer in memory of the data it wants to serialize to the Everest API. Everest takes this pointer and represents the structure in the XML Implementable Technology Specification (ITS); ITS is a set of rules for representing structures "on the wire", i.e. the set of XML rules for parsing HL7 messages. The Everest API then pushes the "on the wire" message to the WCF.

After receiving the "on the wire" message from the Everest API, the WCF implements a set of transport rules such as reliable messaging, security, and ws addressing. Depending upon the message configuration, WCF takes care of transporting the message to the service and implementing the proper protocols. The number of messages exchanged between WCF and the service are dependent upon the system configuration.

After WCF finishes the "conversation" with the service, it then passes an XML message (i.e. the response) back to the Everest API. Everest then employs the ITS rules to parse the message into the HL7 Reference Information Model (RIM) or the Refined Message Information Model (R-MIM). The Everest API then passes a pointer back to the client application.

3.2 Experimental Design and Environment

In order to test the impact of reliable messaging and security protocols on the efficiency of transmitting electronic health records, we utilize a full factorial design with replications as summarized in Table 1.

Table 1: Experimental Design

Factors	
Factor	Levels
Reliable Messaging	Yes/No
Security	Yes/No
Concurrent Users	1/2//4/8
Type of Message	I/II/III
Response Variables	
Response Time	
Throughput	
Total Bandwidth on the Wire	
Errors	

In order to best gauge the impact of reliable messaging and security on the performance of systems transmitting electronic health records, we include several factors. The first factor is reliable messaging, which has two levels to indicate whether this protocol is enabled or not. The second factor has two levels and is indicative of whether or not there is security enabled for the transmittal of the messages. The experimental design also includes factors for the number of concurrent users (1,2,4,8) and the type of message (I,II,III). The last factor examines whether or not the type of message has any impact on the overall performance of the system.

The experiment examines several different response variables including the response time, throughput, bandwidth on the wire, and the number of errors encountered. The experiment was run with 100 replications in order to ensure consistent results with a minimal amount of variance due to noise in the experimental data.

The experiment was run on a Dell Optiplex 980 with a 2.80-gigahertz processor capability and 8 gigabytes of memory. The machine was running under the Windows 7 64 bit operating system with service pack 1 installed.

4 Discussion and Results

In this section, we provide a summary of the results from our experiment. Table 2 displays the results of our experiment in terms of response time for the system.

Table 2: Experimental Design

Experiment Number	Reliable Message	Security	Number Users	Message Type	Avg. Response Time	Std. Dev Response Time	Confidence Value (99%)	Low CI	High CI
1-1-1-1	Yes	Yes	1	1	32.3	38.5	9.9	22.4	42.2
1-1-1-2	Yes	Yes	1	2	43.7	7.4	1.9	41.8	45.6
1-1-1-3	Yes	Yes	1	3	79.8	5.4	1.4	78.4	81.2
1-1-2-1	Yes	Yes	2	1	100.8	123.3	31.8	69.0	132.5
1-1-2-2	Yes	Yes	2	2	84.9	99.5	25.6	59.2	110.5
1-1-2-3	Yes	Yes	2	3	105.1	102.3	26.3	78.8	131.5
1-1-4-1	Yes	Yes	4	1	163.2	133.8	34.5	128.7	197.7
1-1-4-2	Yes	Yes	4	2	154.6	132.9	34.2	120.4	188.8
1-1-4-3	Yes	Yes	4	3	137.1	148.1	38.2	99.0	175.3
1-1-8-1	Yes	Yes	8	1	115.9	113.6	29.3	86.7	145.2
1-1-8-2	Yes	Yes	8	2	186.6	138.3	35.6	150.9	222.2
1-1-8-3	Yes	Yes	8	3	160.4	93.5	24.1	136.3	184.5
1-2-1-1	Yes	No	1	1	75.7	41.0	10.6	65.1	86.2
1-2-1-2	Yes	No	1	2	23.9	7.8	2.0	21.9	25.9
1-2-1-3	Yes	No	1	3	39.6	7.8	2.0	37.6	41.6
1-2-2-1	Yes	No	2	1	49.0	34.9	9.0	40.0	58.0
1-2-2-2	Yes	No	2	2	50.5	7.1	1.8	48.7	52.4
1-2-2-3	Yes	No	2	3	77.4	101.4	26.1	51.3	103.5
1-2-4-1	Yes	No	4	1	74.6	93.6	24.1	50.4	98.7
1-2-4-2	Yes	No	4	2	87.4	112.1	28.9	58.5	116.2
1-2-4-3	Yes	No	4	3	137.3	124.2	32.0	105.3	169.3
1-2-8-1	Yes	No	8	1	182.2	144.7	37.3	144.9	219.5
1-2-8-2	Yes	No	8	2	107.3	116.6	30.0	77.3	137.4
1-2-8-3	Yes	No	8	3	216.7	290.3	74.8	141.9	291.5
2-1-1-1	No	Yes	1	1	49.9	36.9	9.5	40.4	59.4
2-1-1-2	No	Yes	1	2	91.4	63.8	16.4	75.0	107.9
2-1-1-3	No	Yes	1	3	47.0	9.3	2.4	44.6	49.3
2-1-2-1	No	Yes	2	1	110.9	97.2	25.0	85.9	135.9
2-1-2-2	No	Yes	2	2	134.9	121.4	31.3	103.7	166.2
2-1-2-3	No	Yes	2	3	78.0	102.0	26.3	51.7	104.3
2-1-4-1	No	Yes	4	1	84.7	80.5	20.7	64.0	105.4
2-1-4-2	No	Yes	4	2	86.1	86.7	22.3	63.8	108.5
2-1-4-3	No	Yes	4	3	230.2	157.9	40.7	189.6	270.9
2-1-8-1	No	Yes	8	1	94.8	22.7	5.8	89.0	100.7
2-1-8-2	No	Yes	8	2	188.6	121.8	31.4	157.2	220.0
2-1-8-3	No	Yes	8	3	129.9	95.5	24.6	105.4	154.5
2-2-1-1	No	No	1	1	23.7	7.8	2.0	21.7	25.7
2-2-1-2	No	No	1	2	43.7	6.7	1.7	42.0	45.4
2-2-1-3	No	No	1	3	69.4	8.4	2.2	67.2	71.5
2-2-2-1	No	No	2	1	85.5	47.0	12.1	73.4	97.6
2-2-2-2	No	No	2	2	31.7	54.0	13.9	17.8	45.6
2-2-2-3	No	No	2	3	88.1	31.8	8.2	80.0	96.3
2-2-4-1	No	No	4	1	76.1	68.2	17.6	58.6	93.7
2-2-4-2	No	No	4	2	75.3	78.3	20.2	55.2	95.5
2-2-4-3	No	No	4	3	75.5	82.9	21.3	54.2	96.8
2-2-8-1	No	No	8	1	95.6	46.5	12.0	83.7	107.6
2-2-8-2	No	No	8	2	53.0	16.1	4.2	48.9	57.2
2-2-8-3	No	No	8	3	170.2	93.8	24.2	146.0	194.3

The first observation that can be gleaned from these results is that experiments with a greater number of users had higher average response times. This is to be expected, since more users will place a greater load on the system. A second observation is that the message type by itself does not seem to make an appreciable difference in the overall response time when factoring in the size of the confidence intervals.

In further analyzing the results from this experiment, it is constructive to look at the differences between the experiments that used both reliable messaging and security and those that did neither. The mean response time for the experiments that used neither reliable messaging nor security was lower in 11 out of the 12 experiments. In addition, the experiments that did not use reliable messaging had lower standard deviations in 10 out of the 12 cases. However, this

amount of significance does not hold up when looking at the confidence intervals. At a 99% level of significance, there was only a difference between the two types of experiments in 7/12 of the cases.

We also conducted an analysis of variance (ANOVA) on our response time results in order to understand the impact of the various factors involved in the experiment. The results of this ANOVA are shown in Table 3.

Table 3: Analysis of Variance

Source	Type III Sum of Squares	df	Mean Square	F	Sig.
Corrected Model	990026.03	47	21064.384	1.373	0.1
Intercept	2285838.89	1	2285838.89	148.95	0
Reliable Messaging	72414.326	1	72414.326	4.719	0.03
Security	6288.633	1	6288.633	0.41	0.52
Users	74878.444	3	24959.481	1.626	0.19
Message Type	75536.54	2	37768.27	2.461	0.09
Reliable * Security	204.464	1	204.464	0.013	0.91
Reliable * Users	71160.659	3	23720.22	1.546	0.21
Reliable * Mtype	65260.591	2	32630.295	2.126	0.13
Security * Users	38022.868	3	12674.289	0.826	0.48
Security * Mtype	14371.456	2	7185.728	0.468	0.63
Users * Mtype	290450.328	6	48408.388	3.154	0.01
Reliable * Security * Users	34886.565	3	11628.855	0.758	0.52
Reliable * Security * Mtype	48361.017	2	24180.508	1.576	0.21
Reliable * Users * Mtype	33489.134	6	5581.522	0.364	0.9
Security * Users * Mtype	58231.019	6	9705.17	0.632	0.7
Reliable * Security * Users * Mtype	106469.986	6	17744.998	1.156	0.34
Error	1473300.64	96	15346.882		
Total	4749165.55	144			
Corrected Total	2463326.66				

Here, both reliable messaging and the interaction between the number of users and message type are significant. The interaction between the number of users and the message type is not surprising given that each of these factors increases the amount of network traffic. For example, message type 3 is significantly larger than message types 1 and 2, and the number of users is a ratio level measurement.

The impact of these factors can also be seen in Table 2 where the standard deviation for the more complicated experiments (multiple users and larger messages) is lower when reliable messaging is disabled. In general, disabling reliable messaging resulted in lower corresponding standard deviations in $\frac{3}{4}$ of the cases as shown in Table 2. In terms of the other factors, we did not find any errors in any of the experiments. Furthermore, the throughput and the total bandwidth on the wire were closely related to the overall response times shown in Table 2.

5 Conclusions and Recommendations

It is essential to have secure and reliable messaging in medical information systems in order to comply with legislative and privacy requirements as well as to gain acceptance in clinical practice. Medical Information systems must also deliver information to practitioners in a quick and seamless manner due to the time sensitive nature of the field.

In this research, we examined the overhead of security and reliable messaging on the transportation of medical messages through the use of a full factorial experiment with multiple replications. The results of this experiment show two significant factors in the variance of the overall response time. First, the reliable messaging is a statistically significant factor as is the interaction of the number of users with the message type. In analyzing the mean response times and standard deviations, we found a tendency toward lower response times (11/12 cases) and lower standard deviations (10/12 cases) when comparing experiments where both reliable messaging and security were enabled compared to where they were both disabled. However, the response time was only statistically significant in 7/12 cases due to the noise created by experimental errors and other factors. Finally, disabling reliable messaging produced lower standard deviations in 18/24 cases although this did not always result in lower response times.

This work is limited in the number of users supported by the system and in the amount of load placed on the system. Hence, we make no claim as to whether these lower response times hold under industrial loading of the system.

6 Future Work

The analysis of variance in our experiment shows the error term to be a bit high, which limits the types of broad conclusions that can be drawn from this experiment. For future work, using a distributed architecture that can handle a greater number of users may limit some of the variance due to user type and message whilst providing more reliable results. Our preliminary work in this area also found that a greater number of users had more stable results when both reliable messaging and security were disabled. Future experiments should focus on scaling the approach to see if this finding holds under industrial loads.

7 Acknowledgements

The authors of this paper would like to acknowledge the continued support of the Mohawk Applied Research Centre at Mohawk College. The authors would also like to thank NSERC for providing the initial funding to develop the EVEREST framework that enabled the development of the measurement tool.

8 References

- [1] Emily K. Adams, Mehoor Intwala, and Apu Kapadia. MeD-Lights: a usable metaphor for patient controlled access to electronic health records. In Proceedings of the 1st ACM International Health Informatics Symposium (IHI '10), Tiffany Veinot (Ed.). ACM, 800-808.
- [2] Danny Ammon, Dirk Hoffmann, Tobias Jakob, and Ekkehard Finkeissen. 2008. Developing an architecture of a knowledge-based electronic patient record. In Proceedings of

- the 30th international conference on Software engineering (ICSE '08). ACM, 653-660.
- [3] Corey M. Angst and Ritu Agarwal. 2009. Adoption of electronic health records in the presence of privacy concerns: the elaboration likelihood model and individual persuasion. *MIS Q.* 33, 2 (June 2009), 339-370.
- [4] David Avison and Terry Young. 2007. Time to rethink health care and ICT? *Commun. ACM* (June 2007), 69-74.
- [5] Mrinmoy Barua, Xiaohui Liang, Rongxing Lu, and Xuemin Shen. 2011. ESPAC: Enabling Security and Patient-centric Access Control for eHealth in cloud computing. *Int. J. Secur. Netw.* 6, 2/3 (November 2011), 67-76.
- [6] Claus Bossen. 2006. Representations at work: a national standard for electronic health records. In *Proceedings of the 2006 20th anniversary conference on Computer supported cooperative work (CSCW '06)*. ACM, 69-78.
- [7] Cruz-Correia R.J., Vieira-Marques P.M., Ferreira A.M., Almeida F.C., Wyatt J.C., and Costa-Pereira A.M. 2007. Reviewing the Integration of Patient Data: How Systems are Evolving in Practice to Meet Patient Needs, *BMC Medical Informatics and Decision Making*, 7:14.
- [8] Gillian Hardstone, Mark Hartswood, Rob Procter, Roger Slack, Alex Voss, and Gwyneth Rees. 2004. Supporting informality: team working and integrated care records. In *Proceedings of the 2004 ACM conference on Computer supported cooperative work (CSCW '04)*. ACM, 142-151.
- [9] Health Level 7. December 5, 2011. 'About HL7', Health Level 7 webpage, <http://www.hl7.org/>, <http://www.hl7.org/>.
- [10] Hinchley, Andrew. 2007. *Understanding Version 3 Guide: A primer on the HL7 Version 3 Communication Standard*. Alexander Monch Publishing, Munich, Germany. ISBN 3-933819-21-0.
- [11] Jagbir S. Hooda, Erdogan Dogdu, and Raj Sunderraman. 2004. Health Level-7 compliant clinical patient records system. In *Proceedings of the 2004 ACM symposium on Applied computing (SAC '04)*. ACM, 259-263.
- [12] Jing Jin, Gail-Joon Ahn, Hongxin Hu, Michael J. Covington, and Xinwen Zhang. 2009. Patient-centric authorization framework for sharing electronic health records. In *Proceedings of the 14th ACM symposium on Access control models and technologies (SACMAT '09)*. 125-134.
- [13] Rita Kukafka, Jessica S. Ancker, Connie Chan, John Chelico, Sharib Khan, Selasie Mortoti, Karthik Natarajan, Kempton Presley, and Kayann Stephens. 2007. Redesigning electronic health record systems to support public health. *J. of Biomedical Informatics* 40, 4 (August 2007), 398-409.
- [14] Nelson, Stuart J., Zeng, Kelly, and Kilbourne, John. 2009. Building a Standards-Based and Collaborative E-Prescribing Tool MyRxPad. In *Proceedings of the 2009 IEEE International Conference on Bioinformatics and Biomedicine (BIBM '09)*. IEEE Computer Society, 210-215.
- [15] Østerlund, C. S. (2008). Documents in place: demarcating places for collaboration in healthcare settings. *Computer Supported Cooperative Work, an International Journal*, 17(2-3), 195-225.
- [16] Østerlund, Carsten (2004): *Mapping Medical Work: Information Practices Across Multiple Medical Settings*. *Journal of the Center for Information Studies*, vol. 5, pp. 35-43.
- [17] Enrico Maria Piras and Alberto Zanutto. 2010. Prescriptions, X-rays and Grocery Lists. Designing a Personal Health Record to Support (The Invisible Work Of) Health Information Management in the Household. *Comput. Supported Coop. Work* 19, 6 (December 2010), 585-613.
- [18] Pressman, R. *Software Engineering: A Practitioner's Approach*. 6th Edition. McGraw Hill Publishing, 2005.
- [19] Christine Reidl, Marianne Tolar, and Ina Wagner. 2008. Impediments to change: the case of implementing an electronic patient record in three oncology clinics. In *Proceedings of Participatory Design 2008 (PDC '08)*, 21-30.
- [20] B. Skov and Th. Hoegh. 2006. Supporting information access in a hospital ward by a context-aware mobile electronic patient record. *Personal Ubiquitous Comput.* 10, 4 (March 2006), 205-214.
- [21] Marina Sokolova, Khaled El Emam, Sean Rose, Sadrul Chowdhury, Emilio Neri, Elizabeth Jonker, and Liam Peyton. 2009. Personal health information leak prevention in heterogeneous texts. In *Proceedings of AdaptLRTtoND '09*, Association for Computational Linguistics, 58-69.
- [22] Sommerville, I. *Software Engineering*. 8th Edition. Addison-Wesley Publishing, 2007.
- [23] Tiffany C. Veinot, Kai Zheng, Julie C. Lowery, Maria Souden, and Rosalind Keith. Using electronic health record systems in diabetes care: emerging practices. In *Proceedings of the 1st ACM International Health Informatics Symposium (IHI '10)*, Tiffany Veinot (Ed.). ACM, 240-249.
- [24] Wai Gen Yee and Brett Trockman. 2006. Bridging a gap in the proposed personal health record. In *Proceedings of the international workshop on Healthcare information and knowledge management (HIKM '06)*. ACM, 49-56.

Developing Nim Game for iPhone

Palani Dharanidharan, Kevin Daimi, Michael Canjar
 Department of Mathematics, Computer Science and Software Engineering
 University of Detroit Mercy
 4001 W. McNichols Rd., Detroit, Michigan 48221
 {palanidh, daimikj, canjarm}@udmercy.edu

Abstract— Mobile games play a vital role in today's society through entertaining and educating the mobile consumers. This paper presents the design and development of an unbiased (perfect information) game, *Nim*, for iPhone. *Nim* consists of N piles each containing M objects, where M varies between piles. Two players take turns removing objects from the piles. This game is one of many games found within Combinatorial Game Theory. Many mathematicians have been studying *Nim* Game theory and have designed some winning strategies enabling players to win the game. The enhancement of the *Nim* game environment to allow the game to be played on hand held devices, particularly on Apple iPhone, is investigated and analyzed. This mobile game application is designed using photo editing tools and developed utilizing a combination of Adobe Flash Builder IDE, Adobe Flex, MXML, and ActionScript technologies.

Index Terms— *Nim* Game, Game Development, iPhone, Software Engineering, Combinatorial Game Theory

I. INTRODUCTION

Many of the products we use today are a result of advances in technology. Mobile phones, laptops, music players and tablets are all the outcomes of scientific discoveries and advanced engineering. Technological advances have impacted our health, our life and the way we communicate with one another. Even though there are side effects to some aspects of technology (pollution, for example), but generally technology has immensely improved quality of living for most people. People started adopting these technologies essentially for their entertainment. Among the entertainment endeavors available, gaming plays a major role regardless of the age factor. Since ancient times, games have been a part of human culture [13]. A game is a doable endeavor involving talent, luck, and patience among parties playing it. In general, two or more persons play a game according to a set of rules, usually for their own pleasure. It is possible for a computer to be one of the players. According to Adams et al [1], "A game is a type of play activity, conducted in the context of a pretended reality, in which the participants try to achieve at least one arbitrary, nontrivial goal by acting in accordance with rules."

A game implies structured playing, usually undertaken for enjoyment and sometimes used as an educational tool [12]. Key components of games are goals, rules, challenges, and interactions. Games commonly involve mental inspiration, physical inspiration, or both. Claypool et al [7] developed a set of game-centric project-based modules to allow students to exercise and participate in various phases of software engineering including requirement engineering, game design, testing, maintenance, and project management. Rankin et al [19] introduced a user centered game design to evaluate massive multiplayer online role playing games for second language acquisition. They concluded that results from their experimental studies indicated that their approach is encouraging as a language learning tool for vocabulary acquisition. Another application was introduced by Campbell et al [6]. They used game design principles to promote physical activity while targeting fun, maintainability, and performance change. The objective of their project was to have game design play a key role in promoting a healthier lifestyle.

Earlier mobile phones were simple communication devices with inadequate computation and communication capabilities. Mobile devices with increased power, faster and reliable communications and higher resolution displays are becoming increasingly common. As the time advanced, mobile phones have been encapsulated by computationally powerful devices with high resolution colored displays and real operating systems. This trend in mobile phone industry attracted mobile game developers [23]. The evolution of mobile games is comparable to the evolution of computer games. At first, mobile games were simple single player games with limited graphics and intelligence. For the past few years, the revolution of smart phones has created more jobs and appealed to more entrepreneurs. More people started developing mobile applications and gaming business is booming. Mobile game applications are simple and useful for education and entertainment.

Based on what has been mentioned above, it is vital to design and develop a mobile game application using the techniques of software engineering. Software

engineering is an engineering discipline, which is concerned with applying adaptable engineering processes that leads to high-quality product satisfying the needs of the stakeholders [18], [22]. Software engineering first emerged in the 1968 NATO software Engineering conference [14], [26]. It is defined as “The application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software; that is, the application of engineering to software” [26]. Currently, various tools are available to assist software engineers in the analysis, specification, design, implementation and verification of software products. The first step in the mobile game development is developing the game requirements using requirements engineering (RE) process. Requirements Engineering (RE) is applied to improve systems modeling and analysis capabilities so that organizations can better understand critical system aspects before they actually build the system [20]. RE is generally a sophisticated interaction and negotiation process involving different stakeholders, such as mobile customers, players, designers, developers, and tools creators. The main goal of RE is to extract functional and non functional requirements for the software product. The functional requirements along with quality attributes and other nonfunctional requirements will constitute the Software Requirements Specification (SRS) [27]. Proper behavior establishes the functionality of a system and there is often a firm correspondence between particular requirements and particular functions of the software product [2], [25]. However, nonfunctional requirements represent constraints or quality metrics that the system should abide by. These requirements may be viewed as parameters of functionality in that they determine how quickly, how accurately, how reliably, and how securely, these functions must operate [2], [25]. After the requirements have been extracted and organized, the next subprocess is to design the software. Software design is the central focus of software engineering. Design patterns are increasingly important. Björk et al [3] have identified more than 300 patterns depicting game mechanics and interaction elements in traditional games and computer based games. Patterns are normally a recommended approach in software engineering. They definitely do not solve all the design problems for a software system, however, if some patterns that fit the problem are available, the design phase will be expedited [8].

The development of mobile application is the process by which an application software is developed for handheld device,s such as mobile devices, media player and tablet computers using their own native and supportive programming language, tools, and operating systems. Our proposed Nim game targets Apple’s iPhone. Therefore, evaluating its usefulness requires

understanding Apple products and consumers. The current releases of iPod touch, iPhone and iPad are now seen as a critical turning point in the history of Apple. In 2010, Apple announced it is relaxing the previous restrictions on the use of third-party development tools for the creation of mobile applications for iOS, the operating system that powers the iPhone [16]. Many service providers started releasing cross platform development tools, which allow users to use their own programming language to develop the mobile game application and disseminate it to the App Store with their native language. Distribution of apple mobile application is done through App Store. Apple requires every developer to register and get the required approval before being able to distribute their application in App Store.

This paper describes the design of a mobile game application system, WinNim, guided by software engineering code of ethics. WinNim game is a mobile application targeted for the Apple’s iPhone based on the ancient Nim game. The requirements engineering principles will be first applied to solicit and gather Nim game’s functional and nonfunctional requirements. The requirements are then analyzed and the game design is pursued. Software engineering process is applied to the the design and development of the mobile game. WinNim is a standalone mobile application system, which runs under the Adobe Flash Builder Platform. The programming tools MXML and ActionScript are used to implement the system. The rest of the paper is organized as follows: Section II provides history, description, and the mathematical theory behind Nim. Mobile game operation is introduced in section III. Section IV targets mobile game functional and non-functional requirements. The Game design, and development are presented in sections V and VI respectively. Finally, conclusions and future work are presented in section VII.

II. NIM GAME DESCRIPTION AND ITS MATHEMATICS

A. Nim Game History and Description

It is believed that the ancient game, Nim, has its origin in China, but that origin is unconfirmed [11]. The current name was devised by Charles Bouton of Harvard University [11], [15]. It is a simple combinatorial game with finite possibilities [21], illustrating basic principles of Game Theory. The meaning of the word Nim is “taking” [15]. Hence, the game’s rules demand taking objects from a given piles of objects on the Nim board. The Nim game is an unbiased, finite game. It is also called a perfect information game, consisting of N piles each containing M objects, where M varies between piles.

Nim is a mathematical game of strategy in which two players take turns removing objects from distinct piles. On each turn, a player must remove at least one object, and may remove any number of objects provided they all come from the same pile [4]. Nim is usually played as “Misere” game in which the player who takes the last object loses. Nim can also be played as “Normal” play game, where the person who makes the last move wins.

The Nim game is played by two players (two persons, or a person and a mobile device) using a single Nim board. The number of objects in each pile is quite arbitrary, except that no pile may initially contain zero objects. A player selects one of the piles, and takes as many objects as she/he chooses including all the objects in that pile. Players cannot, however, pick objects from more than pile simultaneously.

B. Mathematical Theory

The mathematical solution of WinNim game deals with any number of initial piles and objects. The mathematical theory dictates that there is an easily calculated way to determine which player is going to win, and what winning moves determine that win.

In Nim Game Theory, we say that a board is a *Winning Board* (WB) or *Winning Position* if the current player can force a win from that position. The board is a *Losing Board* (LB) or *Losing Position* if the opponent can force a win. Every Nim Board is either a WB or LB. The Nim winning strategy is based on determining whether the board is WB or LB. If it is a WB, all possible winning moves should be found.

Given a board $[X_1, X_2, \dots, X_n]$, where n is the number of piles, and X_i represents the number of objects in pile i , the first step will be to express these numbers in binary and line up their respective bits in columns. Then, the sum of bits in each column is computed. Having done that, the column is classified as to whether or not it is even or odd. The board is called *Even Position* if every column is even. If any column is odd, the board is an *Odd Position*. Another way of determining whether a column is odd or even is to find the binary digital sum (exclusive or) of each column. If the result of xoring the bits of a column is zero, the column is even. Otherwise, it is 1 indicating the column is odd [9]. Every move from a winning position is to a non-winning (losing) position. From a losing position, there is at least one move into a winning position. Figure 1 introduces an example that illustrates what was mentioned above.

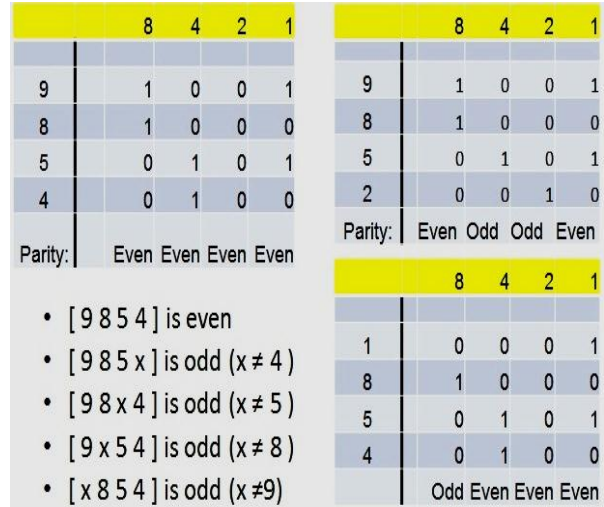


Figure 1. Even/odd sums for board [9, 8, 5, 4]

The winning strategy involves giving the opponent an *Even Position*. The opponent will return an *Odd Position*. This will win every Normal Game, but will lose every Misere Game. This implies that the player needs to change an *Odd Position* to *Even Position*. To achieve that:

- In the first *Odd Column*, find one row (pile) in which there is a “1.” There must be at least one “1” in the first *Odd Column*. Since having all 0s is even.
- In that pile, change the bits in each *Odd Column*.

A pile is singleton if it has 1 element. A board is called Trivial if all non empty piles are singleton. For example let B be a trivial board with N non empty piles [1 , 1 , 1 , . . . , 1]. We having the following conditions:

- If N= 1: We pick the last object. [Lose Misere Game, Win Normal Game]
- If N= 2: Opponent picks the last object. [Win Misere Game, Lose Normal Game]
- If N= 3: [Lose Misere Game, Win Normal Game]
- If N= 4: [Win Misere Game, Lose Normal Game]

Hence ,

- If N is even, then *Normal Game* board B is a *Losing Board*, and *Misere Game* board B is a *Winning Board*.
- If N is odd, then *Normal Game* board B is *Winning Board*, and *Misere Game* board B is *Losing Board*.

A board is a *Decisive Position* if it contains exactly one non-singleton, e.g. [17, 1, 1, 1. . . 1]. Any play of a non-trivial game will pass through a Decisive Position. There will be last non-trivial position and that must be decisive.

III. MOBILE NIM GAME OPERATION

There are two players, often called Player1 (Host Player) and Player2 (Human or iDevice). The type of opponent should be selected before starting the game. A player will not be able to move when the Empty Board (no objects at all) condition is reached. The game will ask the Host Player to choose their opponents. The Host Player has two options: playing against the iDevice (Machine) or playing against another player. Each game will ask the host player to choose the mode of playing. The game concludes when the last object of the last pile is taken. Based on the last object taken, the game could be played in two modes: Normal Game (whoever takes the last object wins), and Misere Game (whoever takes the last object loses).

The next step will be entering the number of piles and clicking on the "SUBMIT" button. Players will be prompted for the number of objects to begin each pile with. Having entered the number of objects in each pile, the "START" button must be clicked. At this point, the system will then decide which player to go first based on tossing a coin. After specifying the pile to remove from, and how many objects to remove, the system will reflect the change in the game status. Now, it is the other player's turn. The player who takes the last object wins (Normal Game), or loses (Misere Game). The system imposes the following restrictions:

- $2 \leq n \leq 9$, where n is the number of piles.
- $m \leq 100$, where m is the number of objects in a pile.
- Only letters are allowed for player's name.
- For coin tossing, players must choose either 1 or 2.

IV. NIM SYSTEM REQUIREMENTS

A. Functional/Nonfunctional Requirements

Before starting the development of the mobile Nim game, the first and foremost task to be considered is engineering the requirements. The outcomes of requirement engineering are functional and non-functional requirements which are critical to the success of any software project [17]. Functional requirements dictate what functions constitutes a solution, and nonfunctional requirements are constraints on the functional requirements [2], [25], [5]. In what follows, Nim and WinNim (the mobile version) will be

used interchangeably. Below is a sample requirements set adopted by the system.

- The system should allow any of the players to start the game.
- The system should allow players to set the options available for the game.
- The system should allow players to choose the level of the game such as easy, medium, or hard.
- The system should allow the host player to choose their opponent (another player or iDevice).
- The system should allow players to go back to the home page in order to access the home page features.
- The system should allow the player to choose the type of game; Normal or Misere.
- The system should allow the player to change the game mode.
- The system should allow players to specify the number of objects and piles.
- The system should allow players to start the game when the number of piles and objects have been entered.
- For Player vs iDevice, the system should allow the iDevice to play first.
- On each turn, the system should enable one player's play button and disable the opponent's play button.
- The system should not allow the same user to play next before the other has completed his play.
- The system should not allow players to change the setting of the board after the game starts.
- The system should not allow any of the players to play when reaching the Empty Board condition.
- For Player1 vs Player2 game, the system should allow players to use the coin tossing feature.
- The system should allow players to take time to read and understand alert messages.
- The system should announce the winner based on the last pick from the WinNim board.
- The system should not allow players to go back to the previous move and make changes.
- The systems should take less than 5 seconds to provide responses (display new page view).
- The system should catch inappropriate input.
- The system should save game status if the game gets disrupted due to hardware or any other problem.

B. WinNim Use case Modeling

Use case modeling is a graphical representation form of functional requirements [17], [18], and [22]. For this modeling, the use case diagram is composed of a number possible scenarios related to the usage of the WinNim game system. In the object-oriented (OO) analysis and design process, use cases are the fundamental input to the

design stage. From a use case diagram, a game developer can gain an unambiguous idea about the system's boundary, actors involved in the system, and the actions the actors can carry out. Figures 2 illustrate the use cases of the WinNim Game System.

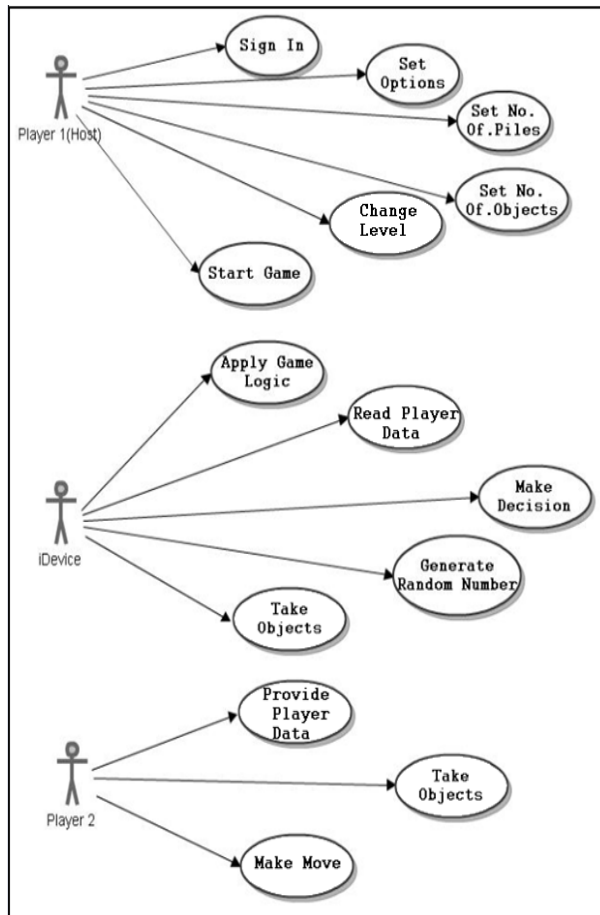


Figure 2. Use Case for WinNim Game

V. NIM GAME DESIGN

Software design is an innovative and creative task. During the design phase, a software developer concludes how to implement a software system to meet customer's need [8]. A good design should exhibit high cohesion and low coupling [22].

A. System Characteristics

The Nim game system is a view-based mobile game application targeted for Apple's iPhone. This game has four modes: *Player1 vs. iDevice – Normal Game*, *Player1 vs. Player2 – Misere Game*, *Player1 vs. Player2*

– *Normal Game*, and *Player1 vs. iDevice – Misere Game*. Each mode demands a totally different game logic.

There are three user groups involved in the system,

1. *Host Player1*: The host player is the owner of the device who has the right to set the options initially and accept opponents. Host Player1 is also in charge of setting the number of piles and objects.
2. *iDevice*: The device (iPhone) acts as a decision maker to end the game by announcing the winner. It can also act as an opponent playing against the host player.
3. *Player2*: Acts as an opponent player playing against host player1.

The WinNim game facilitates the following:

- Controlling all the stages of WinNim mobile game's life cycle.
- Allowing players to choose the game type and play mode.
- Enabling players to make changes before confirming the board.
- Recording all the game movements and displaying them when needed.
- Allowing players to feed the input data necessary for playing the game.
- Offering a toss coin option to select the first player.
- Permitting players to feed the number of piles and objects which should be placed on Nim Board.

B. System Architecture

Architecture design is a model for the system's structure. It illustrates how all of the system's components are connected, and how they collectively function to achieve their goal [24].

WinNim architecture consists of 7 components, and each component is further decomposed into several modules (Figure 3). Each module focuses on a specific kind of task. Our goal is to make these components task-independent and easy to reuse. Hence the modules are loosely coupled. This will enhance the flexibility and maintainability of the entire WinNim game system. Below is a brief description of these components.

Home View component: The home view component encompasses four modules; the Game Start module to handle the player's *Game Start* process, the *Options* module to handle various settings, such as sound, timer and moves recording, the *Challenge* module to choose

between different levels of board, and the *I* module for information about WinNim and its rules.

Main View Component: This component consists of two modules: *Player1 vs. iDevice* module, which allows the host player to choose the device as the opponent, and *Player1 vs. Player2*, which permits the player to choose another player as the opponent.

Play Mode View: It consists of two modules. The *Normal Game* module allows players to start the game. The winner will be the one picking the last object. The *Misere Game* permits players to start the game. Here the winner will be the one having the empty board when it is their turn.

Tossing Coin component: The *Tossing Coin* component is designed for *Player1 vs. Player2* games. It declares the player who is going to start the WinNim game first.

Settings Component: This *Settings* component incorporates three modules. The *Sound* module allows the player to activate or deactivate the option of having sound for each and every button click and background song. The *Timer* module controls start and stop timers to monitor the time taken by a player when removing objects from Nim board in order to win. When initiated by a player, the *Record Moves* module enables the system to track the opponent's moves. This will allow the player to play better to win.

Input Data component: The Input Data component embraces two modules. This component handles all possible errors made by the player while feeding input data. The *Character* Module will check and only accept the input data if it is all letters. The *Number* Module will check and accept the input data if they are all digits. If an error exists, this component will display alert messages.

Random Piles/Objects Generator Component: The *Random Piles/Objects Generator* component is comprised of two modules. These two modules are used by the Challenge Component. The input for number of piles is generated by *PilesRG* module and the number of objects in each pile is generated by *ObjectsRG* module.

Decision Maker Component: The *Decision Maker* Component embraces two types of modules. The *Winning Board* module will evaluate an input board as an Odd or Winning board based on the Nim Sum. If it finds the value of Nim Sum is non-zero it will try to convert the given board into an odd board with a Nim sum value of zero. The *Losing Board* module tries to force an Even or Losing board. If it finds that the value of Nim Sum is

zero, then the system will randomly select any number of objects from one pile.

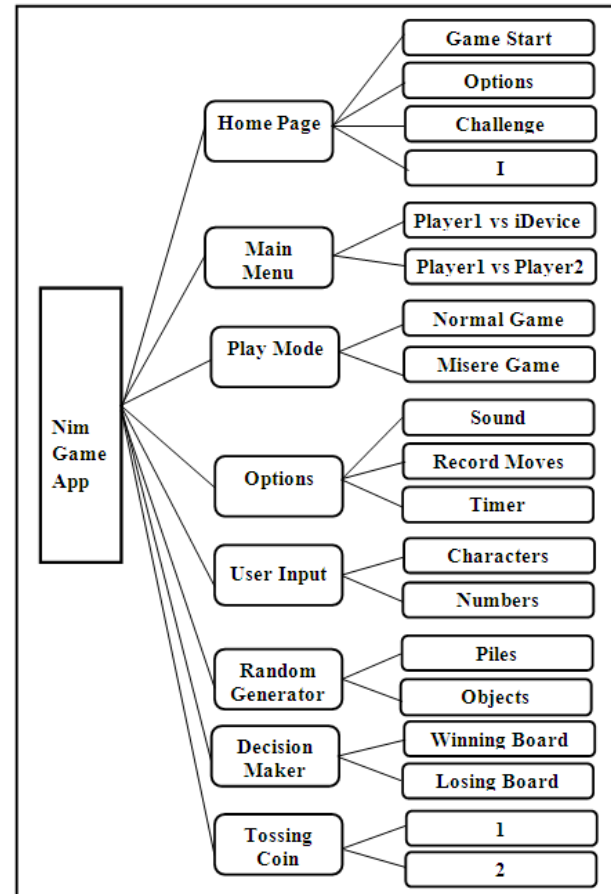


Figure 3. WinNim Architecture

VI. WINNIM GAME DEVELOPMENT

Adobe technologies are popular among developers for design and animation. Adobe Flash Platform tools and ActionScript 3.0 code are used to build Adobe AIR (a multi-operating system, multi-screen runtime) applications for the iPhone and iPod. These applications are distributed, installed, and run just like other iPhone applications [10]. It is possible to develop mobile applications in Flex with similar facilities and quality as the desktop platforms [10]. Many existing Flex components have been extended to work on mobile devices, including the addition of support for touch based scrolling. Flex also contains a set of new components designed to make it easy to build applications that follow standard design patterns for phones and tablets. The Adobe Flash Builder IDE permits developers to select the targeting hand held device for which the application is being developed. Since we are developing the Nim Game for iPhone, the operating system for Apple, iOS, was targeted.

The development of the game system was guided by the software engineering process. Applying the game requirements and game design principles, the Nim game was developed using the Adobe technologies. Apple devices and software have unique features. Hence special attention is needed when designing software for iPhone. There are three templates available to develop complete systems based on the application type. These include Blank Template, View-Based Application, and Tabbed Application. Win Nim was developed using View-Based Applications.

The developed WinNim game mobile application using AIR was then run through the simulation software targeting the output device, iPhone. The following figures are the screenshots of the WinNim game developed with iPhone display specification.

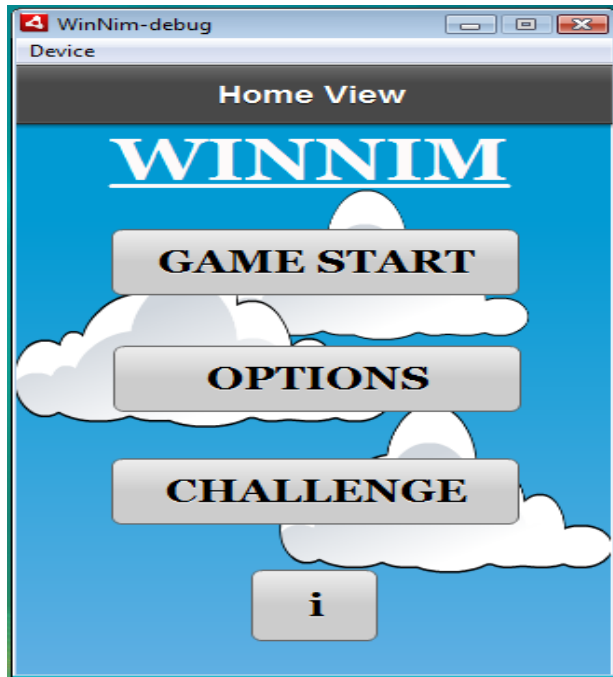


Figure4. Home View

Figure 4 illustrates the WinNim Game's main page. All the view pages will have a title bar with different title for each view page. This will help us maintain a unified style of the system's interface. The Home View of the Nim game contains the following options:

- **Game Start:** Using the Game Start button, the player can play the game by choosing the opponent for the game and the play mode.
- **Options:** Using the Options page, the player can set up the additional features to be added to the game including sound, timer and moves recording.

- **Challenge:** The challenge view allows the player to see all possible levels of Nimboard with predefined number of piles and objects.
- **I :** This is the Information view, which allows the player to learn about the WinNim game and the rules for playing the game.



Figure 5. Menu View

The WinNim Menu View, which is demonstrated in Figure 5, contains the following options:

- **Player1 vs iDevice:** By using this button, the host player can choose the opponent as the iDevice (iPhone) to play Nim game against. This will take the user to the play mode and the Nim Board respectively.
- **Player1 vs Player2:** Using this button, the host player can play against another player. The play mode and the Nim Board will then follow.

The Nim Board View is illustrated in Figure 6. This view is presented after selecting the opponent player and the mode of play. The Nim Board contains ' n ' number of piles and ' m ' number of objects in each pile. It has three buttons:

- **Input:** The number of piles and number of objects are the two inputs needed to set the initial Nim Board. The system will check for constraints violations and provide appropriate alerts if needed.
- **Play:** In WinNim game, each player takes objects when it is their turn. When a player is conducting moves, the system prevents the opponent from carrying out any move.
-



Figure 6. Nim Board View

- **Taking Objects:** In each turn, the system should allow the player to remove objects from a pile. This is achieved through overwriting the number in that pile. The system will check for possible errors, such as the entered number exceeding the current contents of that pile, the entered number is greater than 100, a negative number is encountered, or non-digits are discovered.
- **Decision Maker:** The game's logic will check for the Empty Board condition. Once the empty board is

reached, the system will verify the game mode. Based on the game setting, the winner is declared.

VII. CONCLUSIONS AND FUTURE IMPROVEMENTS

Games targeted for Apple devices are constantly increasing. Since there are thousands of games available in Apple's App Store, it is critical for the design and development of any mobile game not to deviate from the approach taken by other games in the market. This paper presented the requirement analysis, design and development of WinNim game application for Apple Devices, particularly for iPhone. The designed game is for a well-known ancient mathematics-based game, Nim. Unlike many other games, WinNim mandates calculations and thinking before making a move. An unthoughtful or random move will most likely result in losing the game.

Future developments of WinNim will include more challenging levels for playing the game, adding multiplayer option, and online opponents. By improving the rich user interface with all possible available features and targeting other apple devices, iPod touch, Mac [Desktop Application], and iPad, it is hoped that the upcoming version of WinNim will be available in the App Store.

REFERENCES

- [1] E. Adams, and A. Rollings, *Game Design and Development*, Prentice Hall, 2007.
- [2] A. Anton, "Goal-Based Requirements Analysis," in *Proc. the 2nd IEEE International Conference on Requirements Engineering (ICRE '96)*, Colorado Springs, 1996, pp. 136-144.
- [3] S. Björk, and J. Holopainen, *Patterns in Game Design*, Charles River Media, 2004.
- [4] C. L. Bouton, "Nim, A Game with a Complete Mathematical Theory," *The Annals of Mathematics*, Vol. 3, No. 1 / 4, pp. 35- 39, 1992.
- [5] I. Bray, *An Introduction to Requirement Engineering*, Harlow Essex: Addison Wesley, 2002.
- [6] T. Campbell, B. Ngo, and J. Fogarty, "Game Design Principles in Everyday Fitness Applications," in *Proc. the ACM 2008 Conference on Computer Supported Cooperative Work (CSCW'08)*, San Diego, 2008, pp.249-252.
- [7] K. Claypool, and M. Claypool, "Teaching Software Engineering Through Game Design," in *Proc. the Tenth Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE'05)*, Monte da Caparica, Portugal, 2005, pp. 123-127.
- [8] K. Daimi, and L. Nowicki, "Software Engineering for Pinochle Game Development," in *Proc. the International Conference on Software Engineering Research and Practice*, Las Vegas, 2009, pp. 77-84.

- [9] D. J. Davis, "The Game of Nim: Expository Paper", 2006, Available: http://scimath.unl.edu/MIM/files/MATEExamFiles/Davis_MAT_Exam_ExpositoryPaper.pdf.
- [10] Developing Mobile Application with Flex and Flash Builder, Adobe Systems Inc., 2011, Available: http://Help.Adobe.Com/En_US/Flex/Mobileapps/Developing_Mobile_Apps_Flex.Pdf.
- [11] C. M. Freeman, *NIM: Serious Math with a Simple Game*, Waco: Prufrock Press, 2005.
- [12] Game, Wikipedia, 2012, Available: <http://en.wikipedia.org/wiki/Game>.
- [13] J. Huizinga, *Homo Ludens: A Study of the Play-Element in Culture*, Beacon Press: Boston, 1950.
- [14] P. Naur, and B. Randell, "Software Engineering: Report of a Conference Sponsored by the NATO Science Committee," in *Proc. NATO Software Engineering Conference*, Garmisch, Germany, 1968, pp.9-65.
- [15] Nim, Wikipedia, 2102 Available: <http://en.wikipedia.org/wiki/Nim>.
- [16] S. Perez, Apple Relaxes Restrictions on Mobile App Development, 2010, Available: <http://www.readwriteweb.com/mobile/2010/09/apple-relaxes-restrictions-on-mobile-app-development.php>.
- [17] S. L. Pfleeger and J. M. Atlee, *Software Engineering Theory and Practice*, Upper Saddle River, NJ: Pearson Higher Education, 2010.
- [18] R. Pressman, *Software Engineering: A Practitioner's Approach*, McGraw-Hill, 2010.
- [19] Y. Rankin, M. McNeal, M. Shute, and B. Gooch, "User Centered Game Design: Evaluating Massive Multiplayer Online Role Playing Games for Second Language Acquisition," in *Proc. the ACM SIGGRAPH Sandbox Symposium*, Los Angeles, 2008, pp. 43-49.
- [20] W. Robinson, S. Pawlowski, and V. Volkov, "Requirements Interaction Management," *ACM Computing Surveys*, Vol. 35, No. 2, pp. 132-190, 2003.
- [21] G.A. Sarcone, Nim Game: A Binary Challenge, 2012, Available: www.archimedes-lab.org/game_nim/nim.html.
- [22] I. Sommerville, *Software Engineering*, Boston: Addison Wesley, 2011.
- [23] R. Spanek, P. Kovar, and P. Pirkl (2007) "The BlueGame Project: Ad-hoc Multilayer Mobile Game with Social Dimension," in *Proc. The 3rd International Conference on Emerging Networking Experiments and Technologies (CoNEXT'07)*, New York, 2007.
- [24] R. N. Taylor, and A. Hoek, "Software Design and Architecture: The Once and Future Focus of Software Engineering," in *Proc. 2007 Future of Software Engineering*, Washington, DC, 2009, pp. 226-243.
- [25] A. van Lamsweerde, *Requirements Engineering: from System Goals to UML Models to Software Specification*, Chichester: Wiley, 2009.
- [26] J. A. Wang, "Towards Component-Based Software Engineering," *Journal of Computer Science in College*, Vol. 16, No. 1, pp. 177-189, 2000.
- [27] K. E. Weigers, "Karl Wieggers Describes 10 Requirements Traps to Avoid," *Software Testing and Quality Engineering*, Vol. 2, No. 1, pp. 34-40, 2000.

Relational Database Schema Evolution using Fragile Watermarking Approach

Sri S. Ravichandra and Dr. DVLN Somayajulu
 Department of Computer Science and Engineering
 National Institute of Technology, Warangal, (A.P.) India

ABSTRACT — *This paper discusses a Hybrid approach to Fine Grained Isolation of Changes in Relational Database Schema for guaranteeing the integrity of the database. Many of the researchers addressed various issues related to Core Schema Evolution which includes identifying and incorporating changes to the schema while preserving the consistent state of the schema as well as propagating the changes to the data associated with the schema. The methods proposed by researchers were inadequate in characterizing, localizing and isolation of changes to the database. The nature of change (deletion, addition or modification) cannot be known. The proposed approach uses a method by which changes can be localized, characterized and recovered. This approach uses the concept of fragile watermarking to capture the changes and design matrix is used to represent the database schema. An algorithm is proposed for incorporating the changes related to Schema and it can easily be extended to find out the integrity preservation in the presence of schema changes.*

Keywords — **Fine Grained Isolation, Design matrix, Schema evolution.**

1. INTRODUCTION

With enormous advancements in database technology there is a need to integrate database systems developed earlier. This is due to the instability of database systems in their initial implementation. The reason for this instability is due to the design of database schemas in a self-contained way, which results in serving a limited application domain. Hence integration is required for transforming and integrating the source schemas in to global schemas. But incorporating such integrations will initiate a variety of changes in database schemas and also becomes more complicated if large quantities of data are involved. Such types of changes are becoming relatively frequent in such database integrations and often becoming troublesome for database administrators and developers.

Many researchers believed that there is an increase in programmer's effort due to database schema modifications which is because of the differences in the estimates with respect to resources, effort and cost. Hence the modification in the database schemas leads to schema evolution [1]

problem which is observed while propagating schema change. With these issues, the researchers suggested that there should a mechanism to ensure of the integrity of the database schema while performing such modifications. It is necessary to propose a method which considers the issue of schema changes by preserving the structural consistency of database schema. Therefore it understood that there should be a mechanism for detecting and localizing the schema changes. The mechanism proposed in this paper is the concept of fragile watermarking technique. Most of the digital watermarking techniques are used in the context of databases is only for copy protection, whereas the proposed fragile watermarking scheme allows legal changes to the database schema and protects the database from illegal changes and hence it can is called fragile watermarking [13]. It is observed that technique of fragile watermarking on database schemas is appropriate because the database relations contain independent tuples with little redundancy and with no enforced relationship between the tuples. Unlike the highly correlated multimedia data and image data whose relative positions are fixed. All these issues are considered as a new technical challenge for fragile watermarking in database schemas. A fragile watermark is some kind of information that is embedded into underlying data for change detection, localization, ownership proof, and/or traitor tracing purposes. Fragile watermarking consists of two basic processes: Watermark insertion and Watermark detection. For watermark insertion, a key is used to embed watermark information into an original database by producing the watermarked database which is used for publication or distribution. With the appropriate key and watermark information, the watermark detection process is applied to any modified/changed database schema for determining its legitimacy. After detecting and localizing the changes to the database schema, it is checked against for various kinds of integrity constraints to ensure the integrity. After preserving the integrity of the database, the behavior of the changes or modifications is known. If the changes are consistent, they are allowed otherwise they are discarded. If the changes are allowed, the effect of these changes on the application are studied and triggered. Thus the databases can always be ensured of integrity. In this proposed paper the database schema is represented in the form of a design matrix. Design matrices [15] were a better choice the amount of information in each cell is reduced to one of 4 values. This makes it easier to encode it for recovery later. It is easy to characterize what changes occurred using this method, since more information about the type of attributes is stored in a design matrix. It is very easy to check for

integrity constraints using design matrices since the relationships between the attributes in each table are clearly shown. Hence the proposed work in this paper can be summarized as follows: The database schemas are converted into design matrix representation and the watermark is inserted. Then some structural changes were applied on the database schema and these changes were identified by verifying the watermark. Then the changes which are consistent are allowed by tracking the database schema by ensuring its integrity. Experimental results of the proposed scheme are shown

Section 2 presents the related work and different issues encountered while addressing the Problem. Section 3 explains the proposed algorithms with some notations used, Section 4 is about Analysis of the present scheme, and Section 5 is about Conclusion and Future work.

2. RELATED WORK AND SOME ISSUES

The Schema evolution problem was first addressed based on traditional database systems. Most of the database systems support a few simple changes automatically, such as adding or deleting record fields, only a few systems [1][2] support more general transformations. In these cases, the administrator/developer is responsible for explicitly describing how to convert the data from its old format to its new format using a special purpose data translation language. This approach is a powerful one, but creation of the transformer is a manual process. Most of the researchers addressed problems related to Core Schema Evolution which includes identifying and incorporating changes to the schema while preserving the consistent state of the schema as well as propagating the changes to the data associated with the schema. Shneiderman and Thomas examine a system and architecture for automatically converting relational databases following changes to its structure [2]. Borgida and Williamson discuss a method that incorporates exceptional facts. These are facts that are consistent with the real world (which has changed), but do not conform to the current structure of the database (which has not changed) [4]. Takahashi describes the concept of hybrid relations to support schema evolution [5]. The research is relatively simple in that it only describes the change due to addition of new attributes to an existing relation. A commercial relational database management system, Evolutionary DBMS (EDBMS) has been developed by Information Research Associates [3]. This system uses temporal concepts to support schema evolution, specifically the evolution of data definitions and supports dynamic restructuring of the schema. Dadam and Teuhola's Non-First-Normal-Form (NF2) model examines schema evolution using temporal definitions [6]. The NF2 relational database permits the storage and manipulation of non-first-normal-form relations. The NF2 with temporal definitions handles time in the form of versions of data and schemas. Clifford and Croker introduce the concept of lifespan in their description of Historical Relational Data Model (HRDM) [7]. Each attribute value in HRDM is associated with a lifespan parameter that defines its period of existence. Ariav examines schema evolution using the Temporally Oriented Data Model (TODM) [8]. The central data structure proposed in this research is the data cube. The data

cube is a temporal data construct in which time, objects, and attributes form the primary dimensions of the stored data. Roddick describes a model that incorporates temporal support into the metadata of the relational database [9]. Specifically, the addition of this temporal support has been investigated with reference to semantics of null values, its effect on integrity constraints and its impact on query languages. Scalas et. al. discuss another model for schema evolution in temporal relational databases [10]. Changes are classified into two types: redefinition and revision. In redefinition, the schema after the change is completely independent of the schema before the change. Applying changes to the most recent schema "revises" the schema.

3. PROPOSED ALGORITHMS AND EXPLANATION

The algorithm proposed in this paper is a hybrid approach for localizing and recovering from changes. To achieve this, design matrix representation is used for database. This algorithm considers the following:

- a) The row and column values depend on the hash value of the respective names of the rows and columns. The amount of information in each cell is reduced to one of 4 values. In this paper the design matrix was represented in the following way.
 - i. No attribute: 0 -> 0
 - ii. Attribute: A -> 3
 - iii. Foreign key: F -> 11
 - iv. Primary key: P -> 17
- b) It is very important to choose the appropriate numbers to be multiplied into each cell. The characteristics that these numbers should possess are as follows.
 - i. The numbers are not dependent on the order of the rows or columns in the database
 - ii. The numbers are not dependent on the number of rows or columns in the database
 - iii. It should be possible to determine these numbers given a new database
 - iv. These numbers should help in characterizing the changes occurring to the database.
- c) Based on these requirements, the scheme used to find the numbers is:
 - i. A number for each row and column is found by $\text{HMAC}(\text{row name or column name})\%100$
 - ii. Assume that the maximum number of columns possible is 100. (this number should be determined based on the application)
 - iii. The preliminary number for each cell[i][j] is found by $\text{row number}[i] * \text{maxCol} + \text{column number}[j]$
 - iv. After finding the preliminary number, it should be checked if it is a multiple of 3, 4, 7, 11 or 17. If it is a multiple, 1 is added to the number until the number is not a multiple.

- d) To recover the database if the previous watermark is known, a set of simultaneous equations must be solved. Since only one DDL statement is being considered at a time, only one row is changed at a time though multiple columns may be changed. In this case:
 - i. Maximum number of changes (variables): number of attributes (I_{Attr})
 - ii. Number of equations available: $I_{Attr} + 1$

Hence, any single change in the form of DDL statement is recoverable. Since it uses the concept of fragile watermarking and design matrix representation for database and both the concepts were combined to form a hybrid approach for solving the schema evolution problem.

The notations used in this approach are given below:

I_{Attr}	Total number of attributes
I_{Tables}	Total number of Tables
Str _{Table names}	A string array of table names
Str _{Attributes names}	A string array of attribute names
Max _{Col}	the maximum number of attributes expected
O_w	Old Watermark
N_w	New Watermark
N_d	New design matrix
K	Watermark embedding key

3.1 Convert to a design matrix:

Design matrix is a representation of the database schema which allows us to easily see the relationships between the tables and the attributes. This representation allows for easy checking of integrity. The main advantage of using this is that it reduces the amount of information stored in each cell so that it is easy to retrieve lost information.

Consider the database:

- T1: A1, A2*, A3
- T2: A3, A1*, A5
- T3: A2, A3*, A4

In a design matrix representation, the above database will be represented as:

	A1	A2	A3	A4	A5
T1	P	F	A		
T2	F		P		A
T3		P	F	A	

Where P = primary key, F = foreign key, A = attribute, T = Table etc.

In this representation, each cell can hold only 1 of 4 values. The design matrix was encoded with digits instead of characters with the following substitutions: a) P = 17. b) F = 11. c) A = 3. d) Otherwise 0 and hence the previous database will look like this:

	A1	A2	A3	A4	A5
T1	17	11	3	0	0
T2	11	0	17	0	3
T3	0	17	11	3	0

3.2 Architecture of the proposed Algorithm

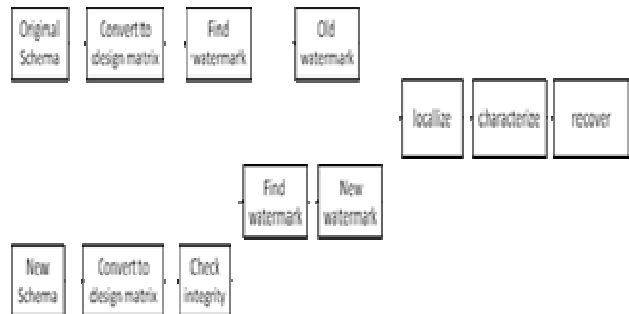


Figure 1: Architectural diagram

3.3 Proposed Algorithms

3.3.1 Algorithm for Convert to a design matrix

```

//Algorithm for Convert to a design matrix
Input: Uninitialized database Table
Output: Initialized design matrix
Method:
1. for i in 0 to ITables do StrTable names[i] ← TableNames[i];
2. end for
3. for i in 0 to ITables do
4. for j in 0 to IAttr do
5. Strncpy (temp,strTables[i][j]);
Call the lookup function to check if this attribute is already
there in the String Attributes Names (pos=lookup()).
6. if pos is equal to size then size++; end if
7. if pos is greater than max
8. max=pos;// Accordingly set the value of the
Matrix[i][pos] end if
9. if the attribute is a primary key then Set the value to 17
end if
10. if the attribute is a foreign key then Set the value to 11
end if
11. if it is attribute then Set the value to 3 end if
12. max=max(lookup());
13. IAttr=max+1;
14. end for
15. end for
    
```

3.3.2 Algorithm for Finding Watermark:

//Algorithm for finding watermark

Input: groups of data to be provided as input

Output: if any changes occur localized and characterized

Method:

1. Find the correct numeral for each cell
2. **for** i in 0 to I_{Tables} **do** RowNum[i] = HMAC (Str_{Table} Names[i] % 100; **end for**
3. **for** i in 0 to I_{Attr} **do** ColNum[i] = HMAC (Str_{Attributes} names[i] % 100; **end for**
4. **for** each cell in the matrix
5. Numeral[i][j] = rownum[i]*maxCol + colnum[j]; **end for**
6. **while** Numeral[i][j] is multiple of 3,7,4,17 or 11
7. Numeral[i][j]++; **end while**
8. Multiply each cell of the design matrix with the numeral
9. **for** each cell
10. Numeral[i][j] = numeral[i][j]*dmat[i][j]; **end for**
11. Find the row and column watermarks
12. **for** i in 0 to I_{Attr} **do**, ColWatermark[i] \leftarrow 0;
13. **for** j in 0 to I_{Tables} **do**, ColWatermark[i] = ColWatermark[i] + Numeral[i][j]; **end for end for**
14. **for** i in 0 to I_{Tables} **do**, RowWatermark[i] \leftarrow 0;
15. **for** j in 0 to I_{Attr} **do**, RowWatermark[i] = RowWatermark[i] + Numeral[i][j] **end for end for**

3.3.3 Algorithm for recovery of the table

//Algorithm for recovery of the table

Input: table of data

Output: recovered table

Method:

1. **if** the number of row watermarks in O_w < number of tables in N_w , then a DROP statement was executed.
2. **for** i in 0 to I_{Attr} **do**
3. diff[i] = $O_w.ColWatermark[i] - N_w.ColWatermark[i]$
4. Add a new row to the bottom of the design matrix
5. **if** diff[i] = 0 then value of the cell is 0 **end if**
6. **if** diff[i] % 3 == 0 then value of the cell is 3 (attribute) **end if**
7. **if** diff[i] % 11 == 0 then value of the cell is 11 (foreign key) **end if**
8. **if** diff[i] % 17 == 0 then value of the cell is 17 (primary key) **end if**
9. **end for**
10. **end if**
11. **if** the number of row watermarks in O_w > number of row watermarks in N_w , then a CREATE statement was executed.
12. Compare the new watermarks with the old watermarks and determine which row was added
13. Match the corresponding attributes
14. Delete that row to get the original table **end if**
15. **if** the number of row watermarks in O_w = number of row watermarks in N_w , then an ALTER statement was executed

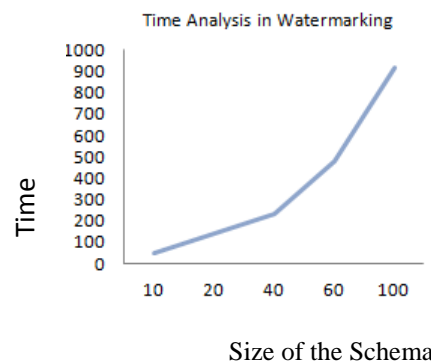
16. Compare the row watermarks of O_w and N_w find which table is changed.
17. Match the corresponding attributes
18. To recover the original row:
19. **if** the diff = 0, then no change
20. Otherwise use the following table
21. **end if**

4. ANALYSIS

There are certain features like watermark insertion, watermark detection & verification, performance to be analyzed in the present scheme .They are as Follows

Time Analysis

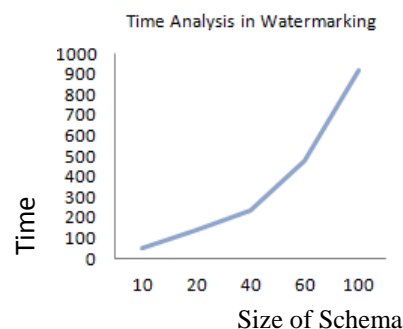
4.1 Watermark Insertion



Analysis on watermark Insertion

1. The Time Complexity on the whole is $O(|v| \times |\gamma|)$, where v is Average no. of Tables in a group and γ is no. of attributes in a relation
2. This graphs shows the time complexity in embedding a watermark vs the size (in terms of Big O notation)
3. This shows that as size of schema progresses time complexity increases in a polynomial fashion
4. Here the size of schema may be taken as no. of tables or total no. of attributes in the schema
5. On the whole, since it is in polynomial time, the algorithm is scalable.

4.2 Watermark Detection and Verification

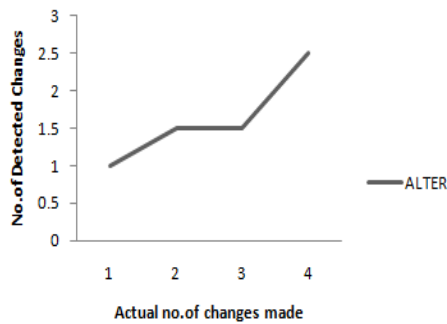


Analysis on Watermark Detection and Verification

1. The Time Complexity on the whole is $O(|v| \times |\gamma|)$, where v is Average no. of Tables in a group and γ is no. of attributes in a relation
2. This graphs shows the time complexity in finding a watermark and verification vs the size (in terms of Big O notation)
3. This shows that as size of schema progresses time complexity increases in a polynomial fashion
4. Here the size of schema may be taken as no. of tables or total no. of attributes in the schema
5. On the whole, since it is in polynomial time, the algorithm is scalable.

4.3 PERFORMANCE ANALYSIS

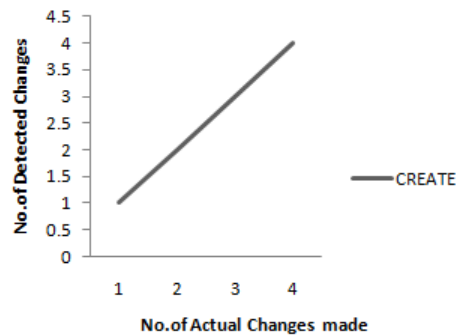
4.3.1 ALTER



Analysis on ALTER

This graph shows that that multiple changes made in the same group can be detected but Localization of changes becomes difficult as the size of schema progresses

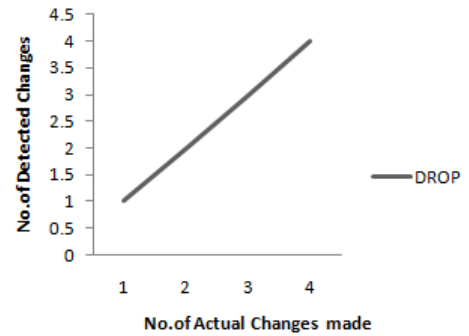
4.3.2 CREATE



Analysis on CREATE

From the above graph it can be inferred that any number of columns added can be detected even as the size of database increases.

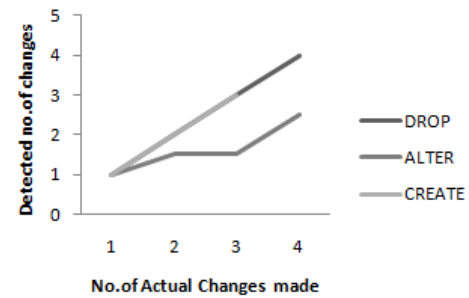
4.3.3 DROP



Analysis on DROP

From the above graph it can be inferred that columns dropped can be detected even as the size of database increases.

4.3.4 CHARACTERIZATION OF CHANGES (DDL)



Analysis:

From the above graph it can be inferred that columns added or dropped can be detected and not all the modifications made to the attribute values can detected and localized.

5. CONCLUSION

In this paper we proposed an hybrid approach for localizing and isolation of schema changes in relation database based on fragile watermarking concept. We also discussed the design matrix approach for localizing, isolation of schema changes in providing solution for schema evolution problem.

References:

[1] NAVATHE, S. *Schema analysis for database restructuring*. ACM Transactions on Database Systems. 5, 2 (June 1980), 157-184

[2] SHNEIDERMAN, B. and THOMAS, G. *Architecture for automatic relational database system conversion*. ACM Transactions on Database Systems. 7, 2 (June 1982), 235-257

- [3] EDBMS - *The Evolutionary Database Management System - concepts and summary*, Technical Report, Information Research Associates, Austin, Texas, (1983)
- [4] BORGIDA, A. and WILLIAMSON, K. E. *Accommodating exceptions in databases and refining the schema by learning from them*. Proceedings of the 11th Very Large Databases Conference. Pirotte, A. and Vassiliou, Y (Ed.), Stockholm, Sweden, (August 1985), 70-81
- [5] TAKAHASHI, J. *Hybrid relations for database schema evolution*. Proceedings of the 14th Annual International Computer Software and Applications Conference. Knafl, G. J. (Ed.), Chicago IL, USA, (1990), 465-470
- [6] DADAM, P. and TEUHOLA, J. *Managing schema versions in a time versioned non-first normal form relational database*. IBM Scientific Research Center Tech. Report 87.01.00, Germany, (1987)
- [7] CLIFFORD, J. and CROKER, A. *A Historical Relational Data Model (HRDM) and algebra based on Lifespans*. Proceedings of 3rd IEEE Int. Conference on Data Engineering, Los Angeles CA (1987), 528-537
- [8] ARIAV, G. *Temporally oriented data definitions - Managing schema evolution in Temporally Oriented databases*, Data and Knowledge Engineering, 6, 1 (1991), 451-467
- [9] RODDICK, J.F. *Dynamically changing schemas within database models*. *The Australian Computer Journal*, 23, 3(1991), 105-109
- [10] SCALAS, M. R., CAPPELI, A., and DE CASTRO, C. *A model for schema evolution in temporal relational databases*. Proceedings of the European Conference on Computers in Design, Manufacturing and Production (CompEuro'93), IEEE Computer Society Press, (1993), 223-231
- [11] RODDICK, J.F. *Schema evolution in database systems- an updated bibliography*, Technical report (1994) CIS-94-012. School of computer and Information Science, University of South Australia
- [12] HUIPING GUO, YINGJIU LI, ANYI LIU, SUSHIL JAJODIA. *A fragile watermarking scheme for detecting malicious modifications of database relations*. Information Sciences: an International Journal, Volume 176 Issue10, May, 2006, Elsevier Science Inc. New York, NY, USA
- [13] YINGJIU LI, HUIPING GUO AND SUSHIL JAJODIA. *Tamper Detection and Localization for categorical data using fragile Watermarks*, Proceedings of the 4th ACM workshop on Digital rights management, 2004, ACM, New York, NY, USA
- [14] YINGJIU LI, *Database watermarking: systematic View, Handbook of Database Security, Applications and Trends*, 2008, Springer US
- [15] SANTIAGO GOMEZ, HOANG DUONG, *Towards a schema representation through design matrices*, Proceedings of the 2005 Australian conference on Software Engineering, IEEE Computer Society Washington, DC, USA, 2005

A Text Messaging Pay For Parking Service (iParked.ca)

D. Deugo¹ and N. Matic²

¹ The School of Computer Science, Carleton University, Ottawa, Ontario, Canada

² Espirity Inc., Ottawa, Ontario, Canada

Abstract - In this paper, we present a case study of the development of the iParked.ca parking service. We discuss the service and its intellectual achievements. We also examine its uniqueness and originality and the steps taken and timeline to develop and launch the service.

Keywords - SMS; Parking; Mobile Phone;

1 Introduction

In this paper, we present a case study of the development of the iParked.ca parking service. iParked.ca [1] is a Short Message Service (SMS) text messaging based parking service that has four simple steps that enable users to take advantage of the ease, comfort and safety of paying for their parking using their mobile phones. The steps, shown in Figure 1, include a one-time, on-line registration, paying for parking using one's mobile phone, receiving expirations notifications on one's mobile phone and receiving transaction receipts via e-mail.



Figure 1. iParked.ca User Steps

After registering one's mobile phone number, default license plate and credit information at iParked.ca, a user can pay for parking using their mobile phone by sending text messages similar to "car p2 2h" to the iParked.ca short code 727533, the numbers which correspond to the letters PARKED on one's mobile phone keypad. The format of the message includes a vendor ("car", for example, is Carleton University's vendor id), the lot/zone (p2) identifier and the duration (2h). This format supports multiple vendors, lots/zones, and durations, such as 90m for ninety minutes. Another message format enables users to pay for parking for an automobile other than their default-registered license plate. After confirming a parking request by sending a text message to the user, the service sends the user an e-mail containing their credit card receipt. Finally, ten minutes before the user's parking is about to expire, they are sent a warning text message of this fact. Vendors signing up for the service choose a three-character identifier, e.g. "car" for the Carleton University, and provide the lot/zone identifiers, the payment schedule for each lot/zone, and merchant information.

By linking users and vendors, the service enables users to pay for parking using their mobile phone from any environment they feel safe in: their warm, dry car, at lunch, from a meeting, from their office, from an appointment, or wherever they happen to be. In addition, they do not have to use or expose money or credit cards to pay. When it is cold, when it is raining, or when a user cannot get back to their car in time, they can simply and easily pay or renew their parking using their mobile phone. iParked.ca is one of the first of its kind in North America.

The remainder of the paper is organized as follows. Section 2 provides a description of the service and the intellectual achievements required to overcome problems in the development of the service. Also include is a discussion of the service's uniqueness and originality. Section 3 describes the timeline up until the launch of the service. Sections 4 and 5 discuss the commercialization and impact of iParked.ca.

2 Description

iParked.ca pay-by-text parking solution is a trademark of Espirity Inc. [4], and fully developed in Ottawa, Canada. For parking payment processing, the solution uses a payment gateway/acquirer that is fully compliant with the PCI Data Security Standard (PCI DSS). For incoming SMS text messages, the solution uses a certified text messaging aggregator as per standards defined by Canadian Wireless Telecommunication Authority (CWTA) [2] that manages and administers Canadian short codes.

From the user's perspective, the first step in using the system is registration, available at the iParked.ca web site. Registration is free and is required only once. During registration, a user's mobile phone number, e-mail address, language indicator, promotions indicator (for receiving notifications about promotions and specials), license plate and province, and billing information (VISA, Master Card are supported) are captured. Billing information is securely stored in the payment gateway. iParked.ca partners with a payment processing gateway vendor to deliver a fully PCI Data Security Standard (DSS) [3] compliant payment solution. Customer sensitive credit card information is not stored in the iParked.ca infrastructure. iParked.ca accesses this information through the gateway's electronic cash register interface technology. The gateway vendor is a level 1 PCI DSS certified vendor.

iParked.ca users can, at any time, reset their password by texting the message "resetpassword" to the short code 727533 (PARKED). The system will verify the user based on their phone number, reset the user's password and send a new one to the e-mail account specified in the user's iParked.ca profile.

Once registered, users can text parking request messages to the short code 727533 (PARKED) to pay for parking at any parking location using the iParked.ca service. The parking request is in the format "car p1 90m" where "car" indicates Carleton University as a vendor (unique three letter identified designated for Carleton University in the iParked.ca system), "p1" indicates unique parking location (in this case parking lot P1), lot, zone or space for which the parking payment is being made, and "90m" indicates the parking duration specified in minutes (in this case it would be for 1.5 hours of parking). Once the text message is received, the user is verified based on the mobile phone number. If the user is not registered, a text message is sent to the user explaining the registration process. The invalid request is logged within the iParked.ca system for later error reporting and administrative support. If the user is registered, the system validates the parking request. If the parking request format and details are invalid, a text message in the user's language of choice is sent both to the user's mobile phone, and to his or her e-mail account, and the invalid request is logged within the iParked.ca system. If the parking request is valid, the parking payment is requested from the payment gateway. For a billing error, a text message in the user's language of choice is sent to the user's mobile phone, an e-mail is sent to the user's e-mail account, and the invalid request is logged within the iParked.ca system. For a successful payment transaction, a parking booking is logged, a confirmation text in the user's language of choice is sent back to the user's mobile phone, and a receipt is sent to the user's e-mail account. The parking confirmation text message contains details of the parking transaction such as license plate and province, parking expiration time, and parking payment, as shown in Figure 2:



Figure 2. Sample iParked.ca Booking

The payment gateway deposits payments to Espirity's merchant account. When users receive their monthly credit card statement, parking charges are indicated with "Espirity (iParked.ca)" as a merchant. Should a vendor desire credit card statements showing parking vendor specific information of parking charges, the parking vendor must open a merchant account with our gateway. In this case, the parking payment as

well as the transaction fee charged by iParked.ca will be deposited into the parking vendor's merchant account.

Ten minutes before a user's parking expires, a text message is sent to the user's mobile phone notifying the user about the parking expiration. At this time, the user can choose to send another parking request message to extend the parking. If the parking extension is allowed based on the parking rules, the system will make another booking and the remaining time from the original booking will be grandfathered.

For the parking request such as "car p1 90m", the booking is made for the default license plate and province from the user's account. Should the user desire to make a payment for a different license plate, a parking request must also contain the license plate and the province. For example "car p1 90m ddnm101 on" indicates 90 minutes of parking at P1 parking lot at Carleton University for the Ontario license plate DDNM101. Figure 3 illustrates the typical workflow from a user's perspective.

From the parking vendor's perspective the system allows access to our servers through a web interface for running various reports on parking bookings associated with the parking vendor. This ability is available from a link on the Espirity Inc. web site. The server's functionality becomes available once the parking vendor is known to the iParked.ca system. The functionality includes accounting and parking usage reports, available in web or Excel formats. These reports include daily and weekly usage of the system with details associated with the usage, such as a booking's location, user's name and phone number, license plate, start parking time, end parking time, and the parking payment. The administrative functionality also allows for a search of a booking based on a phone number for a given day.

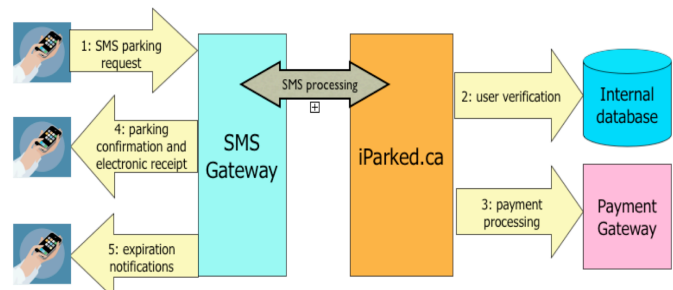


Figure 3. Booking Workflow

The parking vendor has access to the mobile web interface that Parking Enforcement Officers can use for validating parking bookings based on a vehicle's license plate, or by viewing a report of all active bookings for a given parking location. Figure 4 illustrates typical workflow in the system from the Parking Enforcement Officers' perspective.

The iParked.ca solution has the capability to broadcast messages to its users' e-mail accounts. This is used for parking promotions and special events. All users wanting to receive such notifications receive them. The iParked.ca also has the ability to send notifications via Facebook and Twitter to all users following iParked.ca activities on these networks.



Figure 4. Parking Enforcement Officer's Workflow

2.1 Intellectual Achievements

The creation of the iParked.ca service required solutions to several issues. The first related to timing and simplicity. To be accepted, the service needed to be easy to use and responsive. The issue of simplicity relates to how easy it is for users to register and book their parking. Our profile registration and update web site require a minimum amount of information from users. This information includes their phone, address, credit and default license plate information. By registering this information, we minimized the amount of information needed from users when they text message to book a parking. As mentioned earlier, users text a vendor-id, lot/zone and duration to make a parking booking. If they want to park a car other than the one in their profile, they add the license plate and province information to the text message. These two formats are easy to remember, and do not rely on credit information in insecure text messages. To help validate our assumption, we log all messages sent to the service in the wrong format. If too many users were sending illegal messages, it would be quickly apparent that our format was too complicated. We have found that very few users send wrongly formatted messages. Most mistakes relate to spelling, rather than formatting errors. For example, rather than sending "car p2 2h", they might send "cas p2 2h" that is a simple spelling mistake made with their mobile phone's keyboard. We immediately send users of an invalid message an e-mail with a description of valid message formats. We find this attention to detail and responsiveness to users has helped with the overall acceptance of our system.

The issue of responsiveness also meant that we had to process and respond to the user's request as quickly as possible. Our system takes about 2.5 seconds to process a request. The majority of this time is taken by our credit-card gateway processing the transaction. Text messages sent to our short-code are processed by our SMS aggregator, which both receives and sends our SMS message, in less than a second. This means that we can respond to a request in as little as four seconds. The only delay in our system is when carriers have a backlog of SMS messages they need to process. Users can always check their profile to see all their bookings made in the last thirty days, providing up-to-the-second information.

To keep within the four-second timeframe of processing parking requests, the system was developed so that any number of servers could be added to process text messages from one, two, or all vendors. This means that as the load of parking booking requests increases, we simply add more

servers to handle the load. And, if extra capacity exists, a single server can handle the load from multiple vendors. This feature is important for us, as we knew it would be difficult to establish loading data until vendors signed up for the service. However, we wanted to be able to service any vendor, no matter what the load. Through experimentation, we determined our request processing time and designed the service around that number so we could always respond to requests.

Registering and keeping credit information on a corporate server is permitted, but leads to PCI-DSS compliance issues. To store credit card data on one's servers a company must complete a self-assessment questionnaire (SAQ) and have it reviewed. This SAQ requires that servers are locked down almost as tightly as Fort Knox! In our system, we solved this problem by not storing credit card data. We developed the application so that it offloaded the credit information to the responsibility of our credit card gateway, which is PCI DSS certified. The result of this approach is that users' payments for parking are safe and secure, as is their credit information.

The issue of credit information is compounded because we needed to support payment to the merchant accounts of other vendors. Our solution supports multiple vendors and merchant accounts, while still providing secure and safe storage of credit information using our credit card gateway.

The final issue we faced was making our service available at an affordable price. People were already familiar with paying a service fee to text message their votes to enterprises such as Canadian Idol. In an extreme case, we knew that users would pay over \$10.00 for the convenience of booking event tickets using TicketMaster.com. Our belief, which has proven correct, was that users would be willing to pay a minimal amount for being able to pay for parking using their mobile phones. We make a minimal amount on every parking transaction. While we would never have enough transactions to be profitable from a single parking lot, after registering multiple lots, vendors and entire cities, we can be profitable.

In summary, we had many hurdles to overcome to make our service possible and a success. Whatever we did, we knew we could charge a minimal service fee. The service had to be simple and easy to use, and be responsive. The service also had to secure credit information. Through experimentation, we discovered we could provide an affordable service that was very responsive. Through continued thought about the issues and research on possible solutions, we developed a way to safely handle users' credit information. Through the logging of users errors, we have concluded our solution is simple and easy to use, and that the average consumer can use our service with confidence and ease.

2.2 Uniqueness and Originality

While the use of mobile phones to pay for parking using text messages is not unique in the world, it is in North America. iParked.ca is the first system of its kind that provides a text messaging solution for booking and acknowledging parking requests, and sending expiration notifications. The service was first piloted at Carleton University in Ottawa, on July 5, 2010. Since then it has seen continued growth in user registration and usage, as shown in Figures 6.

The service can be broken into four feature areas. The first is user profile registration and updating. The second is the ability for vendors to validate a user's parking and receive daily and weekly reports on revenue collected from parking. The third area is SMS text messaging for booking parking requests. The final area is vendor parking payment calculations.

From the iParked.ca web site, users can both create and update their profiles in English or French. The information collected is for the purpose of managing their profile, contacting them, identifying their mobile phone, their default car's license plate, and their credit information. We have kept this information to the bare minimum so user profile creation and revision is fast and easy. We have found that the most difficult part of the registration process is when users fill in the reCAPTCHA [5] text at the bottom of the page above the Register button. This is required to stop computerized systems from automatically creating thousands of bogus users. In addition, either from their mobile phone by sending the "stop" message, or from their profile by marking the account inactive, users can stop the process of using their mobile phone to pay for parking. This is an important feature if their mobile phone is lost or stolen.

After talking with vendors, it was established that they required a good reporting mechanism to see the money collected from parking bookings and that they required an easy way to validate parking. We provide a web site where vendors can get accounting information and expiration information. In particular, vendors can gather information on expirations for the current day based on lot/zone ids. The report lists active bookings and shows license plate, province, parking location and expiration date and time. The active bookings report looks as follows:

DDNM101, Ont, p1, X:13:51:33, 10-04
 DDNM102, Ont, p1, X:14:51:33, 10-04
 DDNM103, Ont, p1, X:15:51:33, 10-04
 DDNM104, Ont, p1, X:16:51:33, 10-04
 DDNM105, Ont, p1, X:17:51:33, 10-04

Another page lets vendors search by license plate, and two other pages are specific for gathering the same information from small screen mobile devices. Vendors can retrieve parking reports for the current day or week, based on a phone number or just the day or week. One such report is shown in Figure 5, containing only sample names, plates, and phone numbers. The report gives vendors everything they need to verify parking and the amounts charged to users and their mobile phones.

The SMS text messages, as already described, are unique to iParked.ca. They contain the minimum amount of information to park, because we know the default plate and credit information from the profile associated with the user's mobile phone number. If required, users still have the option to park a different car provided they add the plate information

and province abbreviation to the message. We believe nothing could be simpler.

2010-10-06						
Request	Phone Number	License	Lot	Start Time	End Time	Amount
car p1 3h	+16441234	ABC123	p1	2010-10-06 8:10 AM	2010-10-06 11:10 AM	\$9.50
car p2 6h	+11231234	DEF456	p2	2010-10-06 8:17 AM	2010-10-06 12:17 PM	\$12.50
car p2 80m	+12221234	XYZ122	p2	2010-10-06 8:47 AM	2010-10-06 10:07 AM	\$4.50
car p2 1h	+13331234	ABC123	p2	2010-10-06 8:56 AM	2010-10-06 9:56 AM	\$3.50
					Total	\$30.00
					Vendor Total	\$28.00
Total Requests: 4						

Figure 5. Figure 8. Sample Report

The final part that is unique to our system is our parking calculators. Vendors provide us with the three-letter id that they want, such as "car" for Carleton. They provide us with the different lot/zone ids they would like, such as "p1", "z1", or even just "a". The ids can be as long or as short as vendors want, although we encourage short ids to simplify parking request messages. This allows less room for error. Vendors also provide us with the rate information for each lot/zone. We then take this information and create a custom-parking calculator for the vendor that computes parking charges. Changes can be made anytime to the calculator as lots/zone are added, removed, renamed, or are given a new rate.

3 Development

While the development of a service like iParked.ca will never be finished, as new features are added and current ones enhanced, the service is now in full operation at Carleton University, Algonquin College, and running as a pilot at Trent University with discussions underway with other universities, colleges, cities and private parking operators. In the following sections, we describe the timeline that took us from the initial idea to production.

3.1 Milestones

- Discussion about paying for parking using SMS
- Developed plan for a pilot
- Met with SMS message aggregator
- Met with vendor to discuss how parking works on a campus
- Began discussions with Credit Card Payment Gateway

- Signed deal with SMS message aggregator
- Short code approved
- Production short code activated on participating networks
- Merchant account approved
- Domain name iParked.ca secured
- Approached vendor about piloting service
- Vendor agreed to pilot the service
- Applied for iParked.ca trademark
- Pilot started at vendor site
- Pilot extended full service
- Received the iParked.ca trademark.
- Applied for iParked.com trademark.
- Domain name iParked.com secured.

3.2 Discussion

The development of the iParked.ca service took almost two years from idea to pilot. Initially, we found ourselves having difficulty with a pay-and-display machine. It was cold and raining one night and we could not get the machine to take a credit card. We had recently seen in other parts of the world that it was possible to pay for parking using one's mobile phone. This conversation raised two questions: Had anyone produced a parking pay-by-text service in Canada, and if not, why? The answer to the first was no, but we could not find the answer to the second. So we started the development of our service and the searched for an answer as to why it had not been done already.

After coming up with a specification for the functionality of the service, we started to look at the unknown properties of the system that we decided to approach first to decrease risk. At the top of the priority list was how to automate the sending and receiving of SMS text messages. In the early development stage we purchased a wireless 3G device capable of sending and receiving text messages. We developed software for the machine to test its functionality and soon realized that the monthly cost of the carrier connection we needed to pay to support the levels of expected SMS messaging would not enable us to keep the service affordable for end users and vendors. In addition, the amount of new software needed to support this approach seemed too large. Shortly after we gave up on the idea of developing our own software for sending and receiving text messages and decided to find an SMS message aggregator we could use.

Next we researched SMS aggregators, and completed the feature requirements for our service. As we started the development of our system, we defined an application programming interface (API) for how we wanted to communicate with our SMS aggregator. We then met with an SMS aggregator with a solid reputation, to discuss their services and our application.

Satisfied that the SMS aggregator could meet our needs, we continued to develop our system. While we had not signed

a deal with them, we knew how to interface to their system and developed a simulator to send and receive text messages exactly like their real system. This enabled us to develop and test our system, without having to commit to the early expense of signing a contract with the aggregator.

Later we met with a university vendor to find out how parking worked at the university. We knew that we could not charge users much for the service but wanted to find out what it cost a parking vendor to support their current parking service. This meeting revealed that the main expenses were the cost of servicing their pay-and-display machines and credit card transaction/transmission fees. The meeting also provided us with the expected number of parking events per month. From this information, we realized that our service needed to be inexpensive from a vendor perspective, staying well below of the cost of buying or renting pay-and-display machines and their corresponding service.

Next we began discussions with our credit card payment gateway that offered everything we needed to accept credit cards and make electronic payments. Like the SMS aggregator, we did not sign a deal immediately. However, we did gain access to the gateway's APIs so we could test our system with their payment gateway in simulation mode.

Finally, we had the software developed, tested and running in simulation mode. It was time to get real! After further discussions with the aggregator and gateway, we determined that we could use their services and provide affordable service. Not much less, but given the expected high volume of parking events we wanted to support, we could be profitable. We signed a deal with the aggregator to provide us with the capability to send and receive text messages. Then, with the help of our aggregator, the short code 727533, which spells PARKED, was officially approved. Next we signed an agreement with our payment gateway and received a corresponding merchant account to perform online Visa and MasterCard transactions. We secured the domain name iParked.ca and developed the web site for user profile registration and updating. We applied to trademark the name iParked.ca that was finally granted.

We approached Carleton University with the idea of piloting the new service. From these initial discussions, everyone at Carleton liked the service and on June 16 2010, it was announced that Carleton would pilot the service in lot P2. The pilot was a success and was extended to lot P1. During the pilot, minor changes were made to make it easier for the parking officers to use their hand-held devices to validate parking. Since the beginning of the pilot, we have seen week-by-week growth in user registration and weekly usage.

Looking beyond the Canadian market, we secured the domain name iParked.com and applied for and were granted the corresponding trademark.

4 Commercialization

The first pilot for the iParked.ca service was announced at Carleton University on June 16, 2010 and went live on July 5, 2010. The graph in Figures 6 shows the linear growth of registered users of the service since its start. The graph show continued increase as more people become aware of the

service, register and use it. As the weather is cold or wet we see greater usage.

We have been in contact with other universities and colleges. As expected each vendor has different requirements for parking fees. Some fees are calculated by the hour, with a maximum four-hour time limit. Others have a flat fee paid after a certain time such as 5:00 p.m, while other fees include a single flat fee for the full day. In either case, our parking calculators are customizable for each vendor's situation. The way parking enforcement officers operate at different institutions is also different. Some parking enforcement officers have wireless handheld devices that require a wireless connection. Other enforcement officers have smart phones. In either case, our web interface provides the data for enforcement officers to validate parking. Since our interface for the officers is web based, all that is actually required is a device connected to the web.

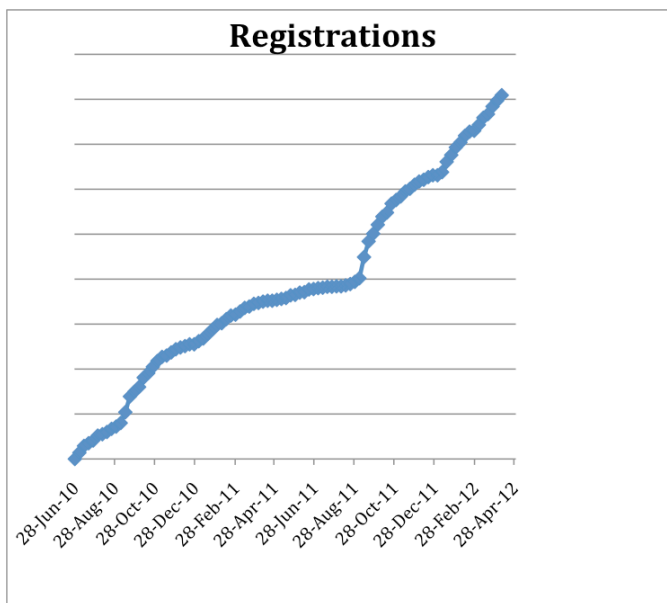


Figure 6. Total Registration Growth

Beyond approaching cities, colleges and universities directly, we are looking to partner with Pay & Display vendors. We see pay-by-text as a natural extension to the pay-and-display machines, the difference being that you do not need to expose credit cards or money, or even be at a machine to make your parking booking. Pay & Display vendors can use our service to help complement theirs. And as new technologies are developed, such as Gtechna's automated License [6] Plate Recognition technology, and are merged with ours and the pay & display machines, parking and its enforcement become easier for all parties involved.

5 Impact

The greatest impact of the iParked.ca service is that it provides people a safe, simple, easy to use method for paying for parking. At pay-and-display machines, a user must either expose their money or their credit card and as a result becomes easy prey for thieves, when out alone or at night. As one can imagine, a user's attention is usually fully focused on the

machine and its interface rather than on their surroundings. When it is raining, snowing, cold, or all three, it is an unpleasant experience to get a parking receipt from a machine. Having to walk to the pay-and-display machine and return to your car compounds the negative experience. With the iParked.ca service, there is no need to expose money or your credit card. You do not even have to get out of your locked, warm, dry car. You are as safe as you normally are in your car. And the parking request is completed in less than five seconds. These features are particularly important for those with disabilities.

In addition, when your parking expires, and you are using the pay-and-display machines, you have to get another ticket. With the iParked.ca service, you send another text message booking request, taking again less the five seconds, from wherever you are. The iParked.ca service saves you the time that you would waste going back and forth to the machines. The iParked.ca service also saves users the cost and inconvenience of having to either pay or fight parking tickets, a benefit to anyone who has ever received a ticket. The result is that the service saves people money. Since users can make a booking from anywhere that has mobile phone reception, we provide a simple, easy and fast way to renew a booking. A user will no longer have to play the game of guessing if they are going to be ticketed because they cannot get back to their car for another ten minutes.

The second impact of the iParked.ca service is that it will provide jobs. As the service grows, Espirity Inc. will also grow to support the demand for the service. Given it is currently a very small company with only three full and three part-time contract employees, it will need to hire office staff and developers. It will need to rent larger office space. In short, it will need to grow to meet the service demand. The iParked.ca service is facilitating the birth of a service-oriented company that is looking to expand across Canada and throughout North America.

iParked.ca have has media coverage from across Canada and the United States. Listed below are representative links of articles. In addition, the service has been featured on CBC and Live 88.5 Radio in Ottawa, and CHML Radio in Hamilton. The following list includes samples of related news stories:

- <http://www.ottawasun.com/news/ottawa/2010/06/15/14395616.html>
- <http://www.ottawacitizen.com/technology/Parking+campus+snap+with+professor/3161499/story.html>
- <http://www.obj.ca/Technology/2010-06-15/article-1294871/Carleton-professor-develops-parking-payments-by-text/1>
- <http://video.vancouver.24hrs.ca/archive/carleton-unveils-text-message-payment-parking/96372747001>
- <http://cacm.acm.org/news/94408-parking-on-campus-a-snap-with-carleton-professors-app/fulltext>
- <http://oncampus.macleans.ca/education/tag/parking-tickets/>

The response to this service has been overwhelmingly positive. We are asked constantly why it took so long to

come to Canada? We are just happy we are able to make it available now. We have asked ourselves the same question over two years ago and the rest is, as they say, history!

6 References

- [1] iParked.ca. (2012). Retrieved April 9, 2012, from <http://iparked.ca>
- [2] Canadian Wireless Telecommunication Association. (2012). Retrieved April 9, 2012, from <http://www.cwta.ca/>
- [3] PCI Security Standards Council. (2012). Retrieved April 9, 2012, from <https://www.pcisecuritystandards.org/>
- [4] Espirity Inc. (2012). Retrieved April 9, 2012, from <http://http://espirity.com>
- [5] reCAPTCHA. (2012). Retrieved April 9, 2012, from <http://www.google.com/recaptcha>
- [6] GTechna. (2012). Retrieved April 9, 2012, from <http://www.gtechna.com/>

SESSION

SOFTWARE QUALITY ASSESSMENT + OPTIMIZATION METHODS + REDUNDANCY REMOVAL + RELIABILITY ISSUES

Chair(s)

TBA

Fuzzy Measure Extraction for Software Quality Assessment as a Multi-Criteria Decision-Making Problem

Xiaojing Wang*, Martine Ceberio†, Shamsnaz Virani‡, Christian Del Hoyo§, Luis Gutierrez§
Computer Science Department

The University of Texas at El Paso, El Paso, Texas 79968-0518

* xwang@utep.edu, † mceberio@utep.edu, § cdelhoyo@miners.utep.edu, ¶ lcgutierrez@miners.utep.edu

‡ Engineering Division, Penn State Great Valley

School of Graduate Professional Studies, Malvern, PA 19355

Email: ssv1@psu.edu

Abstract—Being able to assess software quality is essential as software is ubiquitous in every aspect of our day-to-day lives. In this paper, we rely on existing research and metrics for defining software quality and propose a way to automatically assess software quality based on these metrics. In particular, we show that the software quality assessment problem can be viewed as a multi-criteria decision-making (MCDM) problem. In Multi-Criteria Decision Making (MCDM), decisions are based on several criteria that are usually conflicting and non-homogenously satisfied. Non-additive (fuzzy) measures along with the Choquet integral can be used: they model and aggregate the levels of satisfaction of these criteria by considering their relationships. However, in practice, it is difficult to identify such fuzzy measures. An automated process is necessary and can be used when sample data is available. We propose to automatically assess software by modeling experts' decision process: to do this we automatically extract the corresponding fuzzy measure from samples of the target experts' decision. We were able to improve previous approaches to automatic software quality assessment that used machine learning techniques.

I. INTRODUCTION

Pfleeger summarizes software quality assessment in three ways [16]:

- 1) The quality of product;
- 2) The quality of process; and
- 3) The quality in the context of the business environment.

This paper focuses on software product quality assessment based on direct measurements of code properties. Software product quality is important because software is present in every aspect of normal day-to-day life. Software problems such as server breakdowns, software crashes, and data leaks have become common occurrences. Pre-existing software problems do not stop software spending. Even though there are large amounts of money spent on developing software, the quality of this software still remains a mystery. The obvious questions therefore are: what is software quality and how is it measured?

In this paper, we rely on existing research and metrics for defining software quality, as presented in Section II. Once metrics adopted, it is relatively easy to decide, for each metrics, which piece of software is better than the others based on the measure / satisfaction level related to the corresponding metrics. However, even so, the problem of assessing software quality based on several metrics remains since it constitutes a complex decision involving several criteria (based on metrics such as length of code, number of classes, inter-dependence of classes).

Experts are used to assess software quality, but it is desired that this process could be automated so as to ensure the consistency of the assessments as well as its timeliness (e.g., on-the-fly assessment). Such a task has been tackled previously by Fuentes et al. [15] using a machine learning approach. In this article, we show that the software quality assessment problem can be viewed as a multi-criteria decision-making (MCDM) problem, and we propose to extract experts' decision process using Fuzzy Measure Extraction for MCDM.

In what follows, we start by reviewing the state of the art in Software Quality and Software Quality Assessment (Section II). We then provide information about theoretical topics that are central to our MCDM approach (Section III) and we show how SQA and MCDM are related, with details about our specific approach (Section IV). We tested our approach: our strategy along with our experimental results and analysis are provided in Section V. Finally, we conclude and draw directions for future work in Section VI.

II. SOFTWARE QUALITY ASSESSMENT (SQA): DEFINITIONS, CURRENT STATUS, AND EXISTING TECHNIQUES

A. SQA: definitions and current status

Pressman defined software quality as “Conformance to explicitly stated functional and performance requirements, explicitly documented development standards and implicit

characteristics that are expected of all professionally developed software” [19].

This definition addresses two aspects of software quality. The first is conformance to explicitly stated functional and performance requirements and explicitly documented development standards. These can be found in the requirements document developed between the customer and client. This requirement can be measured by counting the number of columns and comparing the data type to that stated in the requirements data. The performance requirements are also measurable. For example, the time a software product takes to complete a given task is a measure of performance. Functional and performance requirements and explicitly documented development standards are thus measurable.

The second aspect of software quality addressed in this definition is the implicit characteristics of all professionally developed software. These are characteristics such as reusability or flexibility. These implicit characteristics are also known as quality factors. Most software engineers, programmers and managers believe that software quality factors are best judged by experts. However, research has shown that expert judgments are often inconsistent and subjective. For example, experts such as Lorenz and Kidd considered multiple inheritances (software property) as a sign of bad quality code, but multiple inheritance is widely accepted in the programming community ([10]). Inconsistent expert opinion makes judgment of implicit characteristics such as reusability difficult. There is no quantitative measurement for quality factors such as reusability. This subjective aspect of software quality results in making software quality management difficult.

Solid information regarding the quality of the software product is difficult to estimate. There is currently no tool or process that uses quantitative information to calculate quality factors for software products. This lack of solid information creates problems with software project management.

Sommerville elaborates that software quality management is difficult because of the two different aspects of software quality [20]. The explicit aspect of software quality is to some extent measurable, but the implicit aspect of software quality is solely based on expert opinion which is frequently inconsistent. There is a need to directly measure the implicit aspect of software quality.

To understand the subjective estimates and the measurements, it is important to understand the software quality research.

Theoretical models define several quality factors such as reusability and flexibility but do not quantify them. Predictive models are models based on statistical techniques that predict characteristics such as fault density or fault proneness using direct measurements from code (product metrics). Predictive models predict the faults but have no theoretical evidence to support causality. One solution to remedy this problem is to add prediction capability to a

theoretical model. Quality factors defined in a theoretical model are not measurable and hence cannot be predicted. One theoretical software quality model that defined the quality factors and linked all of them to measurable metrics in object-oriented software was Bansiya and Davis’s model [2]. Bansiya and Davis’s model is more complete than other theoretical software quality models, so it was chosen for analysis here. Bansiya and Davis’s model defines the quality factors and links them to QMOOD set of metrics defined in Table I.

Although the Bansiya and Davis model provides a solid explanation for the design quality of object-oriented design, it presents some limitations. The major problems with the Bansiya and Davis model are their validation process, data used in the research and lack of prediction capability.

B. SQA: Existing Techniques

Machine learning is an important aspect in predicting software product quality because the more a classifier can learn, the better decisions it will make in building a predictive model [12]. Osbeck et al improved the prediction capability using J48, Part, and Random Forest, and the ensemble learning techniques examined were boosting, bagging, and stacking [15].

In our work, we show that SQA can be seen as a Multi-Criteria Decision-Making Problem problem and solved accordingly, and that, therefore, software product quality can be predicted by using fuzzy measures and Choquet integral.

III. THEORETICAL BACKGROUND

In this section, we introduce important theoretical concepts necessary to later understand how SQA can be viewed as a Multi-Criteria Decision Making (MCDM) problem as well as why Fuzzy Measure Extraction (FME) needs to be carried out.

A. Multicriteria Decision Making (MCDM)

Multicriteria decision making (MCDM) is the making of decisions based on multiple attributes (or criteria). Usually, it consists of a set of consequences, a finite set of n criteria (or attributes), and a preference relation \succeq on the set of consequences.

The set of consequences X is a multidimensional space, where $X \subseteq X_1 \times \dots \times X_n$, and each X_i represents a set of values of attribute i , where $i \in \{1, \dots, n\}$. For each criterion (or attribute), there is a preference relation \succeq_i on each space X_i , such that for $x_i, y_i \in X_i$, $x_i \succeq_i y_i$ means that x_i is preferred to y_i . Then, the preference relation of a consequence for all criteria can be combined, using an aggregation operator, into a global value such that the final level of satisfaction of the consequences follows the partial preferences. A preference over the set of consequences X will be denoted as: $\forall x, y \in X, x \succeq y$ or $y \succeq x$.

TABLE I
QMOOD MODEL [2]

Quality Factor Definition	Bansiya and Davis's Model	Metric Definition (QMOOD metric)
Reusability Reflects the presence of object oriented design characteristics that allow a design to be reapplied to a new problem without significant effort.	$-0.25 * \text{Coupling} + 0.25 * \text{Cohesion} + 0.5 * \text{Messaging} + 0.5 * \text{Design Size}$	Design Size (DSC) A measure of number of classes used in the design. Hierarchies (NOH) Hierarchies are used to represent different generalization-specialization aspects of the design. Abstraction (ANA) A measure of generalization-specialization aspect of design. Encapsulation (DAM) Defined as the enclosing of data and behavior within a single construct. Coupling (DCC) Defines the inter dependency of an object on other objects in a design. Cohesion (CAM) Accesses the relatedness of methods and attributes in a class. Composition (MOA) Measures the "part-of", "has", "consists-of", or "part-whole" relationships, which are aggregation relationships in object oriented design. Inheritance (MFA) A measure of the "is-a" relationship between classes. Polymorphism (NOP) It is a measure of services that are dynamically determined at run-time in an object. Messaging (CIS) A count of number of public methods those are available as services to other classes. Complexity (NOM) A measure of the degree of difficulty in understanding and comprehending the internal and external structure of classes and their relationships.
Flexibility Characteristics that allow the incorporation of changes in a design. The ability of a design to be adapted to provide functionality related capabilities.	$0.25 * \text{Encapsulation} - 0.25 * \text{Coupling} + 0.5 * \text{Composition} + 0.5 * \text{Polymorphism}$	
Understandability The properties of designs that enable it to be easily learned and comprehended. This directly relates to the complexity of design structure.	$-0.33 * \text{Abstraction} + 0.33 * \text{Encapsulation} - 0.33 * \text{Coupling} + 0.33 * \text{Cohesion} - 0.33 * \text{Polymorphism} - 0.33 * \text{Complexity} - 0.33 * \text{Design Size}$	
Functionality The responsibility assigned to the classes of a design, which are made available by classes through their public interfaces.	$0.12 * \text{Cohesion} + 0.22 * \text{Polymorphism} + 0.22 * \text{Messaging} + 0.22 * \text{Design Size} + 0.22 * \text{Hierarchies}$	
Extendibility Refers to their presence and usage of properties in an existing design that allow for the incorporation of new requirements in the design.	$0.5 * \text{Abstraction} - 0.5 * \text{Coupling} + 0.5 * \text{Inheritance} + 0.5 * \text{Polymorphism}$	
Effectiveness This refers to the designs ability to achieve the desired functionality and behavior using object oriented design concepts and techniques.	$0.2 * \text{Abstraction} + 0.2 * \text{Encapsulation} + 0.2 * \text{Composition} + 0.2 * \text{Inheritance} + 0.2 * \text{Polymorphism}$	

The common aggregation operator being used is a weighted sum; i.e.,

$$u(x) = \sum_{i=1}^n w_i u_i(x_i),$$

where w_i is the weight of each criterion, representing the importance of each criterion, $\sum_{i=1}^n w_i = 1$, and $u_i(x_i)$ represents the level of satisfaction assigned to alternative $x_i \in X_i$. The best consequence ($x \in X$) is the one with the highest value of u . Although this approach is simple, easy to use, and low complexity, using an additive aggregation operator assumes that all criteria are independent, which, in practice, is seldom the case: often, decisions are based on several conflicting criteria and using linear additive aggregation will lead to possibly very counterintuitive decisions. Non-linear approaches also prove to lead to solutions that are not completely relevant. Therefore, using additive approach is often not good: based on our previous work [13], we choose to use non-additive approaches, i.e., fuzzy measures and integrals [3].

B. Fuzzy measures and integrals

Fuzzy measures are non-additive measures. They can be used to represent the degree of interaction of each subset of criteria [4]. In what follows, we consider a finite set of criteria $A = \{1, \dots, n\}$.

Definition Let A be a finite set and $\mathcal{P}(A)$ the power set of A . A fuzzy measure (or a non-additive measure) defined on A is a set function $\mu : \mathcal{P} \rightarrow [0, 1]$ satisfying the following axioms:

- (1) $\mu(\emptyset) = 0$
- (2) $\mu(A) = 1$
- (3) if $X, Y \subseteq A$ and $X \subseteq Y$, then $\mu(X) \leq \mu(Y)$

The fuzzy measures are used to show the importance of each subset and how each subset of criteria interacts with others. Fuzzy measures are expensive to determine: for a set defined over n criteria, 2^n values of a fuzzy measure are needed because there are 2^n subsets of A .

Two main integrals can be used to combine fuzzy measures: the Sugeno and the Choquet integrals. Although they are structurally similar, they are different in nature [8]: the Sugeno integral is based on non-linear operators and the Choquet integral is usually based on

linear operators. The applications of Sugeno and Choquet integrals are also very different [14]: the Choquet integral is generally used in quantitative measurements, and a MCDM problem usually uses a Choquet integral as a representation function. In this article, we focus on the Choquet integral.

Definition Let μ be a fuzzy measure on A . The Choquet integral of a function $f : A \rightarrow R$ with respect to μ is defined by:

$$(C) \int_A f d\mu = \sum_{i=1}^n (f(\sigma(i)) - f(\sigma(i-1)))\mu(A_{(i)})$$

where σ is a permutation of the indices in order to have $f(\sigma(1)) \leq \dots \leq f(\sigma(n))$, $A_{(i)} = \{\sigma(i), \dots, \sigma(n)\}$ and $f(\sigma(0)) = 0$, by convention.

C. Determining Fuzzy Measures

In MCDM, we would expect the decision maker to be more than likely to give the values of the fuzzy measure, but in most circumstances this is not the case. Attempts of making fuzzy measure identification easier for the decision makers have been made in [3], [22].

- In [3], the authors attempt to make this task easier by only requiring the decision maker to give an interval of importance for each interaction.
- In [22], the author suggests a diamond pair-wise comparison, where the decision maker only must identify the interaction of 2 criteria using a labeled diamond. From there, the algorithm evaluates the values of the numeric weights.
- In [22], the author discusses user specified weights mixed with an interaction index denoted λ or ξ . This algorithm is applied using an online aggregation application [21].

However, in most cases, the decision maker either does not understand the interactions well enough to provide a good value of each fuzzy measure, or does not have constant access to an expert who may give all values of the fuzzy measures. In addition, since there are $2^n - 2$ values of a fuzzy measure for a problem with n criteria expert identification: it would be too time consuming anyway to be practical [7]. This is where fuzzy measure extraction comes into play.

D. Fuzzy Measure Extraction (FME) and Optimization

For lack of an expert to provide all values of the fuzzy measure, we need seed data to give us an idea of the preferences / decision-making process: we use sample data. Extracting fuzzy measure is performed starting from such seed data.

Let's take a look at the following situation: Our MCDM problem involves n attributes, and m sample data. If we knew the fuzzy measure $\tilde{\mu}$, we would be able to compute preference values \tilde{y}_j as $(C) \int_A f d\tilde{\mu} = \sum_{i=1}^n (f(\sigma(i)) - f(\sigma(i-1)))\tilde{\mu}(A_{(i)})$, where f is a utility function defined on X .

However, with the sample data, we only have access to the preference values of a subset of X . In order to have access to preference values of other alternatives in X , we need to determine μ , which is, the $2^n - 2$ values of the fuzzy measure. We are going to determine μ such that the corresponding computed Choquet integral is as close to the preference values of the sample data as possible. As a result, we aim at minimizing the following sum (and getting as close to 0 as possible) [9]:

$$e = \sum_{j=0}^m \left(\tilde{y}_j - \sum_{i=1}^n (f(\sigma(i,j)) - f(\sigma(i-1,j)))\mu(A_{(i)}) \right)^2 \quad (1)$$

Moreover, the optimal solution must satisfy constraints: fuzzy measures must be monotonic and must always be between 0 and 1.

Therefore, fuzzy measure extraction is a constrained optimization problem, and the candidate solutions must be evaluated to make sure they fit the constraints.

Several optimization approaches have been proposed to extract fuzzy measures, including Gradient Descent methods [1], Genetic Algorithms [4], [27], and Neural Networks [24]. Besides these, many optimization techniques exist, such as for instance Harmony Search [6], Particle Swarm Optimization [18], Simulated Annealing [5].

However, the main drawbacks in these techniques are that the returned solution (found minimum of the objective function) might just be a local minimum, or even worse, a good value. There is no guarantee that it would be the global minimum at all.

In previous work [26], we proposed to use a tuned version of the Bees Algorithm [17] to extract fuzzy measures. The results show promise of the approach, and although the results were not guaranteed to be global (same drawback as pointed out of the other approaches), they were consistently better than best approaches before this one. In this work on SQA, we used the Bees Algorithm to extract fuzzy measures that best model experts' decision processes.

In what follows, we first motivate why MCDM is related to SQA and how Fuzzy Measure Extraction can help automate SQA. We then provide details about the specific optimization technique we use for Fuzzy Measure Extraction.

IV. FUZZY MEASURES EXTRACTION FOR SOFTWARE QUALITY ASSESSMENT USING THE BEES ALGORITHM

As hinted earlier, Software Quality Assessment can be seen as a Multi-Criteria Decision-Making problem. The straightforward reason for that is the following: the general quality assessment (i.e., final decision of software) is based on a set of metrics (i.e., multiple criteria).

As a result, based on what we presented about fuzzy measures and Choquet integrals, if we can find an appropriate fuzzy measure μ , assessing the quality of software

can be expressed as follows:

$$\text{SQA}(\text{software } A) = \text{Choquet}(\mu, \text{metrics values}(A))$$

Based on the above formula, we focus on determining μ , which can be viewed as a quantitative model for the expert's decision-making process. It is obtained from seed data, namely sample data of experts' decisions with respect to known pieces of software. As a result, in the above 3-body problem, two of the components are known: the expert's decision and the set of metrics values, and we aim at determining the matching μ .

In practice, we have access to a number of such expert(s)' decision values along with the corresponding sets of metrics values for different pieces of software A_i . As pointed out earlier, in Section III, determining μ consists of solving the following problem:

$$\begin{array}{l} \min e = \sum_{A_i} (\text{SQA}(A_i) - \text{Choquet}(\mu, \text{metrics}(A_i)))^2 \\ \text{s.t. } \mu \text{ satisfies monotonicity constraints} \end{array}$$

A. Our approach: the Bees Algorithm

The Bees optimization Algorithm, proposed in [17], uses bees' natural food foraging habits as a model for the exploration of the search space. The Bees Algorithm combines a local and "global" search that are both based on bees natural foraging habits. It roughly unwinds as follows:

- 1) First a number of "scout bees" are randomly sent out.
- 2) The patches of "nectar" (elements of the search space / candidate values for the fuzzy measure) are then ranked according to evaluated fitness. More bees are dispatched to look in neighboring areas of good patches of "nectar".
- 3) At each iteration, a number of "scout bees" are kept to explore other areas in hope of better patches of "nectar": this keeps the algorithm searching "globally".
- 4) When a new patch is found, its fitness is evaluated and compared against previously explored patches and a proportional number of "bees" is sent to it. The dispatched "bees" perform a local search by moving in a random direction from the patch of "nectar".
- 5) If a local search "bee" finds a better patch of "nectar", the location from where it was dispatched is moved to the new location [17]. The Bees Algorithm performs local search by sending an amount of "bees" that is proportional to the patch's fitness.

It is believed that the best ranked "patch", when a stopping criterion is met, is the optimal solution, although there is no theoretical guarantee for it. In [17], the Bees algorithm, in most cases, was faster than a number of other algorithms (including genetic algorithms, simplex method, stochastic simulated annealing), and returned a solution within .1% of the perfect solution every time run

(in particular up to 207 times faster than the Genetic Algorithm on the benchmarks used). On their test cases, the Bees algorithm was also always able to reach the global optimum and ignore local optima.

V. EXPERIMENTS

A. What do we want to show and how?

Through our experiments, we aim at quantifying the performance of our approach in automatically predict software quality. As hinted before, the algorithm we use to extract fuzzy measures is the Bees algorithm, as it showed promise in previous work [26].

The metrics we use are Quality Model for Object-Oriented Design (QMOOD) metrics, as defined in [2] and used by Virani et al [23]. In this work, we focused on the metrics and the quality factors related to QMOOD model, as in [2]. Definitions for each of the quality factors and metrics used in QMOOD model are provided in Table I.

The data at our disposal to run our tests is from 31 software packages and 2330 samples. The rating options are bad, poor, good, fair, and excellent. Each package was rated individually by a group of experts for each of the metrics and for the total quality. Note that the data is the same data set as used in [15], in which the authors proposed a machine learning techniques to predict SQA.

Using the above-described data, our goal is to show that: (1) our approach allows to accurately recreate decisions (data) used to determine the reasoning process (fuzzy measure); and that (2) it also works to predict decisions (data) that were not included in determining the reasoning process (fuzzy measure) but that were available to us.

We were also interested in addressing the following questions:

- 1) Are experts consistent when they assess software? Focusing on one expert at a time, we wanted to know how well we would be able to reconstruct each expert's decision-making process. Again, this was highly dependent on each expert's original ability to make consistent decisions.
- 2) Do experts agree on software assessment? or are we able to create a "super expert"'s decision process using our approach? Looking at one software package at a time, across experts, we wanted to get a sense of common decision-making process among the experts. We wanted to see how closely we would be able to reconstruct the decisions, which is very tightly coupled with a sense of agreement across the experts.

B. Experimental Results

1) *Reconstructing the data: Using all data and all experts.* The 2330 samples we have at our disposal include 31 software packages and are evaluated by 78 experts. We first extracted a fuzzy measure from all samples that fits them all the best. In Table II, we report the accuracy reached when determining the target fuzzy measure; i.e.,

the sum of the differences between the reconstructed data and the original data for all the data. In particular, we provide this information for different iteration counts of our Bees algorithm: we can observe that the quality is stabilized already after 100 iterations.

TABLE II
OPTIMAL FUZZY MEASURE FOR ALL SAMPLES

Iterations	e
100	0.07675
1000	0.07671
10000	0.07670

Although the results are stable after 100 iterations, it is about 10 times worse than the results of the toy examples in [26]. This is because 78 experts were involved the SQA evaluation and the decisions are not consistent among different experts. Even for the same expert, the decisions may not be consistent. This is the object of our next experiments.

Focusing on one expert at a time. We ran experiments to see whether experts were consistent in their decision process; that is, for different software packages, whether the values of the metrics are similar, then each expert's decision of the quality should be very close. Table III shows the results for each expert.

TABLE III
TESTING RESULTS FOR EACH EXPERT

e	# of experts
< 0.01	25
[0.01, 0.035)	25
[0.035, 0.075)	16
≥ 0.075	12
min	1.68E-05
max	0.4398

We find that most individual expert's decision processes (64%) are consistent, while some experts' decision shows big discrepancies from one sample or package assessment to another.

For example, one expert evaluated 5 samples for 1 software package, and the result is 0.4398. We check the sample data and find no matter how difference the value of the metrics are, the expert always makes the same decision, which means the expert's opinion on the SQA is not consistent.

Focusing on one software package at a time. We tested to see whether the evaluation from different experts for the same software package is consistent, that is, for the same software package, since evaluated by different experts, the discrepancy of the experts' decision should be larger than the previous results. Figure 1 shows the results for each software package.

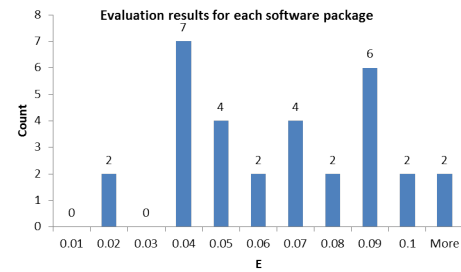


Fig. 1. Evaluation Results for each Software package

Since each software package was evaluated by at least 6 experts, and each expert has a different point of view for each Software package, it is reasonable that the results for each Software package is worse (less consistent) than the results for each expert.

2) *Predicting the data:* As mentioned earlier, the goal of this research is to test how well fuzzy measure extraction can help predict software quality. We test it by first extracting fuzzy measure from part of the sample data (randomly selected) and then using the extracted fuzzy measure to predict the software quality for the rest samples to see if the predicted software quality matches the original software quality. Testing results are in table IV.

TABLE IV
TESTING RESULTS FOR RANDOMLY SELECTED PARTIAL SAMPLES
(ITERATION = 1000)

Sample numbers	e
500	0.07904
1000	0.07862
2330	0.07671

C. Analysis of our results

What we have shown in above is how well the Bees algorithm used to extract fuzzy measure. Now we want to test how well the extracted fuzzy measure can help predict Software quality.

We compared the evaluations we obtained to the original experts' decision and assessed the accuracy of our approach using 3 different evaluation processes.

- 1) Eval1: We computed the average assessment over the known experts' decisions and considered this average the target evaluation. Anything else would just be considered wrong.
- 2) Eval2: We used the same average as before but allowed for some flexibility by accepting any evaluation within σ (standard deviation) of the target average. This evaluation accounted for the uncertainty of the experts' decisions.
- 3) Eval3: We mapped the number of experts decisions from 0% to 100% based on the maximum number of votes for one rating (which would be the one to

receive 100% accuracy) and interpolated all other possible ratings (numerical values in between posted ratings) to determine their accuracy.

The overall evaluation results for all quality factors are in table V, and we also list the results using machine learning approach to compare with.

TABLE V
ACCURACY USING HYBRID2

Quality Factor	Machine learning approach [15]	Eval 1 (Average)	Eval 2 ($\mu \pm \sigma$)	Eval 3
Reusability	69.91%	41.67%	79.55%	72.34%
Flexibility	75.39%	47.89%	75.71%	66.34%
Extendibility	70.37%	42.80%	77.43%	70.91%
Functionality	78.54%	27.39%	42.74%	59.50%

VI. CONCLUSION AND FUTURE WORK

In this article, we proposed an multi-criteria decision-making-based approach to software quality assessment. By viewing this assessment problem as a multi-criteria decision making (MCDM) problem, we were able to use fuzzy measures to model software expert's decision making process and help predict/evaluate software quality. We were able to show that our approach (specifically fuzzy measure extraction based on experts' decisions data) helps to predict/evaluate software quality with consistently over 60% accuracy, which is as accurate as previous approaches to SQA conducted using machine learning techniques.

Our current approach can be further improved as follows. Although the Bees algorithm we implemented provides good and reasonably fast results for fuzzy measure extraction, we believe we can improve it by combining it with another solver. Moreover, the expert's opinions (data used to extract a decision process model) usually are linguistic values; for example, Excellent, Good, Fair, Poor, and Bad. These words have different meanings to different experts, and therefore, expert's linguistic ratings are uncertain and their interpretation should not be uniform. In particular, using a continuous utility function that assigns a precise value to each of these evaluation results would result in losing accuracy. In order to better fit the expert's opinions, in the future, we will use an interval end-points approach [11] and may use non-linear utility functions. Finally, we need to study the amount of data that is necessary to extract meaningful and accurate decision process models: for instance, what is the impact of a reduced sample data set on the quality of the decisions? What is the critical number of data w.r.t. the number of criteria for instance?

REFERENCES

- [1] S. H. Alavi, J. Jassbi, P. J. A. Serra, and R. A. Ribeiro. Defining fuzzy measures: A comparative study with genetic and gradient descent algorithms. In *Intelligent Engineering Systems and Computational Cybernetics*, pages 427–437. Springer Netherlands, 2009.
- [2] J. Bansiya and C. Davis. A hierarchical model for object-oriented design quality assessment. *IEEE Transactions on Software Engineering*.
- [3] M. Ceberio and F. Modave. An interval-valued, 2-additive choquet integral for multi-criteria decision making. In *Proceedings of the 10th Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems (IPMU'04)*, Perugia, Italy, July 2004.
- [4] E. F. Combarro and P. Miranda. Identification of fuzzy measures from sample data with genetic algorithms. *Computers & Operations Research*, 33(10):3046–3066, 2006.
- [5] L. Davis. *Genetic Algorithms and Simulated Annealing*. Morgan Kaufmann Publishers Inc., San Francisco, CA, 1987.
- [6] Z. W. Geem, J. H. Kim, and G. V. Loganathan. A new heuristic optimization algorithm: harmony search. *Simulation*, 76(2):60–68, 2001.
- [7] M. Grabisch. A new algorithm for identifying fuzzy measures and its application to pattern recognition. In *Proceedings of 4th IEEE International Conference on Fuzzy Systems*, Yokohama, Japan, March 1995.
- [8] M. Grabisch. The application of fuzzy integrals in multicriteria decision making. *European Journal Of Operational Research*, 89(3):445–456, 1996.
- [9] M. Grabisch, H.T. Nguyen, and E. A. Walker. *Fundamentals of uncertainty calculi with applications to fuzzy inference*. Kluwer Academic Publishers, Norwell, MA, 1994.
- [10] M. Lorenz and J. Kidd. *Object-oriented software metrics: a practical guide*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1994.
- [11] J. Mendel. Computing with words and its relationships with fuzzistics. *Information Sciences*, 177:988–1006, 2007.
- [12] T. M. Mitchell. *Machine Learning*. McGraw-Hill, New York, first edition, 1997.
- [13] F. Modave, M. Ceberio, and V. Kreinovich. Choquet integrals and owa criteria as a natural (and optimal) next step after linear aggregation: A new general justification. In *Proceedings of MICAI'2008*, pages 741–753, 2008.
- [14] F. Modave and P. W. Eklund. A measurement theory perspective for mcdm. In *Proceedings of the 10th IEEE International Conference on Fuzzy Systems*, pages 1068–1071, Melbourne, Australia, 2001.
- [15] J. Osbeck, S. Virani, O. Fuentes, and P. Roden. Investigation of automatic prediction of software quality. In *North American Fuzzy Information Processing Society (NAFIPS'2011)*, El Paso, TX, March 2011.
- [16] Pfleger. *Software Engineering Theory and Practice*. Prentice Hall, 2001.
- [17] D. Pham, A. Ghanbarzadeha, E. Koc, S.Otri, S. Rahim, and M. Zaidi. The bees algorithm—a novel tool for complex optimization problems. In *Proceedings of 2nd International Virtual Conference on Intelligent Production Machines and Systems (IPROMS 2006)*, pages 454–459, 2006.
- [18] R. Poli, J. Kennedy, and T. Blackwell. Particle swarm optimization. *Swarm Intelligence*, 1(1):33–57, 2007.
- [19] R. Pressman. *Software Engineering: A Practitioner's Approach*. McGraw-Hill, 2005.
- [20] I. Sommerville. *Software Engineering*. Addison Wesley Publishing Company, Harlow, England, 2004.
- [21] E. Takahagi. Usage: Fuzzy measure-choquet integral calculation system (λ fuzzy measure and sensitivity analysis). <http://www.isc.senshu-u.ac.jp/~thc0456/Efuzzyweb/mant2/mant2.html>.
- [22] E. Takahagi. A fuzzy measure identification method by diamond pairwise comparisons and ϕ_s transformation. *Fuzzy Optimization and Decision Making*, 7(3):219–232, 2008.
- [23] S. S. Virani, S. Messimer, P. Roden, and L. Etzkorn. Software quality management tool for engineering managers. In *Proceedings of the Industrial Engineering Research Conference*, pages 1401–1406, Vancouver, Canada, 2008.
- [24] J. Wang and Z. Wang. Using neural networks to determine sugeno measures by statistics. *Neural Networks*, 10(1):183–195, 1997.
- [25] X. Wang, J. Cummins, and M. Ceberio. The bees algorithm to extract fuzzy measures for sample data. In *North American Fuzzy Information Processing Society (NAFIPS'2011)*, El Paso, TX, March 2011.

- [26] Z. Wang, K. Leung, and J. Wang. A genetic algorithm for determining nonadditive set functions in information fusion. *Fuzzy Sets and Systems - Special issue on fuzzy measures and integrals*, 102(3):463–469, 1999.

An Extension Of The Unit Production Elimination Algorithm

X. Chen¹, D. Pager¹

¹Department of Information and Computer Science, University of Hawaii at Manoa, Honolulu, HI, USA

Abstract - Removing unit productions from LR parsing machines can reduce the space and time cost of parsing. The unit production elimination algorithm of Pager may result in redundant states in the generated parsing machine. This work introduces an extension to remove the redundancy and thus minimize the parsing machine. We implemented the unit production elimination algorithm and its extension algorithm into the Hyacc parser generator. We study its performance and discuss relevant issues here. Theoretical analysis and experiment result show that when the extension is used, the parser generation process uses the same amount of memory, but more processing time. The resulted parsing machine can be much more compact.

Keywords: unit production elimination, algorithm, extension, LR

1 Introduction

1.1 Overview

A unit production in a grammar is a production of the form $x \rightarrow y$, where symbol x is a non-terminal, and symbol y is a terminal or non-terminal. The existence of unit productions in a LR parsing machine can increase parsing table size and waste significant amount of parsing space and time [1]. Eliminating unit productions is among the most attractive approaches to optimize LR parsers.

Pager's unit production elimination algorithm is among the most discussed. It is found that the unit production elimination algorithm of Pager, when applied, can possibly lead to redundant states. This work extended the unit production elimination algorithm of Pager [5] by further eliminating redundant states and thus minimizing the parsing machine. The extension algorithm was implemented in the parser generator Hyacc [18][19][20]. Here we present the algorithm, analyze its complexity, conduct empirical study on its performance and discuss relevant issues.

We use these acronyms for the algorithms involved in this discussion: PGM (Pager's practical general method) [6], UPE (Pager's unit production elimination algorithm) [5], UPEExt (Extension algorithm to Pager's unit production elimination algorithm). In addition, LHS stands for Left Hand

Side, and RHS stands for Right Hand Side. The discussion will be based on LR(1), but should in general apply to LR(k).

1.2 Related work

There have been various studies to eliminate unit productions. Anderson, Eve and Horning [1] presented a unit production elimination method, but the method can increase the number of states in the parsing machine significantly. Joliat [8] gave suggestions to simplify the method of Anderson, Eve and Horning. Tokuda [15] presented a method on bypassed LR(k) parsers, which can naturally derive the algorithm of Anderson, Eve and Horning. The methods of Aho and Ullman [2] and Demers [9] can avoid increasing the number of states in the parsing machine, but require restriction on the grammar that no two unit productions should have the same left hand side. Pager [3][4][5] described an algorithm that can avoid the above problems. Backhouse [10] and Lalonde [7] developed variations of Pager's method. Koskimies [11][12] discussed that Pager's method cannot be used during the construction process of a SLR parser and need to be used on a fully constructed SLR parser. Soisalon-Soininen [13] discussed applying Pager's technique only when it does not affect the use of default reductions. Soisalon-Soininen [14] described that Pager's method can possibly cause increase in the size of the parsing machine and presented a fix. Heilbrunner [16] and Schmitz [17] discussed practical conditions needed to correctly eliminate unit productions.

2 Pager's unit production elimination algorithm

Pager's unit production elimination algorithm [3] is applied to a LR parsing machine to further reduce the number of states to achieve a more compact LR parsing machine.

A unit production is a production $x \rightarrow y$ where both x and y are single symbols. A *leaf* is a symbol that only appears on the RHS of any unit production but never on the LHS of any unit production. The algorithm [3] takes five steps: "1) For each state S in the parsing machine (including new states added in step 2), and for each leaf x where the x -successor of S contains a unit reduction, do step 2. Go to step 3 after finish. 2) For $1 \leq i \leq n$, let x_i be the symbols such that $x_i \Rightarrow x$ (including x itself), and for which shift/goto actions are

defined at state S . Let the x -successor of S be T_1 . If any state R is or at an earlier time has been a combination of states T_1, \dots, T_n , then let R be the new x -successor of state S ; otherwise combine states T_1, \dots, T_n into a new state T and make T the new x -successor of S . 3) Delete all the transitions where the transition symbol is on the LHS of a unit production. 4) Delete all states that now cannot be reached from state 0. 5) Replace all such reductions $y \rightarrow w$ by $x \rightarrow w$, where y is the LHS symbol of a unit production, and x is a randomly selected leaf such that $y \Rightarrow x$.”

Example 1. Given grammar $G1: E \rightarrow E + T \mid T, T \rightarrow T * a \mid a$. The LR(1) parsing machine of grammar $G1$ is shown in Fig. 1. We have two unit productions that are the candidates of elimination: $E \rightarrow T$ and $T \rightarrow a$.

An example of applying the unit production elimination algorithm on the LR(1) parsing machine of grammar $G1$ is shown in Fig. 2. First we need to find the leaves of the grammar. This is achieved by constructing a multi-rooted tree, which is $E \rightarrow T \rightarrow a$ for $G1$. In this case a is the only leaf. Then following the algorithm step 1, we see that only states 0 and 4 have a -successor that has a unit production: state 0's a -successor state 3 has a unit production $T \rightarrow a$, state 4's a -successor is also state 3. Thus we follow step 2 to combine successor states of state 0 and state 4. These are shown in (b) and (c) of Fig. 2. Next, (d) follows step 3, (e) follows step 4, (f) follows step 5 and also rearranges the states in a better-looking layout.

3 Extension to the unit production elimination algorithm

3.1 The extension algorithm

It can be noted that after removing unit productions, the parsing machine can possibly contain redundant states with the same actions. These redundant states can be combined to result in a more compact parsing machine. This is a natural extension of Pager's unit production elimination algorithm.

Definition 1. *Equivalent states* are those states in a parsing machine that have exactly the same actions (*accept*, *shift*, *goto* and *reduce*) on each token symbol (including both terminals and non-terminals).

Algorithm 1 (UPEExt) is shown in the next page. It removes redundant *equivalent states* from the parsing machine obtained from Pager's unit production elimination algorithm.

One concern of the unit production elimination algorithm is that it was designed for LR(k) grammars. For non-LR(k) grammars, more conflict complications can be derived. Under such situations, the unit production elimination algorithm and this extension should not be used. Another concern is for unit productions with semantic actions,

these should not be removed so as to retain the associated semantic actions.

3.2 Complexity analysis

In practice, this extension algorithm is $O(1)$ in space and does not increase the amount of memory used, since it operates on the existing parsing machine. But it takes quite a large percentage of the execution time, because it looks through each entry of the entire parsing table for each state.

The worst time performance (upper bound) is $O(n^2 * m)$, where n is the number of states, and m is the number of tokens (both terminals and non-terminals). The best time performance (lower bound) is $O(n * m)$.

Assume the action of accessing one action of one state is $O(1)$. Derivation of upper bound $O(n^2 * m)$ using the best scenario: the step of finding the set of all the *equivalent states* can be done in linear time by inserting all states into a hash table based on its actions. Since there are n states, and assume each state has m actions in average, this is $O(n * m)$. The next step replaces relevant transitions. Assume those *equivalent states* are S_1, S_2, \dots, S_k ($k \leq n$). Let the number of actions transiting into S_i be X_i ($i = 1, \dots, k$). In the worst case all the n other states transiting to S_i and all the m actions of each state transit to S_i (although this is unlikely in practice), so $0 \leq X_i \leq n * m$. The total number of transitions to replace is $0 \leq X_1 + \dots + X_k \leq n * (n * m)$. Thus $O(n * m + n * (n * m)) = O(n^2 * m)$.

For the lower bound $O(n * m)$, just notice that the first step of finding the set of all the *equivalent states* always takes $O(n * m)$, and the second step of replacement in the best case takes no time when no *equivalent states* are found.

3.3 Implementation in Hyacc

Pager's unit production elimination algorithm and the extension algorithm here are implemented into LR(1) parser generator Hyacc [18][19][20].

Implementation can be on the level of 1) the parsing machine automata, or 2) the parsing table. In Hyacc the UPE algorithm is implemented by manipulating the parsing table, so the extension algorithm UPEExt is implemented based on this way. We think that manipulating the parsing table is easier. The alternative of working on the level of the parsing machine automata, however, may be more intuitive from a human point of view.

Hyacc uses reduced-space LR(1) parser generation algorithms, such as Pager's PGM algorithm. In general the user of Hyacc applies the UPE and UPEExt algorithms on a parser generated from the PGM algorithm or other reduced LR(1) algorithms. In the example and empirical studies below we assume this scenario.

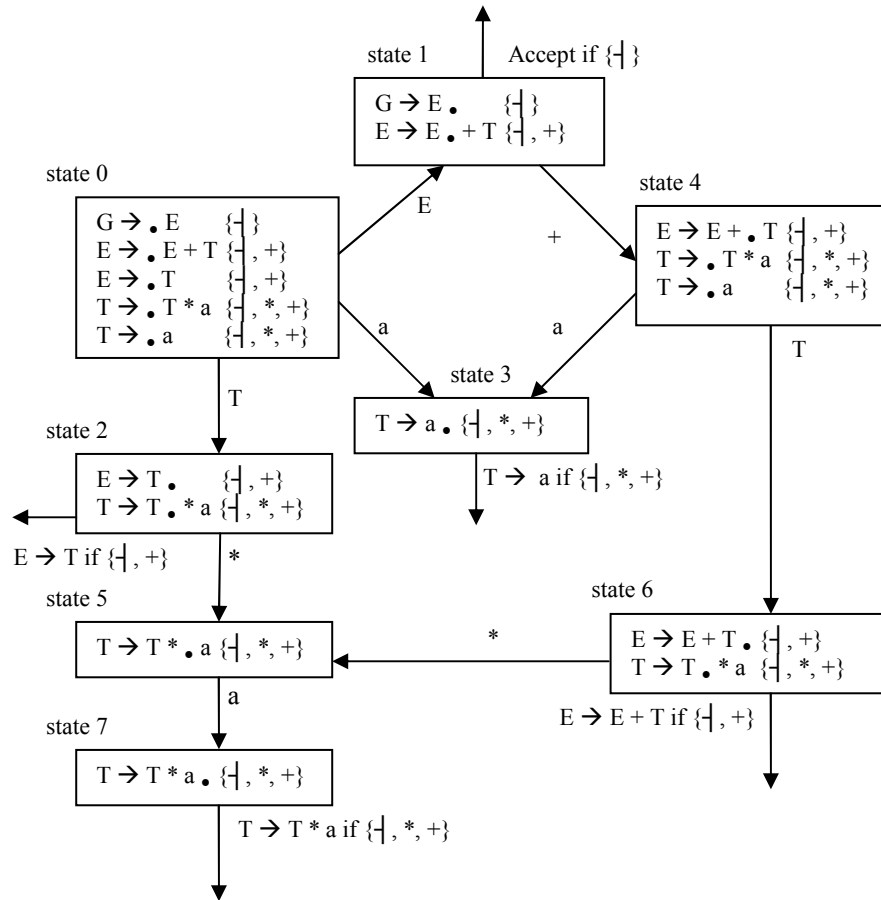


Fig. 1. Parsing machine of grammar G1

Algorithm 1 (UPEExt):

Input: Parsing Machine M

Output: A parsing machine M' where all the equivalent states in M are removed;

- 1 let $\text{Shift}(X, k) \rightarrow Y$ be a Shift transition from state X to state Y on token k;
- 2 **foreach** state S in M **do**
- 3 find the set Σ of all the equivalent states of state S;
- 4 **foreach** state S' in Σ **do**
- 5 **foreach** $\text{Shift}(R, k) \rightarrow S'$ in M **do**
- 6 replace it by $\text{Shift}(R, k) \rightarrow S$;
- 7 **end**
- 8 **end**
- 9 remove Σ from M;
- 10 **end**

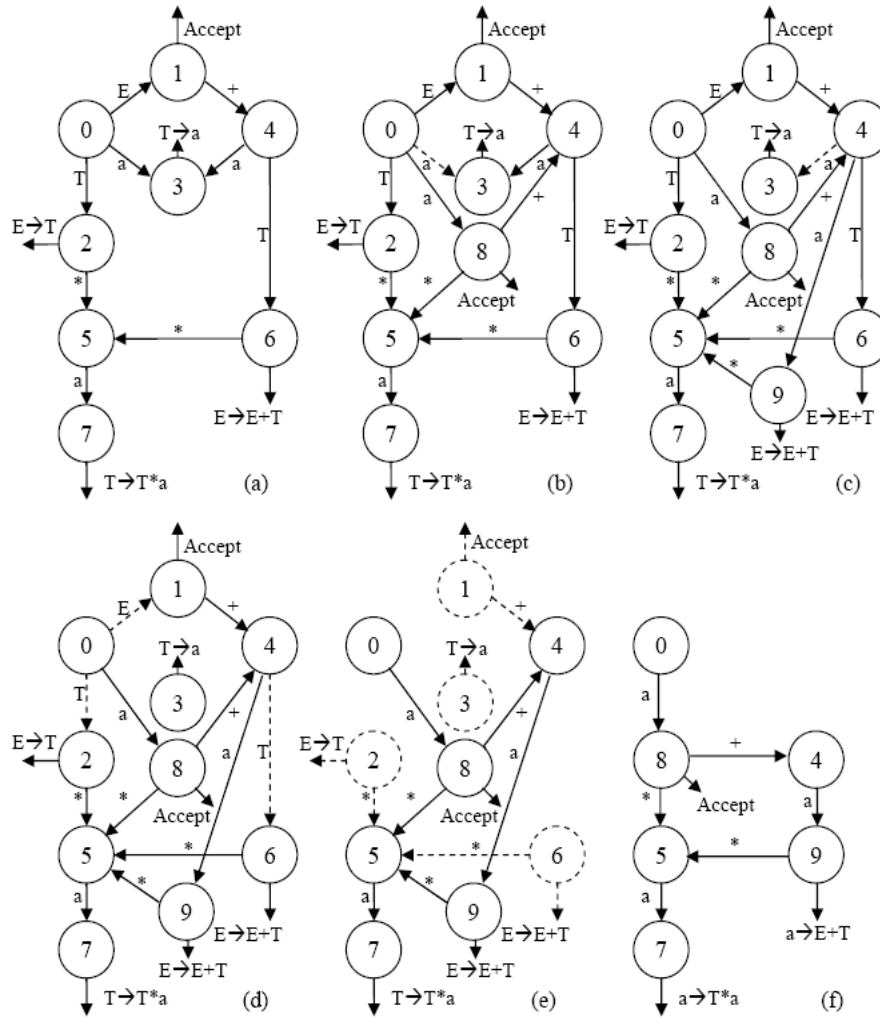


Fig. 2. Apply Unit Production Elimination on the LR(1) parsing machine of grammar G1

(a) Original parsing machine. (b) Combine states 1, 2 and 3 to state 8. Remove link $0 \rightarrow 3$ because there can be only one a-successor for state 0. (c) Combine states 3 and 6 to state 9. Remove link $4 \rightarrow 3$ because there can be only one a-successor for state 2. (d) Remove transitions corresponding to LHS of unit production: E, T. (e) Remove all states unreachable from state 0, and remove their associated action links. (f) Replace LHS of reductions to corresponding leaf.

3.4 An example

Example 2. Given grammar $G2: S \rightarrow d i A, A \rightarrow A T | \epsilon, T \rightarrow M | Y | P | B, M \rightarrow r | c, Y \rightarrow x | f, P \rightarrow n | o, B \rightarrow a | e$. In Fig. 3, (a) is the parsing machine obtained using the practical general method, (b) is the parsing machine after applying the unit production elimination algorithm based on (a), (c) is the parsing machine after applying the extension to the unit production elimination algorithm based on (b). In (b), states 18 to 25 all have the same action $A \rightarrow A T$ for each of the lookahead symbols in $\Sigma = \{ a, c, e, f, n, o, r, x, _ \}$. Thus states 18 to 25 are equivalent states and they can be combined into one state, i.e., state 18 in (c).

In this example, the parsing machine in (a) has 18 states, in (b) has 13 states, and in (c) has only 6 states. So by applying the extension algorithm after the unit production elimination, $(13 - 6) / 13 = 54\%$ reduction in parsing machine size is achieved. The following table compares the number of states, 'shift/goto', 'reduce' and 'accept' actions in the parsing machine after applying each of the PGM, UPE and UPEExt algorithms.

4 Measurements And Evaluations

Measurement data are collected on a Dell Inspiron 600M computer with 1.7GHz Intel Pentium CPU and 1 GB RAM. Operating system is Fedora core 4.0. In the measurements, unit of time is in sec (second), and memory is in MB (megabyte). Hyacc version 0.95 is used. In the empirical study, we measure the performance on three algorithms: PGM, UPE, and UPEExt. This is because UPE is applied after PGM is applied, and UPEExt is applied after UPE is applied. We would like to see the difference of parsing table size, time and memory costs after applying the UPE and UPEExt algorithms. The grammars of 13 real programming languages [21] are used for the study.

4.1 Parsing table size comparison

Table 2 shows the parsing table size comparison. Fig. 4 is the graphic view.

UPE may decrease the number of states as in the case of many simple grammars. but in 12 out of the 13 real programming languages here, UPE actually increases it. Applying the UPEExt algorithm decreases the parsing machine size significantly: although in 10 out of the 13 real language grammars the number of states are still bigger than that of PGM, they are bigger only by a small margin. Therefore it is desirable to apply the extension algorithm. In addition, the number of rules in the parsing machine is also reduced, since unit productions are removed.

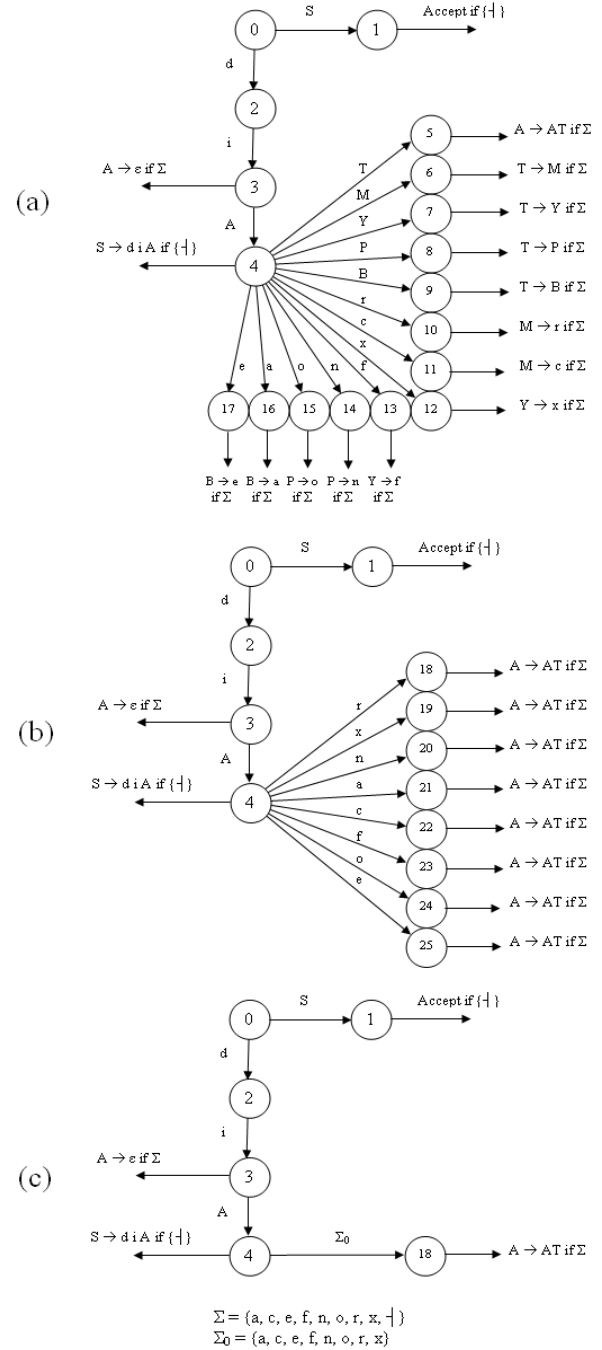


Fig. 3. Remove *equivalent states* after unit production elimination

Table 1. Parsing machine comparison after applying PGM, UPE and UPEExt algorithms

	State #	shift/goto	reduce	accept
PGM	18	17	15	1
UPE	13	12	10	1
UPEExt	6	5	3	1

Table 2. Parsing table size comparison.

Grammar	PGM		UPE		UPEExt	
	State #	Rule #	State #	Rule #	State #	Rule #
Ada	873	459	1074	262	805	262
Algol 60	274	169	498	92	412	92
C	349	212	786	116	380	116
Cobol	657	401	646	268	528	268
C++ 5.0	1404	665	3573	443	2255	443
Delphi	609	358	1195	200	669	200
Ftp	200	74	211	71	211	71
Grail	193	74	247	54	204	54
Java 1.1	439	266	1174	142	673	142
Matlab	174	93	374	53	178	53
Pascal	418	257	844	119	427	119
Turbo Pascal	394	222	649	116	353	116
Yacc	128	103	134	87	134	87

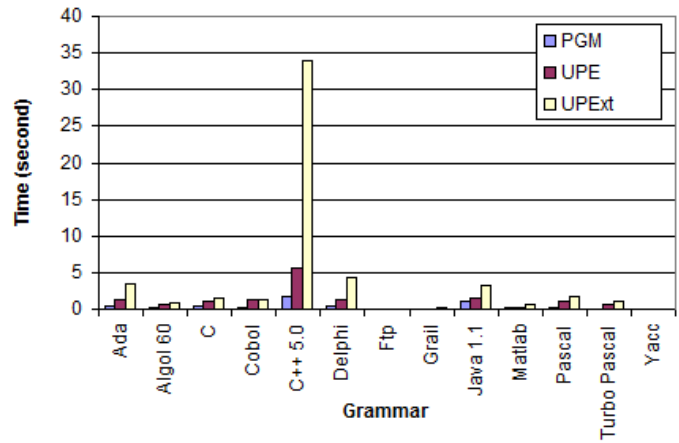


Fig. 5. Running time comparison.

4.3 Memory usage comparison

Table 4 shows the memory usage comparison, and Fig. 6 is the graphic view. When using UPE and UPEExt, there is a slight increase in memory. It can also be seen that UPE and UPEExt use the same amount of memory, because UPEExt only works on the existing parsing table.

Table 4. Memory usage comparison

Grammar	PGM	UPE	UPEExt
Ada	7.9	9.4	9.3
Algol 60	4.2	4.2	4.2
C	6.0	6.0	6.0
Cobol	6.3	6.4	6.4
C++ 5.0	23.9	30.9	30.9
Delphi	6.5	7.5	7.5
Ftp	2.8	2.9	2.9
Grail	2.9	2.9	2.9
Java 1.1	7.8	8.6	8.6
Matlab	3.9	3.9	3.9
Pascal	4.9	5.7	5.7
Turbo Pascal	4.3	4.3	4.3
Yacc	2.6	2.7	2.7

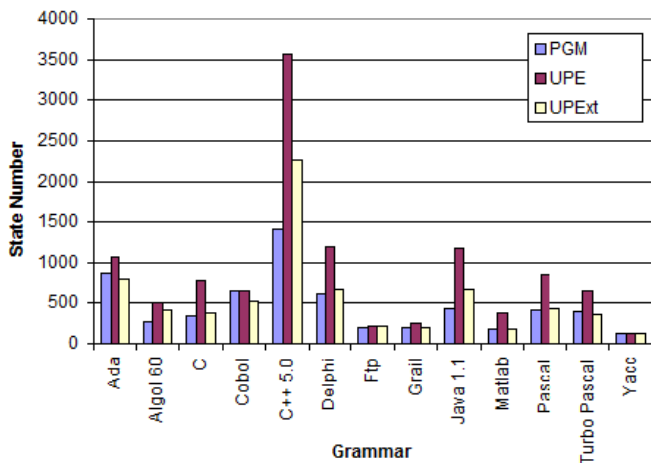


Fig. 4. Parsing table size comparison.

4.2 Running time comparison

Table 3 shows the running time comparison, and Fig. 5 is the graphic view. Compared to PGM, UPE and UPEExt use longer running time, sometimes significantly longer, especially for the UPEExt algorithm. This is as expected.

Table 3. Running time comparison

Grammar	PGM	UPE	UPEExt
Ada	0.406	1.342	3.452
Algol 60	0.290	0.566	0.931
C	0.420	1.142	1.418
Cobol	0.127	1.205	1.206
C++ 5.0	1.779	5.680	33.986
Delphi	0.335	1.347	4.371
Ftp	0.017	0.035	0.035
Grail	0.024	0.066	0.119
Java 1.1	1.026	1.563	3.328
Matlab	0.189	0.307	0.637
Pascal	0.174	1.061	1.787
Turbo Pascal	0.098	0.587	1.159
Yacc	0.026	0.043	0.043

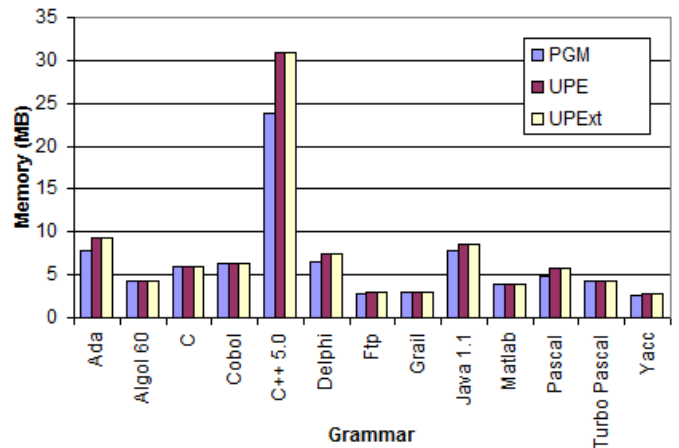


Fig. 6. Memory usage comparison

5 Conclusions

Redundant states exist in the parsing machine after applying Pager's unit production elimination algorithm. An extension is used to remove redundant states, and reduces the size of the parsing machine significantly. Measurements show that when the extension is used, parser generation takes the same amount of memory but more time, and the resulted parsing machine can be much more compact. Unit production elimination algorithm and its extension should be used on LR grammars only.

Although the extension algorithm does not require extra space to run other than needed by the unit production elimination algorithm itself, it may need much longer running time. Since parser generation is a one-time process, it should be worth such an effort.

6 References

- [1] T. Anderson, J. Eve, and J. J. Horning. Efficient LR(1) parsers. *Acta Informatica*, 2:12–39, 1973.
- [2] Alfred V. Aho and Jeffrey D. Ullman. A technique for speeding up LR(k) parsers. *SIAM J. Computing*, 2:2, 106-127. 1973.
- [3] David Pager. On eliminating unit productions from LR(k) parsers. Technical Report PE 245, University of Hawaii, Honolulu. 1973.
- [4] David Pager. On eliminating unit productions from LR(k) parsers. *Automata, Languages and Programming, Lecture Notes in Computer Science*, Volume 14, 242-254. 1974.
- [5] David Pager. Eliminating unit productions from LR parsers. *Acta Informatica*, 9:31 – 59, 1977.
- [6] David Pager. A practical general method for constructing LR(k) parsers. *Acta Informatica*, 7:249 – 268, 1977.
- [7] Wilf R. LaLonde, On directly constructing LR(k) parsers without chain reductions, *Proceedings of the 3rd ACM SIGACT-SIGPLAN symposium on Principles on programming languages*, p.127-133, January 19-21, 1976, Atlanta, Georgia.
- [8] M. L. Joliat. A Simple Technique for Partial Elimination of Unit Productions from LR(k) Parsers. *IEEE Transactions on Computers*, Volume 25 Issue 7, 763-764, July 1976, IEEE Computer Society Washington, DC, USA
- [9] Demers, A. J. Elimination of single productions and merging nonterminal symbols of LR(1) grammars. *Computer Languages* 1:2, 105-119. 1975.
- [10] Backhouse, R. C. An alternative approach to the improvement of LR(k) parsers. *Acta Informatica* 6:3, 277-296. 1976.
- [11] Koskimies, Kai {1976} Optimization of LR(k) parsers (in Finnish). M.Sc. Thesis, University of Helsinki, Helsinki.
- [12] Koskimies, Kai (1979) On a method for optimizing LR parsers. *International Journal of Computer Mathematics* 7(4).
- [13] Eljas Soisalon-Soininen. Elimination of single productions from LR parsers in conjunction with the use of default reductions. 1977.
- [14] Eljas Soisalon-Soininen. On the space optimizing effect of eliminating single productions from LR parsers. *Acta Informatica*. Volume 14, Number 2, 157-174. 1980.
- [15] Takehiro Tokuda. Eliminating unit reductions from LR(k) parsers using minimum contexts. *Acta Informatica*. Volume 15, Number 4, 447-470. 1981.
- [16] Stephan Heilbrunner. Practical conditions for correct elimination of chain productions from LR parsers. Length 77 pages. *Hochsch. der Bundeswehr München, Fachbereich Informatik*, 1983.
- [17] Schmitz, Lothar (1984) On the correct elimination of chain productions from lr parsers. *International Journal of Computer Mathematics* 15(1-4)
- [18] Xin Chen. LR(1) Parser Generator Hyacc. Available: <http://hyacc.sourceforge.net>. January 2008.
- [19] Xin Chen, David Pager. LR(1) Parser Generator Hyacc. *Proceedings of International Conference on Software Engineering Research and Practice*, p.471-477. *WORLDCOMP'11*, Las Vegas, July 18-21, 2011.
- [20] Xin Chen, David Pager. Full LR(1) Parser Generator Hyacc And Study On The Performance of LR(1) Algorithms. *Proceedings of The Fourth International C* Conference on Computer Science & Software Engineering*, p.83-92. Montreal, Canada, May 16-18, 2011.
- [21] "Yacc-keable" Grammars. Available: <http://www.angelfire.com/ar/CompiladoresUCSE/COMPILERS.html>

Reliable Task Allocation in Distributed System

Vinod Kumar Yadav
Motilal Nehru National Institute
of Technology Allahabad,
UP-211004
Email:vkynita@gmail.com

Swagat Ranjan Sahoo
Motilal Nehru National Institute
of Technology Allahabad,
UP-211004
Email: mlsawat@gmail.com

Dharmendra Kumar Yadav
Motilal Nehru National Institute
of Technology Allahabad,
UP-211004
Email: dky@mnnit.ac.in

Abstract—This paper presents a rigorous framework of efficient task allocation in heterogeneous distributed environment where server nodes can fail permanently. The system performance can be improved by increasing the probability of serving queued tasks in the distributed computing system (DCS) before all the node fails. For a large set of tasks that is being allocated into a distributed environment, several allocation methods are possible. These allocations can have significant impact on quality of services such as reliability, performance etc. In distributed environment reliability is highly dependent on its network and failures of network have adverse impact on the system performance. So, one possible way for improving reliability is to make the communication among the tasks as local as possible. Firstly, it divides the whole workload into small and independent units, called tasks and determines expectant hosts from environment. It then applying a two phase hybrid algorithm: Simulated Annealing and Branch-and-Bound (SABB).The Simulated Annealing algorithm finds a near optimal allocation from expectant hosts and then find an optimal allocation by applying the branch-and-bound (BB) techniques by considering the Simulated Annealing algorithm as initial solution. This algorithm overcomes the less improved quality obtained by heuristics as well as the computational cost of the exact algorithms.

Key Words: Reliability, Queuing Theory, Distributed Computing, Communication Links, Renewal Theory.

I. INTRODUCTION

A distributed computing system is a collection of heterogeneous processors interconnected by a communication network. Each processor has its own local memory and other peripherals, and the communication between any two processors of the system takes place by message passing over the communication network. Such systems provide very powerful platform for executing high performance parallel applications, alternative to the very expensive massively parallel machines. But the performance of the system is highly dependent on the tasks allocation onto the available machine. Because different application requires various hardware and software, So these application components will provide their expected functionality only when their requirements will be satisfied. Such type of problem in distributed computing system is referred as task allocation problem.

Typically the communication networks interconnecting the servers suffer the problem of low bandwidth. In order to

enhance the processing capabilities of distributed computing system, workloads are partitioned into small independent units called tasks. To allocate these tasks among the processors is the main concern while maintaining the system reliability. For complex application where tasks can be allocated to different hosts in distributed environment, several tasks allocation method are available. Some of them are more effective than others for some given context in terms of quality of services such as communication cost, network congestion, dependability, performance etc.

In distributed environments network failure is the most potential problem that can lead to disastrous effects on the systems reliability and the software application may not provide its expected functionality. For minimizing to this risk one way is to make the communication among the tasks as local as possible. In this manner the tasks that are allocated on the same host of the DCS can communicate without any respect to the networks status.

In this paper reliability can be achieved by carefully assigning the tasks onto the processors of the DCS by taking into account to failure rate of both the communication links as well as the processors. Here the idea is to assigning tasks with longer execution time to more reliable processors and communication links. This paper also solves the problem of finding the optimal solution for reliable task allocation problem. It first determines the expectant hosts and then develops a mathematical model based on a cost function(execution time, inter-processor communication etc) and then apply a two phase hybrid algorithm: first is Simulated Annealing (SA) [1,3] and second is Branch-and-Bound(BB)[2,4,14,15]. The Simulated Annealing algorithms finds a near optimal solution and then find an optimal solution by applying Branch-and-Bound algorithm by considering the results of Simulated Annealing algorithm as initial solution.

This paper considers a graph - based approach for task allocation. For maximizing the reliability of the application, peer-to-peer distributed environment has taken into consideration. A communication link is a peer-to-peer communication medium with well defined characteristics and behavior [5]. In peer-to-peer architecture two or more computers can directly communicate with each other without requiring any intermediate devices [6,13,16]. In our consideration the tasks are allocated

with respect to the various parameters of communication links that different hosts can supports in distributed environment.

Rest of the paper is organized as follow: Section II formally defines the problem statement. Section III determines the hosts over which tasks can be assigned. Section IV presents a reliable task allocation model. Section V describes the algorithm used for task allocation. Section VI shows the experimental results. Finally Section VII concludes and outline future works.

II. PROBLEM STATEMENT

In a distributed computing system which consists of a set of N heterogeneous computers communicating over a fully connected network and M independent tasks have to be processed by the system. Each computer of the DCS has some capabilities and communication links have some capacities and a probability of failure is associated with each component of the system. We also associate a vector with each task which represents its execution cost at the different computers in the system. The purpose is to allocate M tasks onto the processors of the DCS in such a way that the requirements of tasks gets satisfied and the reliability of the system is maximized.

III. DETERMINATION OF EXPECTANT HOSTS

For determining expectant hosts following steps are considered: Firstly, we allocated the application components secondly, the tasks are allocated in distributed environment and at the end expectant hosts for application components are determined.

A. Allocation of application components

This step specifies the set of application components or tasks that has to be allocated onto the distributed environment for execution. These tasks are connected by a set of communication links that has different characteristics. These components can be compared with nodes of the graph and edges as communication links. These communication links (channels) can be of several types like synchronous, asynchronous, FIFO, etc.

B. Application components graph

Let n_i represents the various components of the application and l_j represents the various communication links. Then the component graph $C_g = (V_{cg}, E_{cg})$ is defined as a graph where $V_{cg} = \{n1, n2, n3, \dots, nN\}$ and $E_{cg} = \{11, 12, 13, \dots, 1M\}$.

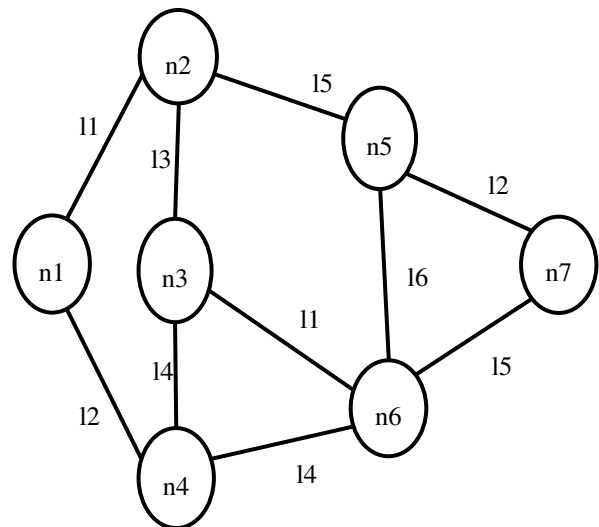


Figure.1 Component Graph

C. Distributed environment

In distributed environment the tasks are allocated independently on the basis of their requirements.

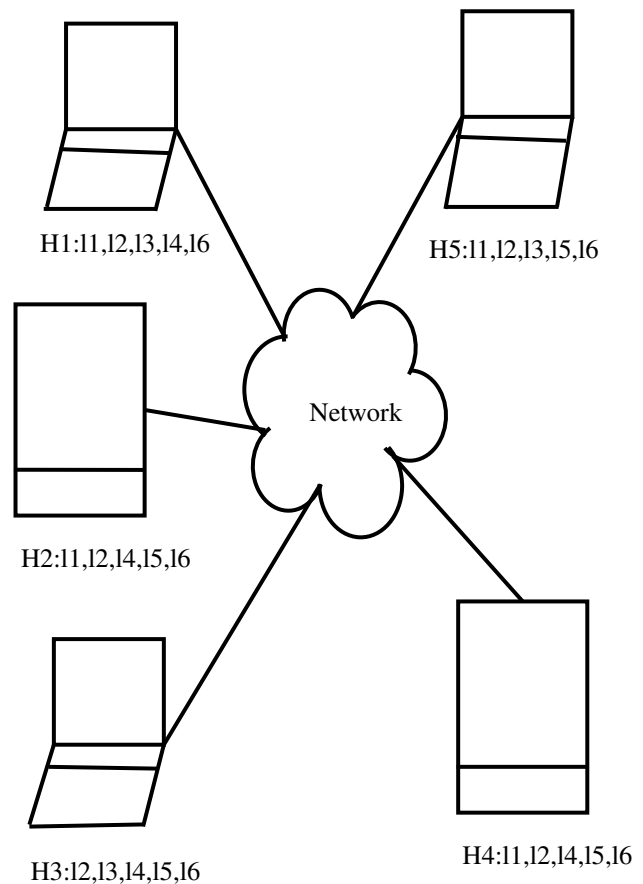


Figure.2 Distributed Environment

Because different hosts of distributed environment having different capabilities, so they execute some specific set of tasks whose requirement gets satisfied by the corresponding host.

Figure.2 shows the heterogeneous DCS consists of five hosts (H1, H2, H3, H4, H5). These hosts are connected by different set of links (11-16).

D. Expectant hosts

For a given component graph $C_g = (V_{cg}, E_{cg})$, host H_j is an expectant host for the allocation of component N_i , only if $l_{N_i} \subseteq l_{H_j}$, where l_{N_i} is the communication links required by component N_i in component graph and l_{H_j} is the communication links supported by host H_j in distributed environment.

Component name	Expectant host
n1	H1,H4,H5
n2	H2,H5
n3	H1,H2
n4	H1,H4
n5	H4,H5
n6	H2,H4
n7	H3,H4,H5

Table-1:Shows the expectant hosts for components

Table-1 shows the candidate hosts for allocating the components of the application from fig-1 to the target distributed environment presented in fig-2. Here we can see that the application components n1 can be allocated either on host H1 or H4 or H5 and task n2 can be allocated on only H2 and H5, and task n3 can be allocated on host H1 and H2 and so on. Only these set of hosts can satisfy the requirements of the respective components.

IV. RELIABLE TASK ALLOCATION MODEL

Reliability of a distributed computing system can be seen as the reliability of its processors as well as the reliability of its communication links. Each component of the distributed system may exist in one of the two states: operational or faulty. For successful execution each processor must be operational during the time of execution of tasks. The communication links must be active during the communication between the end processors. For reliable task allocation it is also necessary that the cost of a task should be minimum. Cost of a task defined in terms of its execution cost and communication cost.

The system reliability for a mission is defined as the time interval during which the system to be active. During a process a task may execute more than once. The Accumulative Execution Time (AET) of a module running on a processor is total execution time incurred for this module running on that processor during the mission i.e. the product of the number of times this module executes during the process and the average time unit for each execution on that processor [7]. And the inter-module communication (IMC) between two modules is the product of the number of times they communicates and the average number of words exchanged in each communication

[7]. A detailed discussion of AET and IMC can be found in [8].

A. Notations and Descriptions

The notations specified here used in rest of the paper:

R_p - Reliability of processor P

T - The set of tasks.

t_n - n^{th} module of the task set T.

P - The set of processors.

P_i - i^{th} processor in P.

L - Set of communication links.

l_j - j^{th} communication link in L connecting the processors P_a and P_b .

ψ_i - Failure rate of processor P_i .

X - A binary matrix ($M \times N$) corresponding to a task assignment.

C_{ni} - Accumulative execution cost of task n on processor P_i .

X_{ni} - An assignment of n^{th} task on i^{th} processor.

ω_{ab} - Failure rate of communication link (l_j)connecting the processors P_a and P_b .

C_{mnab} - The cost of transferring data between task m and n by using communication link l_j (connecting to two processors P_a and P_b).

R_s - Reliability of the distributed system i.e. the product of the reliability of the components of distributed computing system.

R_l - Reliability of communication links.

B. Reliability of processors (R_p)

The reliability of a processor p_i is the probability of being operational during the time interval 't' till the execution of tasks are completed that are assigned to it. If a failure rate of processor p_i is ψ_i , then the reliability of processor p_i is $\exp(-\psi_i t)$ [7, 9, 10]. The reliability of processor p for an assignment X, and accumulative execution cost C for task t_n running on it is defined as:

$$R_p = \exp(-\psi_i \sum_n C_{ni} X_{ni}) \dots (4.1)$$

This expression gives the total time taken for executing the tasks assigned on n^{th} processor.

C. Reliability of communication links (R_l)

The reliability of a communication link l_i (connecting two adjacent processor p_a and p_b) is the probability of being operational during the time interval 't' till the communication of task has completed between adjacent processors. If the failure rate of communication link is ω_{ab} , then the reliability of communication link l_j is $\exp(-\omega_{ab} t)$ [7, 9, 10]. The reliability of communication link for an assignment X and cost of transferring data between two tasks m and n which are assigned to different processors given as:

$$R_l = \exp(-\omega_{ab} \sum_m \sum_{n \neq m} C_{mnab} X_{ma} X_{nb}) \dots (4.2)$$

The summation gives the required time for communication between processors 'a' and 'b' by using link l_j .

So the reliability of the distributed computing system R_s is defined as the product of the reliability of components of the distributed computing system [12].

$$R_s = [\prod_a R_p][\prod_{m \neq n} R_l] = exp(-s) \dots (4.3)$$

Where

$$S = \sum_a \sum_m \psi_i C_{ni} X_{ni} + \sum_a \sum_{b \neq a} \sum_m \sum_{n \neq m} \omega_{ab} C_{mnab} X_{ma} X_{nb} \dots (4.4)$$

In eqn 4.4 first part of cost function determines the unreliability due to execution of tasks on the processors of the system and second part determines the unreliability due to the inter-processor communication.

The reliability of processors and communication links with respect to tasks are vectored and associated with each task, representing the execution cost of the tasks at the different processors in the distributed system. In figure-1 the set of vertices 'V' shows the set of tasks, edges 'E' shows the set of communication links. In figure-2 H1, H2, H3, H4 and H5 shows the processors of distributed computing system. Now the execution cost of each task at various nodes are calculated. Table-2 represents the communication costs of tasks at expectant hosts. The cost infinity represents the requirements of task is not satisfied at that processor.

	H1	H2	H3	H4	H5
n1	10	∞	∞	12	8
n2	∞	15	∞	∞	20
n3	7	6	∞	∞	∞
n4	14	∞	∞	40	∞
n5	∞	∞	∞	20	6
n6	∞	10	∞	5	∞
n7	∞	∞	11	18	35

Table-2 shows the execution cost of each task at the various processors

V. ALGORITHM FOR TASK ALLOCATION

This section presents a two phase hybrid algorithm for task allocation. In the first phase Simulated Annealing(SA) algorithm and in second phase Branch-and-Bound(BB) algorithm has been used. The SA algorithm determines the near optimal allocation and BB algorithm determines the optimal allocation.

A. Simulated Annealing algorithm

Simulated Annealing Algorithm is a global optimization technique which attempts to find the lowest point in an energy landscape [3,11]. The SA method emulates the physical concepts of temperature and energy to represents and solves the optimization problem. It is often used when the search space is discrete. For certain problems, Simulated

Annealing Algorithm may be more efficient than exhaustive enumeration provided that the goal is merely to find amount of time, rather than the best possible solution.

Simulated Annealing technique comes from metallurgy, involves heating and controlled cooling of a material to increase the size of its crystal and reduce their defects. Due to heat, atoms unstuck from their position (a local minimum of the internal energy) and moves randomly through state of higher energy. The slow cooling gives them more chances of finding configurations with lower internal energy than the initial one. Each step of the Simulated Annealing algorithm replaces the current solution by a random nearby solution, chosen with a probability that depends both on the difference between the corresponding function values and also a global parameter T (called temperature). That decreased gradually during the system equilibrium state through elementary transformations which will be accepted if they reduce the system energy. However as the temperature decreases, small energy increment may be accepted and the system eventually settle down to a low energy state.

The probability of accepting an uphill move (transition) from the current state S to a new state S_1 is specified by an acceptance probability function $P(e, e_1, T)$, where T is the temperature. It depends on the energies $e = E(S)$ and $e_1 = E(S_1)$ of two states. Here it is noted that the probability function P must be nonzero when $e_1 > e$. It means that the system may move to the new state even when it is worse (has a higher energy) than the current one. Here we are using the probability function $\exp(-\delta/T)$ where $\delta = e_1 - e$ (energy difference).

B. Pseudo code

The pseudo code given below presents the Simulated Annealing heuristic as described above [11]. It starts by randomly selecting an initial state S_0 and then calculates the energy (cost) E_{s_0} for this state. After setting an initial temperature T, it generate a random chosen neighbor (S_1) from given set of states and calculate the corresponding energy E_{s_1} . If the energy of S_1 is less than the energy of S_0 , then this solution is accepted as new solution. Otherwise, a probability function $\exp(-\delta/T)$ is evaluated to ensure that the new solution may be accepted as a current solution. When a thermal equilibrium is reached at the current temperature T, the value of T is decreased by a cooling factor α , and inner repetition is increased by an increasing factor β .

In this algorithm the neighborhoods defines procedure to move from a solution point to another solution point. Here it is very simple to determine neighboring solution. It can be obtained by choosing a random task n from the current allocation vector and assign it to randomly selected processor P from expectant hosts processor for the task n. Selection of initial temperature is very important parameter in Simulated Annealing algorithm because if the initial temperature is very high, then the execution time of algorithm becomes very

```

1: Select an initial solution  $S_0$ 
2: Calculate the cost at this solution  $E_{s0}$ 
3: Initial temperature T
4: Cooling factor  $\alpha < 1$ 
5: Inner repetition is increased by factor  $\beta > 1$ 
6:  $S \leftarrow S_0$ 
7:  $e \leftarrow E_{s0}$ 
8:  $S_{best} \leftarrow S, e_{best} \leftarrow e$ 
9:  $K \leftarrow T_{min}$ 
10: while T > K do
11:    $S_{new} \leftarrow neighbor(s)$ 
12:    $e_{new} \leftarrow E(S)$ 
13:   compute  $\delta \leftarrow e_{new} - e_{best}$ 
14:   if  $\delta < 0$  then
15:      $S_{best} \leftarrow S_{new}$ 
16:      $e_{best} \leftarrow e_{new}$ 
17:   else
18:     Generate a random value x in the range(0,1)
19:   end if
20:   if  $x \exp(-\delta/T)$  then
21:      $S_{best} \leftarrow S_{new}$ 
22:      $e_{best} \leftarrow e_{new}$ 
23:      $T \leftarrow T * \alpha$ 
24:     Neighbor(S)  $\leftarrow neighbor(S) * \beta$ 
25:   end if
26: end while

```

long and if it is very low then poor results are obtained. So the initial temperature must be only hot enough to allow an almost free exchange of neighboring solution.

The term cooling factor represents the rate at which the temperature T is reduced. This is also an important factor for the success of Annealing process [11]. The increasing factor represents the rate at which the inner number of repetition is increased with respect to reduction in temperature. It is important to spend long time at lower temperature. In this paper temperature is used as stopping condition.

C. Branch-and-Bound Algorithm

Branch-and-Bound Algorithm is a systematic method for solving optimization problems. It is more suitable for solution of those problems where the Greedy method and Dynamic programming fails.

Branch-and-Bound Algorithm requires two steps. The first one is a way of covering the feasible region by several smaller feasible sub-regions (splitting into sub-regions), this is called branching and second is bounding, which is a fast way of finding upper or lower bounds for the optimal solution within a feasible sub-region. In this paper we are using Simulating Annealing algorithm for calculating the bound. The Branch-and-Bound algorithm has all the elements of backtracking, except simply stopping the entire search process any time a

solution is found. We continue processing until we get the best solution. In addition, the algorithm has a scoring mechanism to always choose the most promising configuration to explore in each iteration. Because of this approach, Branch-and-Bound is sometime called a best-first-search strategy. Starting by considering the root problem (the original problem with the complete feasible region), the bounding procedure are applied to the root problem. If the bound is less than the calculated value, prune that branch and choose another branch.

In case bound is greater than the calculated value choose that one as the new bound. Otherwise, the feasible region is divided into two or more regions. The algorithm is applied recursively to the sub-problems until all the nodes have been solved or pruned, or until some specified threshold is met between the best solution found and the bounds on all unsolved sub-problems

The efficiency of this method is highly depends on the effectiveness of the bounding algorithm (Simulated Annealing algorithm) used, bad choice could lead to repeated branching, without any pruning, until the sub-regions become very small.

VI. EXPERIMENTAL RESULTS:

The proposed idea is tested for a large number of tasks that are allocated onto a distributed system. The idea contains two major parts. The first part determines the expectant hosts. In second part it uses two phase hybrid algorithm (SABB) for efficient task allocation. The hybrid method is coded with object-oriented programming language over Unix platform. The nodes are considered as heterogeneous on the basis of reliability function.

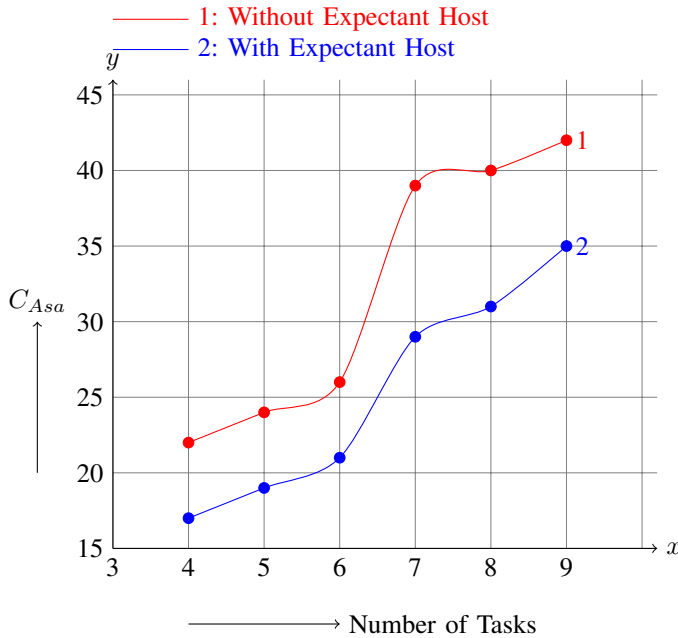
In first part we reduce the number of nodes where a task can be executed. For that we calculate the reliability of each node of the DCS. On the basis of the requirements of the tasks we set the reliability value, below that no task can be executed on a particular node. It can be easily described by following table:

Sl.No	Max Value(1000)	Final Cost	Node Count
1	1000	75	106
2	700	75	142
3	500	75	142
4	300	75	163
5	100	75	163

Table-3 Shows the variation of node counts

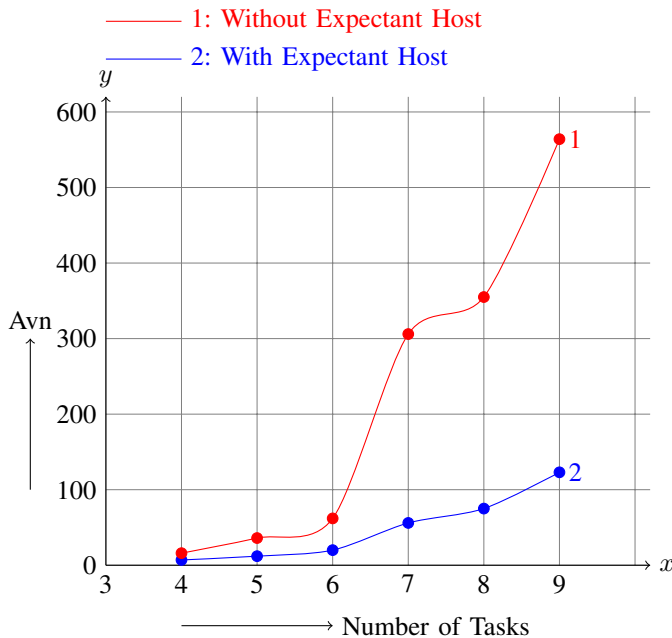
In the above table we can see that when the individual cost assigned to any worker node is equal or greater than the Max bound then the number of nodes visited is minimum (shown in 1st row of the table). In the next of the cases the visiting node is increases (that increase the cost in terms of execution time).

The above simulation results are shown in the following graphs:



Graph.1

Graph.1 shows the bound generated by the simulated annealing algorithm with the consideration of expectant hosts is lower than the bound generated without expectant hosts. C_{Asa} shows the average cost in Simulated Annealing method.



Graph.2

Graph.2 shows the total number of node visited considering both the cases explained in Graph.1. Here the number of node visited is depends upon the bound generated by simulated annealing algorithm. Avn shows the average number of visited nodes.

VII. CONCLUSIONS AND FUTURE WORK:

The task allocation problem in terms of reliability is discussed in this paper. The graph based mathematical approach has been taken to solve the problem. The number of visited nodes have been minimized during the allocation of task by suitably changing the bound value. The determination of expectant hosts has reduced the number of node where a particular task can be assigned. It enhanced the capability of Simulated Annealing Algorithm to determine the bound for Branch and Bound Algorithm. By dynamically updating the bound value we have assigned the tasks in more optimal way. Finally the total number of visited node has been reduced that minimized the over all assignment time, which makes the system more reliable.

The further enhancement would be the content storing strategy of nearest worker node in current node. This may lead to faster allocation of jobs in working nodes.

REFERENCES

- [1] Y. Hamam and K. S. Hindi, "Assignment of program modules to processors: A Simulating Annealing approach", J. of operational research 122, pp.509-513, 2000.
- [2] G. Attiya and Y. Hamam, "optimal allocation of tasks onto networked heterogeneous computers using Minimax Criterion", proc. of the international network Optimization conference (INOC' 03), pp. 25-30, Evry Paris, France 2003.
- [3] E. Aarts and J. Korst, "Simulated Annealing and Boltzmann Machine", John Wiley and sons, New York, 1989.
- [4] W. L. Winston, "operation research: Application and Algorithms", third edition, wadsworth publishing company, Belmont, California, 1994.
- [5] Arbab, F. Reo "A channel - based coordination model for component composition". Mathematical structures in computer science, 14.3 (June, 2004), 329 - 366.
- [6] Schollmeier, R. "A definition of peer - to - peer networking for the classification of peer - to - peer architectures and application", In proceedings of the IEEE 2001 international conference on peer - to - peer computing (p2p 2001), link ding, Sweden, August 27-29, 2001.
- [7] S.M. Shatz, J. P. Wang, and M. Goto, "Task allocation for maximizing reliability of distributed computing system", IEEE trans. computers, vol, 41, no. 9, pp. 1156 - 1168, 1992.
- [8] W.W.Chu et al, "Estimation of intermodule communication (IMC) and its applications in distributed processing system", IEEE trans. comput, vol. 33, no. 8, pp. 691 - 699, Aug 1984.
- [9] S. Kartik and C. S. R. murthy, "Task allocation algorithms for maximizing reliability of distributed computing systems", IEEE trans. computers, vol.46, no.6, 1997.
- [10] D.P. Viyarthi and A. K. Tripathi, "Maximizing reliability of distributed systems with task allocation using simple Genetic algorithm", J. of system architecture 47, pp.549 - 554, 2001
- [11] G. Attiya and Y. Hamam, "Task allocation for maximizing reliability of distributed system: a simulated annealing approach", journal parallel and dist. computing vol. 66. pp.1259 - 1266, 2006.
- [12] G. Attiya and Y. Hamam, "Reliability oriented task allocation in heterogeneous distributed computing systems", proc. ninth int'l symp. computers and comm., pp. 68 - 73, 2004.
- [13] A. Heydarnoori, F. Mavaddat, "Reliable deployment of component - based application into distributed environments", Third international conference IEEE 2006.
- [14] A. O. Charles Elegbede, C. Chu, K. H. Adjallah, F. Yalaoui, "Reliability allocation through cost minimization", IEEE trans. reliab. 52 (2003) 106 - 111.
- [15] S. Srinivasan and N. Jha, "Safety and reliability driven task allocation in distributed systems", IEEE trans. on parallel and distributed systems, vol.10, no.3, pp. 238 - 251, 1999.
- [16] C. C. Hai and S. T. Chanson, "Allocating task interaction graph to processors in heterogeneous networks", IEEE trans. on parallel and distributed system, vol. 8, no. 9, 1997.

On Overcoming Market-Driven Software Development Challenges: *requirements refactoring*

B. Isong¹ and O. Ekabua²

¹ Department of Computer Science and Information Systems, University of Venda, Thohoyandou, South Africa

² Department of Computer Science, North West University, Mmabatho, South Africa

Abstract - *Market-driven (MD) software development organizations are faced with several challenges resulting from their requirements engineering (RE) processes. This is because software is developed for a large market, rather than for a specific customer involving large number of stakeholders/users and continuous requirements inflow. Among these challenges are poor quality of requirements writing/understanding and requirements duplication. Having large number of stakeholders and different forms of requirements poses a great risk for duplicating requirements which in turn increases risk for requirements overload. Furthermore, writing quality requirements that is traceable and understandable is a great issue. These have posed huge challenges to MD development organizations. To rid these issues and improve the quality of requirements specification, we proposed the incorporation of refactoring into MDRE processes. Refactoring requirements will increase requirements understanding, and facilitate detecting inconsistencies, managing requirements changing and even facilitate development decisions like prioritization, releases planning, etc. The purpose of this paper is to propose the incorporation of requirements refactoring into MDRE processes. In addition, we present MD development challenges, software refactoring, opportunities for refactoring and requirements refactoring.*

Keywords: Refactoring, Requirements, Code Smell, Market-driven, Software quality

1 Introduction

In recent years, MD software development is gaining increased momentum and attention in the software engineering world. This is due to the proliferation of the market for COTS (Commercial off-the-shelf) or packaged software [1][2][3]. In MD development, software is developed for a large market, rather than for a specific customer, new versions are developed in a succession of releases, and there is a high pressure on short time-to-market, etc calling for [4]. To meet the market demand, an effective engineering of software requirements is indispensable.

MD software development organization faces many challenges that differ from those found in organizations developing bespoke software. These challenges lie in the distinct RE processes existing between them. MDRE is

characterized by continuous requirements inflow that requires screening and selection, large number of stake-holders and users on a large/ open market as well as schedule constraints [3][4]. These characteristics have posed huge challenges to MD software development organizations. Among these challenges are the issues of writing understandable requirements and the continuous flow of new requirements [1][25]. The continuous flow of requirements is caused by the variety of stakeholders with diverse ideas to contribute. Having large number of stakeholders and different forms of requirements [5][6] poses a great risk for duplicating requirements which can easily increase the risk for requirements overload [5]. In addition to stakeholder's characteristics is the issue of writing quality requirements that is traceable and understandable [1][8]. Requirement is believed to have different meaning to different people and there is no direct link between the stakeholders and the developers. These however, decrease the quality of the requirements and affect other development decisions and activities such as selection and release planning decisions, requirements change management, requirements, implementation, etc negatively.

To satisfy other development decisions and meet market demands, it is important to have quality requirements since good requirements are critical to software quality. With the problems of requirements inflow (i.e. duplicate requirements) and poor requirements writing, software engineers need promising tools and techniques to address these issues. Such a promising technique is refactoring, one of the most important and commonly used techniques of transforming a piece of software artifact in order to improve its design/organization. According to [7][10], refactoring "is the transformation from one representation form to another at the same relative abstraction level, while preserving the subject system's external behavior (functionality and semantics)". Better software development methods and tools exist but most requirements' behaviors are not preserved, instead making it more complex and hard to understand.

Refactoring has received considerable attention and has been used in the context of software evolution to improve the quality of the software such as maintainability, complexity, extensibility, reusability, modularity, efficiency, etc [9]. Refactoring is one of the ways to make object oriented software systems maintainable through reducing its

complexity by incrementally improving the internal software quality without affecting its external behavior [24]. It has application in software artifacts like codes, design and requirements, etc. [10].

In most published literature, refactoring has been used in the context of code restructuring and much has not been done on requirements. We believe incorporating refactoring into MDRE processes will go a long way improving requirements quality and eliminates excessive requirements duplicate. The goal of this paper is to propose the incorporation of requirements refactoring into MDRE in order to improve the quality of specifications and facilitates other development decisions. In addition, we present the challenges faced by MD development organizations (i.e. poor requirements writing and requirements duplicate), refactoring activities, and the application of refactoring to requirements. These are discussed in subsequent sections.

2 Market-driven Development RE Challenges

The goal of a MD product is to take market shares by drawing a wide range of customers, perhaps, increasing profit [11][12]. In order to meet this objective, it is of essence for the RE process to invent quality requirements devoid of duplication, low quality specification etc. that can compete on the market or facilitate other development decisions. However, MD software development faces special challenges resulting from stakeholdings and scheduling constraints [3][4]. In the results of an empirical study conducted by [1][25], several challenges were identified. These challenges include communication gap between marketing staff and developers, writing understandable requirements, managing the constant flow of new requirements, requirements volatility, requirements traceability and interdependencies, requirements are invented rather than discovered, implementing and improving RE within the organization, resource allocation to RE, release planning based on uncertain estimates, etc.

Of special interest in this paper among these challenges are the issue of writing understandable requirements and the continuous flow of new requirements. The continuous flow of requirements is caused by the diverse stakeholders who have influence on the product and like to contribute their ideas. Requirements are continuously collected, stored in a database for further analysis, described in natural language, and of varying quality and nature [4][8]. With these, there is a great risk for duplicating requirements due to large number of stakeholders and different forms of requirements [5][6], which in turn increases the risk for requirements overload.

In addition, the issue of writing understandable requirements and understanding the stated requirements poses a serious

case. Many requirements are shorten and poorly written [1][8][25]. The possible cause is that requirements have different meaning to different people and there is no direct link between the stakeholders and the developers. As a result, the quality of the requirements is poor since the variety and large set of requirements that is available in the database cannot be handled or resolve ambiguities, find relationships, eliminate duplicates, etc.

Consequently, this issue can affect other development decisions and activities in MD such as selection and release planning decisions, requirements change management, requirements, implementation, etc negatively. Therefore there is need to get rid of these problems and improve the quality of requirements specification as it is critical to the overall software quality. Hence, the need for re-engineering techniques such as refactoring is a *sine qua none* to MDRE.

3 What is Software Refactoring?

Software refactoring is as old as creation. Perhaps, many software engineers have been carrying out refactoring unknowingly during software development. Today refactoring is considered an integral part of many development projects [14]. It is one of the tools developers use in cleaning up codes in a more efficient and controllable manner [15]. With the need to modify existing code, refactoring constitutes a highly disciplined approach to restructuring the design without changing the behavior of the code.

Refactoring is a change made to the internal structure of software to make it easier to understand and cheaper to modify (i.e. improving the design) without changing its observable behavior [15]. The process of refactoring code removes duplication, the simplification of complex logic, and clarifies unclear code [14]. The process starts with the identification of when an application should be refactored, proposing which refactoring technique should be applied and where the application of the selected refactoring is to take place [10]. When refactoring is successfully applied to a piece of code, its design is improved. The fact is that creating a good design is a challenging task. Design defects cause the system to display high complexity, improper behavior, and poor maintainability [27]. To cope with these issues, developers are forced to adapt the software design through series refactorings - the only way to keep the software easier to modify and understand [10][15].

Refactoring is an important programming practice and has been in existence for some times now. Initiated by the Smalltalk community, it has been applied to a great number of programming languages or integrated development environments (IDE), software process support (such as software re-engineering, agile software development, and

framework-based software development) and many software artifacts such as programs, design, and requirements [10].

4 Motivations for Software Refactoring

Refactoring is generally motivated by detecting the presence of a 'smell' (i.e. warning signs about potential problems) in a software artifact [10][15]. The occurrence of these 'smells' reduces the overall quality of the software artifact throughout the development process [10][16]. For instance, long method or duplicate method may exist in a code. Once noticed, such situations can be solved by applying the right refactoring. That is, restructuring the source code into a new form that no longer 'smells' without changing its observable behavior [15]. Failure to perform refactoring can affect the quality of the code in terms of maintainability, extensibility, etc in a way not anticipated. Removing the 'smells' early during the stages of development process reduces the costs associated with the software changes. These cost reductions could double five times more in later stages than during requirements activities [17].

Major factors that have influenced program refactoring stems from hard to read or duplicate logic/complex conditional logic programs are hard to modify, etc [15]. Therefore applying refactoring to codes makes it easier to add new codes, improve the design of existing codes, and gain a better understanding of the codes [14][15]. In a study to investigate what motivates software engineers to refactor their code, [18] identified major factors. The report showed that 'smell' is not the only motivating factor for refactoring. Factors such as responsibility with code authorship, self esteem, self efficacy, achievement needs, social norm, unconscious habit, threats of punishment, perceived value, estimated efforts, recognitions from others, tools availability and functionality etc. All these factors contribute to the need to perform refactoring.

5 Requirements and Refactorings

In various published literature, software refactoring efforts have so far been limited to source codes. Refactorings can be applied to any type of software artifacts such as design models, database schemas, software architectures, and software requirements [10][15]. Like source codes, refactoring can also be applied at the level of requirements specifications. As suggested by [10][19], natural language requirements specifications can be restructured by decomposing them into a structure of viewpoints with each having partial requirements of some system components with interactions made explicit. When requirements are restructured in this manner its quality is improved, increases requirements understanding, and facilitates detecting inconsistencies (such as duplicate requirements) and managing requirements changes [10].

Over the past few years, MD has faced lots of challenges arising from their RE process involving low quality of the requirements and duplicate requirements due to constant flow of new requirements. As reported by [1] [25], in most of the companies interviewed, producing a well-formulated requirements posed a great problem since many of the interviewees find it difficult to understand the requirements. Requirements is very critical to software quality and any bug not discovered at this phase will be very costly when discovered at a later phase [16]. The continuous flow of requirements is another serious issue posing a great risk of requirements duplicates which can trigger requirements overload.

With these and more, requirements have to be refactored in order to enhance its clarity, remove duplicates and even facilitates other development decisions such dependencies, priority, release planning, etc. while preserving its behavior. This then calls for urgent need to incorporate refactoring into MDRE processes. Many refactorings and tools for refactoring requirements documents exist but their focal point is on specific techniques like use cases with no descriptions or mechanisms to specify requirements [20]. Other approaches that exist provide no guidelines on how to identify the potential problems. In this work, we propose a generic approach that tends to overcome the existing challenges.

6 Improving Requirements Quality with Refactoring

6.1 Refactoring Opportunities

Like code refactorings, identifying a set of 'smells' that could signify potential refactoring opportunities is a good start in the right direction. In this paper we used the generic approach to identify refactoring opportunities in requirements by Ramos et al [21]. This approach is applicable to any requirements description technique. The refactoring opportunities are not strict rules for automatic refactorings application, rather decisions need to be taken first [21]. Requirements engineer need to make decisions and choose which refactoring is more suitable for each opportunity during the course of application.

Ramos et al [21] identified a collection of refactoring opportunities that can be found in software requirements, corresponding to 'smells' in the specification such as *large requirement*, *complex conditional structure*, *lazy requirement*, *naming problems* and *duplicated activities*. In each opportunity, there is a description of the method to identify occurrences of the problem and the refactorings types that can be used to reduce the effects of the problem occurrences. Accordingly, a collection of refactorings types to manipulate a smelled-requirement are the *extract requirement*, *rename requirement*, *move activity*, *inline requirement*, and *extract alternative flows*. Each refactoring

contains the context, the type of solution it provides, a motivation for the transformations, its mechanics and an example of a refactored description [15][21]. More details about each opportunity and types of refactoring can be found at [21]. These approaches when applied to MDRE process will go a long way improving the quality of the requirements document and rids duplicate requirements.

6.2 Refactoring the Requirements

To perform the actual refactoring after identifying the possible refactoring opportunity, the above mentioned refactorings types can be applied. The description is based on the format recommended by [15] for describing refactorings. That is, each refactoring is described based on the name, context, the solution, the motivation, set of mechanics and an example.

Here we provide one example described by [21] based on the above stated format. Though the example is based on use cases, it can be applied to any requirements description techniques. For instance, to apply *extract requirement*, the following guidelines are necessary:

Name: *Extract Requirement*

Context: Requirement is too large or contains information related to a feature that is scattered across several requirements or is tangled with other concerns.

Solution: Extract the information to a new requirement and name it according to the context.

Motivation: Apply when requirements is too large and can be split into two or more new requirements. This large requirements contain large amount of information that is difficult to understand or ambiguous. This further makes it cumbersome to locate the needed information easily.

Mechanics: The following activities should be performed:

1. Create a new requirement and name it.
2. Select the information you want to extract.
3. Add the selected information to the new requirement.
4. Remove the information from the original requirement.
5. Make sure the original requirement is acceptable without the removed information.
6. Update the references in dependent requirements.

Example: The use case named Complaint Specification dealing with three different types of complaints (animal, food or diverse) is obtained from the Health Watcher system [22].

Main flow of events: This use case makes possible for a citizen to register complaints. Complaints can be Animal Complaint, Food Complaint or Diverse Complaint:

1. The citizen informs the kind of complaint;
2. The system registers the kind, date and time of the attendance;
3. The system shows the specific screen for each type of complaint;
4. The citizen provides the data;
5. The system saves the complaint (with the OPENED state), return a code for the attendance, so that the citizen can take note and query for the situation of his/her complaint.

Figure 1: Use case complaint specification [21][22]

Due to different pre-conditions, data to be manipulated and interfaces arising from each complaint type, each type can be extracted as a separated use case.

Main flow of events:

1. The citizen chooses the kind of complaint;
2. Case the citizen chooses the animal complaint.
 - 2.1. The main flow will follow the one described on [Register Animal Complaint].
3. If the citizen chooses the food complaint.
 - 3.1. The main flow will follow the one described on [Register Food Complaint].
4. If the citizen chooses the diverse complaint.
 - 4.1. The main flow will follow the one described on [Register Diverse Complaint].
5. The system saves the complaint.

Figure 2: Use case complaint specification (after the refactoring) [21]

Main flow of events:

1. The citizen selects the option register animal complaint;
2. The system registers the kind, date and time of the attendance;
3. The system shows the screen for the animal complaint;
4. The citizen provides the complaint description;
5. The system saves the animal complaint (with the OPENED state), return the code for the attendance, so that the citizen can take note and query for the situation of his/her complaint.

Figure 3: Use cases register animal complaint [21]

Details about other refactorings can be found at [21]. With extract requirement as described above, situations like requirements duplication, requirements misunderstanding, etc can be solved and improve the quality of the specification which in turn will enhance the quality of the software.

7 Conclusions

MD software development organization faces many challenges that differ from those found in organizations developing bespoke software. In this paper, we have presented specific challenging issues with regard to RE challenges studies in MD software development conducted in the industry. These challenges are the issues of writing understandable requirements and constant flow of new requirements. The challenges are as a result of the stakeholder's characteristics and the constant flow of new requirements. These however, result in having low quality requirements and duplicate requirements that can affect other development decisions.

To rid MD development of these problems, we have proposed the incorporation of refactoring into its RE processes. To apply refactoring, we proposed the collection of refactoring opportunities that might occur in requirements and provides a collection of refactorings that can be used to improve the quality of requirements where these opportunities appear. This proposal is based on the collections described by Ramos et al. We believe that if refactoring is applied to requirements documents like source codes, the requirements would be made more understandable and improve the overall organization of the project in MD software development. That is, refactoring requirements will go along way enhancing the quality of the design and onward development. For future work, we plan to introduce more refactoring and investigate several issues that may inhibit the actualization of this proposal.

8 References

- [1] Karlsson, L., Dahlstedt, A.G., Natt, J., Regnell, B. and Persson, A.: Requirements engineering challenges in market-driven software development - An interview study with practitioners, *ACM Information and Software Technology*, Vol 49, 2007.
- [2] E. Carmel, S. Becker: A Process Model for Packaged Software Development, *IEEE Transactions on Engineering Management*. Vol. 42, pp. 50–60, 1995
- [3] Sawyer, P., Sommerville, I. and Kotonya, G. Improving Market-Driven RE Processes. *Proc International Conference on Product Focused Software Process Improvement*, 1999
- [4] P. Sawyer, Packaged Software: Challenges for RE, *Proceedings of the Sixth Int. Workshop on Requirements Engineering: Foundations of Software Quality*, Stockholm, Sweden, pp. 137–142, 2000.
- [5] Karlsson, L, et al. Requirements engineering challenges in market-driven software development - An interview study with practitioners. *Information and Software Technology* Vol. 49, 6, pp. 588-604, 2007.
- [6] Natt och Dag, Johan, et al. A Linguistic-Engineering Approach to Large-Scale Requirements Management. *IEEE Software* Vol. 22, pp. 32-39, 2005.
- [7] E.J. Chikofsky and J.H. Cross, "Reverse Engineering and Design Recovery: A Taxonomy," *IEEE Software*, vol. 7, no. 1, pp. 13-17, 1990.
- [8] Natt och Dag, J. Regnell, B.; Carlshamre, P.; Andersson, M.; Karlsson, J.: A feasibility study of Automated Natural Language Requirements Analysis in Market-driven Development, *Requirements Engineering*, pp. 20-33, 2002
- [9] K. N. Reddy and A. A. Rao: A Quantitative Evaluation of Software Quality Enhancement by Refactoring Using Dependency Oriented Complexity Metrics. *IEEE Computer Society*, 2009
- [10] Mens, T. and Tourwe, T.: A Survey of Software Refactoring *IEEE Transactions on Software Engineering*, Vol. 30, No. 2, Feb., 2004
- [11] Harding, J A, et al. An intelligent information framework relating customer requirements and product characteristics. *Computers in Industry*, Vol. 44, pp. 51-65, 2001.
- [12] Sivzvattian, Siv and Nuseibeh, Bashar. Linking the Selection of Requirements to Market Value, 2003
- [13] Host M, Regnell B, Natt och Dag J, Nedstam, J, Nyberg C. Exploring bottlenecks in market-driven requirements management processes with discrete event simulations. In: *Proceedings of the workshop on software process simulation and modeling (PROSIM'2000)*, London, UK, July 2000
- [14] Lippert, M. and Roock, S.: Refactoring in large software projects: performing complex restructurings successfully, *John Wiley & Sons Ltd*, England, 2006
- [15] Fowler, M., Beck, K., Brant, J., Opdyke, W., Roberts, D.: "Refactoring: improving the design of existing code", in: *Object Technology Series*. Addison-Wesley, 2000
- [16] Boehm, B., Sullivan, K.: "Software economics: a roadmap", in: *ICSE – Future of SE Track*. pp.319–343, 2000
- [17] Pressman, R.: "Software Engineering: A Practitioner's Approach", *McGraw-Hill*, 2005
- [18] Wang, Y.: What Motivate Software Engineers to Refactor Source Code? Evidences from Professional Developers, *IEEE*, 2009
- [19] A. Russo, B. Nuseibeh, and J. Kramer, "Restructuring Requirements Specifications for Managing Inconsistency and Change: A Case Study," *Proc. Int'l Conf. Requirements Eng.*, pp. 51-61, 1998

- [20] Xu, J., Yu, W., Rui, K., Butler, G.: "Use case refactoring: a tool and a case study", in: Software Engineering Conference, 2004. 11th Asia-Pacific, pp. 484–491, 2004
- [21] Ramos, R. et al: Improving the Quality of Requirements with Refactoring, VI Simpósio Brasileiro de Qualidade de Software, 2007
- [22] Soares, S., Laureano, E., Borba, P.: "Implementing distribution and persistence aspects with AspectJ", in: Proceedings of the 17th ACM conference on Object oriented programming, systems, languages, and applications, ACM Press 174–190, 2002
- [23] P. Van Gorp, H. Stenten, T. Mens, and S. Demeyer, "Towards Automating Source Consistent UML Refactorings," Proc. Unified Modeling Language Conf., 2003
- [24] W.F. Opdyke, "Refactoring: A Program Restructuring Aid in Designing Object-Oriented Application Frameworks," PhD thesis, Univ. of Illinois at Urbana-Champaign, 1992.
- [25] R. La`mmel, "Towards Generic Refactoring," Proc. SIGPLAN Workshop Rule-Based Programming, 2002
- [26] Karlsson, L., Dahlstedt, A.G., Natt, J., Regnell, B. and Persson, A.: Challenges in Market-Driven Requirements Engineering - an Industrial Interview Study, Proceedings of 8th International Workshop on Requirements Engineering: Foundation for Software Quality, 2002
- [27] Tahvildari, L. and Kontogiannis, K. "A metric-Based Approach to Enhance Design Quality Through Meta-Pattern Transformations", Proc. of the Seventh IEEE European Conf. on Software Maintenance and Reeng. (CSMR'03), pp. 183-192, 2003

Workflow map optimization by using multiobjective algorithms

L. Osuszek

Institute of Informatica, Silesian University, Sosnowiec, Poland

Abstract - This paper covers extending Process Definition Language (XPDL), Workflow map and application of multi-objective optimization algorithms to enable fully automated optimization of the Workflow map. The mathematical model of the business process may be subject to specific multi-criteria optimization algorithms.

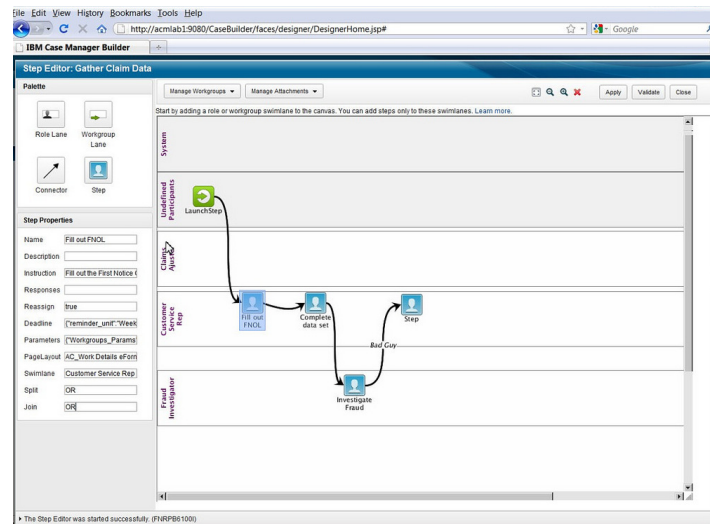
Keywords: P8 BPM, Business Process Management, XPDL, Multiobjective Optimization, Non-Dominated Sorting Genetic Algorithm II (NSGA2), Strength Pareto Evolutionary Algorithm II (SPEA2) and Multi-Objective Particle Swarm Optimization (MOPSO), IBM Case Manager, ICM, Business Process Manager, BPM.

1 Introduction

In the previous article we introduced BPM workflow improvement by adopting code optimization techniques. In this article we look at multi-objective workflow optimization and tangible values it might brought to Business Process Management.

Today, there are several techniques for modeling business processes which enable mapping existing processes of the organization, and allow for creating new ones in order to meet growing market demand. One of the process maps - created in IBM Case Manager (ICM) - is rendered below in figure 1. For each company and organization, business processes and the related decisions are the key element which provides the momentum for their operations and determine their competitiveness. The management of workflow and information within process paths has a major impact on the speed, flexibility and quality of decision-making processes. This is why the acceleration and optimization of processes is decisive for the success of any organization.

Figure 1. Business process map in IBM Case Manager



Processes involve people, systems and information. The maximum efficiency is possible only if all of these elements interoperate in an automated environment. Note also that optimized processes enable a faster response to the changing market situation and to new customers' demands while guaranteeing compliance with applicable regulations. In short, better processes contribute towards continuous improvement of the efficiency of company's operations, and therefore, allow gaining competitive advantage in the industry.

One of the factors which make the Business Process Management systems increasingly popular is the tracking, analysis and simulation of processes. With the monitoring of work progress, with in-depth analysis of current and historical processes, and with the verification of changes to processes prior to their implementation in a production system, these tools guarantee more accurate business decisions. Additionally, they enable fast implementation of best business practices, and reduce the total cost of system ownership with reusable process definitions.

The aforementioned advantages of BPM are quite widely spoken of, but not enough attention is paid to the optimization problem. Today's tools offer the functionality of business process optimization. However, expert knowledge is required to use them efficiently. With their experience backed up by software-based simulations, the consultants who operate such

tools are able to specify the way of optimization of the process concerned. The weakness of the BPM description language is the inadequate description model. Only few description models of business processes enable the use of process map optimization methods and analyses to improve the timeliness (accelerate), to ensure the optimum use of resources, or to save money.

The objective of this paper is to show the ability to extend the business process description model in order to enable a more sophisticated, multi-objective optimization.

Idea of multi-objective optimization could be easily adapted to real uses cases by combination of BVA (Business Value Assessment) and mathematical models. BVA may be used to perform an in-depth cost analysis of each business process component. Such a descriptive model provides additional analysis and optimization possibilities. This data could be easily used to mathematical optimization algorithms. It is a powerful tool that can suggest how to restructure the company or customize the existing business process so as to make it more optimal.

2 Current fields of studies

Several models have been developed to enable description of business processes using mathematical models. The most popular include Integration Definition (IDEF), Computer Integrated Manufacturing – Open System Architecture (CIM-OSA), Object-Oriented Modeling, and the highly popular Petri Nets. These standards were used to develop many tools for business process modeling (ARIS, FirstStep, PrimeObject, etc.). Zakarian [1] integrated the Fuzzy-rule-based Reasoning Approach with IDEF in order to extend the quantitative analysis of the process model. Grigori [2] proposed the Business Process Intelligence, a tool which uses data mining methods for the analysis of business processes.

Multiple algorithms were developed to enable optimization of business problems in the area of logistics Yu and Li [3], as most of business models (including Business Process Modeling Notation) are insufficient from the point of view of the multi-criteria analysis. McKay and Radnor [4] presented a model for the description of business processes which, however, did not include any formal optimization methods.

Most scientific studies on business process optimization focus on selecting the appropriate process model, or on one-dimensional optimization (Hofacker and Vetschera [5]) which is unsatisfactory..

2.1 Multi-objective optimization (multi-criteria)

Generally, there are the following optimization types:

- single-criterion optimization: if the ideal state is required to be reached for a single evaluation criterion;
- multi-criteria optimization (vector optimization, poly-optimization): if reaching the ideal state depends on multiple evaluation criteria.

A large number of criteria for the evaluation of the ideal state often results in contradictions between them. This means that the solution looked for does not reach the extreme values of all criteria considered separately. Instead, it provides some kind of compromise between them. Therefore, the poly-optimization problem consists primarily in defining that compromise. In many cases, the heuristic knowledge about the optimized process allows for specifying another, substitute criterion for searching the compromise solution.

In formal terms, poly-optimization may be specified as follows:

Let $\mathbf{X} = \{x_l\}, l = 1, 2, \dots, N$ be a vector of decision variables considered as independent. Let $\mathbf{F} = \{f_i\}, i = 1, 2, \dots, M$ be a set of criteria (functions) for evaluating solutions when looking for the compromise. Let the following restrictions be imposed on the values of the solutions:

- inequality restrictions: $\mathbf{G} = \{g_k\}, k = 1, 2, \dots, K$, with: $g_k(\mathbf{X}) \leq 0$;
- equality restrictions: $\mathbf{H} = \{h_j\}, j = 1, 2, \dots, J$, with: $h_j(\mathbf{X}) = 0$;

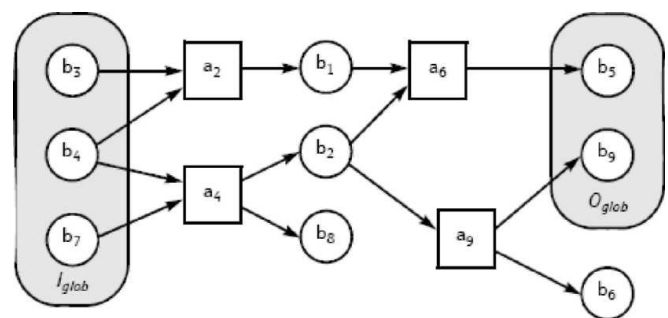
The objective of poly-optimization is to reach a solution which meets the following condition:

$$\min \mathbf{F}(\mathbf{X}) = \{f_1(\mathbf{X}), f_2(\mathbf{X}), \dots, f_l(\mathbf{X})\}$$

If maximization of an f_l * function is required, an auxiliary criterion may be introduced in accordance with the following formula:

$$\min f_l(\mathbf{X}) = -\max f_l^*(-\mathbf{X})$$

Figure 2. Business process design with activities and resources



In most cases, the business process description model includes activities and resources (figure 2). The activities are supposed to enable meeting the objective of the business process. The two sets of resources showed on the figure (Iglob and Oglob) are, respectively, the initiating resources available at the beginning of the process, and the output resources resulting from the performance of the process.

There are two categories of resources 'flowing' through the entire map of the business process: physical resources (e.g., process participants) and information resources.

Business process optimization involves specification of the criteria to be optimized. Usually, these will be costs and process duration.

The literature provides numerous examples of ready-to-use algorithms for multi-criteria optimization of various types of problems. These algorithms may use any criteria through the application of the corresponding mathematical model. The unambiguous conclusion from the analysis of existing multi-criteria optimization algorithms is that the model for the XPDL description of business processes is simple and may be extended with additional information enabling the construction of a more accurate mathematical model. This conclusion is applicable to the XPDL supported by P8 BPM, because as most BPM models, the XPDL model can be represented in BPMN. Let us consider the following example of a business process:

Figure 3. Travel Agency process of holidays offering

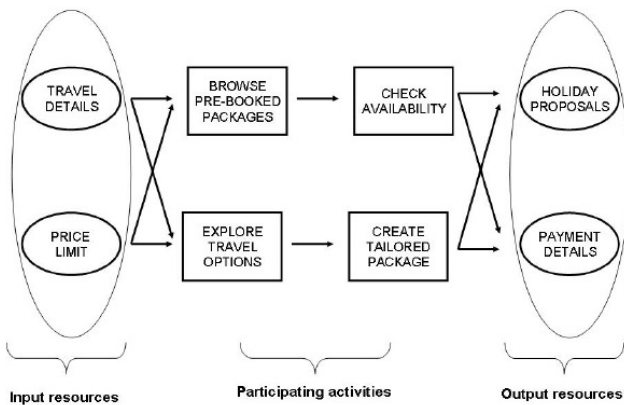


Figure 3 renders generic model of the operation of a virtual travel agency. The input (initiating) data is the customer's guidelines as to the details of a trip, and the maximum price the customer may pay. Then, the travel agent performs a series of interrelated actions (activities) which compose a business process aimed at the delivery of a proposal compliant with the expectations.

Let us assume extension of the standard BPM model describing the business process with additional information on the costs and duration of each action (business step). Additionally, detailed possible actions are specified for each activity. A similar approach was included in IBM Case Manager, the latest tool for describing dynamic business processes. Table 1 describes the business process of finding the appropriate trip proposal.

Table 1. Process map as a set of process elements, cost and duration

Object name	Process element	Alternatives	Cost	Duration
Travel details	Input resource	-	-	-
Price limit	Input resource	-	-	-
Browse pre-booked packages	Activity	1. Search from brochures 2. Search company intranet	2 7	9 5
Explore travel options	Activity	1. Browse past cases 2. Explore new options	4 6	8 6
Check availability	Activity	1. Via intranet/e-mail 2. Via phone/post	10 5	1 7
Create tailored package	Activity	1. Use specific software 2. Combine options manually	11 5	2 6
Holiday proposals	Output resource	-	-	-
Payment details	Output resource	-	-	-

The customer who enters the travel agency reports his/her request to prepare a proposal of holiday in a specific localization. He/she also specifies the price limit to be complied with. To reply, the travel agent may browse through the previously prepared trip packages, or search through the entire proposal database in order to prepare a customized offer. There are two ways (activities) to browse through previously prepared packages: checking the information brochures or searching through the company's intranet database. If the agent decides to perform a more in-depth exploration of the proposal database, he/she may check the past cases or verify new options. Once the details of the choice of the trip are determined, the agent checks the availability of the proposal (through intranet/e-mail or phone/mail). The last step of the proposal construction process is the presentation of a customized trip package to the customer. To do so, the agent may use dedicated tools, or he may create the proposal manually. Each of the selected actions involves the corresponding cost and duration of the activity.

Description of the business process extended in that manner enables creation of a mathematical model which may be optimized with known multi-criteria optimization algorithms. The use of Non-Dominated Sorting Genetic Algorithm II (NSGA2), Strength Pareto Evolutionary Algorithm II (SPEA2) or Multi-Objective Particle Swarm Optimization (MOPSO) allows looking for process map variants optimized for the selected criteria. The process diagrams below present the paths optimized for the duration or for the costs generated

by the process. In this experiment SPEA2 algorithm was choose as a core of multi-objective optimization.

The Strength Pareto Evolutionary Algorithm (SPEA) [6] is a relatively recent technique for finding or approximating the Pareto-optimal set for multi-objective optimization problems. SPEA has shown very good performance in comparison to other multi-objective evolutionary algorithms [7], and therefore it has been a point of reference in various recent investigations. In this experiment, an improved version, namely SPEA2, is applied, which incorporates in contrast to its predecessor a fine-grained fitness assignment strategy, a density estimation technique, and an enhanced archive truncation method.

For the purposes of this experiment, a dedicated application was developed which optimizes the process map in function of the selected criterion by using SPEA2 algorithm. If the process performance time is specified as the optimization criterion (Figure 5), the application analyzes possible combinations of activities in order to determine the shortest delivery path for the entire process. If the main criterion is cost (Figure 4) algorithm build the cheapest process map.

Figure 4. Workflow path optimized for cost criterion.

```

Plik  Edycja  Format  Widok  Pomoc
FINAL COST = 16
FINAL TIME = 16
SOLUTION :
step name = Start
realization name = Start
step name = Browse_prebooked_packages
realization name = Search_from_brochures
step name = Explore_travel_options
realization name = Browse_past_cases
step name = Check_availability
realization name = via_phonepost
step name = Create_tailored_package
realization name = Combine_options_manually
step name = Stop
realization name = Stop
    
```

Figure 5. Workflow path optimized for time criterion.

```

Plik  Edycja  Format  Widok  Pomoc
FINAL COST = 27
FINAL TIME = 10
SOLUTION :
step name = Start
realization name = Start
step name = Browse_prebooked_packages
realization name = Search_from_brochures
step name = Explore_travel_options
realization name = Browse_past_cases
step name = Check_availability
realization name = via_intranetemail
step name = Create_tailored_package
realization name = Use_specific_software
step name = Stop
realization name = Stop
    
```

For that purpose, we create an appropriate text file which includes the process map description:

Figure 6. Description of process map

```

Plik  Edycja  Format  Widok  Pomoc
Start - Start 0 0
Browse_prebooked_packages - Search_from_brochures 2 9 ; Search_company_intranet 7 5
Explore_travel_options - Browse_past_cases 4 8 ; Explore_new_options 6 6
Check_availability - via_intranetemail 10 1 ; via_phonepost 5 7
Create_tailored_package - Use_specific_software 11 2 ; Combine_options_manually 5 6
Stop - Stop 0 0
Connections
0 1 ; 0 2 ; 1 3 ; 2 4 ; 3 5 ; 4 5
    
```

With a proper structure which reflects business process:

```

zero - a 8 8 ; b 7 7
one - c 1 2 ; d 3 3
two - e 5 5 ; f 9 9
Connections
0 1 ; 1 2
    
```

The name of the object, and the possible alternatives of activities together with the information on the costs and duration. The tool provides an optimized process path for the selected criterion (or a conjunction of criteria).

2.2 The SPEA2 Algorithm

SPEA2 was designed to overcome the aforementioned problems. The overall algorithm is as follows:

Algorithm 1 (SPEA2 Main Loop)

Input: N (population size)
 \bar{N} (archive size)
 T (maximum number of generations)
 Output: A (non-dominated set)

Step 1: Initialization: Generate an initial population P_0 and create the empty archive
 (external set) $\bar{P}_0 = \emptyset$. Set $t = 0$.

Step 2: Fitness assignment: Calculate fitness values of individuals in P_t and \bar{P}_t

Step 3: Environmental selection: Copy all non-dominated individuals in P_t and \bar{P}_t to \bar{P}_{t+1} .

If size of \bar{P}_{t+1} exceeds \bar{N} then reduce \bar{P}_{t+1} by means of the truncation operator, otherwise if size of \bar{P}_{t+1} is less than \bar{N} then fill \bar{P}_{t+1} with dominated individuals in P_t and \bar{P}_t

Step 4: Termination: If $t \geq T$ or another stopping criterion is satisfied then set A to the set of decision vectors represented by the non-dominated individuals in \bar{P}_{t+1} . Stop.

Step 5: Mating selection: Perform binary tournament selection with replacement on \bar{P}_{t+1} in order to fill the mating pool

Step 6: Variation: Apply recombination and mutation operators to the mating pool and set \bar{P}_{t+1} to the resulting

population. Increment generation counter ($t = t + 1$) and go to Step 2.

In contrast to SPEA, SPEA2 uses a fine-grained fitness assignment strategy which incorporates density information. Furthermore, the archive size is fixed, i.e., whenever the number of non-dominated individuals is less than the predefined archive size, the archive is filled up by dominated individuals; with SPEA, the archive size may vary over time. In addition, the clustering technique, which is invoked when the non-dominated front exceeds the archive limit, has been replaced by an alternative truncation method which has similar features but does not lose boundary points. Finally, another difference to SPEA is that only members of the archive participate in the mating selection process

3 Conclusions

The approach aimed at expanding the business process description model focuses on the mathematical analysis of the business. The traditional approach towards business process optimization, offered by various IBM tools (e.g., Process Analyzer), focuses more on the aspect of actors (participants) of the process. This allows the experts to quickly find the system steps that may be performed in parallel, or to avoid any dead ends or redundant iterations. However, the proper use of such types of tools requires expert knowledge and methodic experience in the development of business paths. The extension of the BPM for business process description, as proposed in this article, enables a fully automated optimization of the business process. The mathematical model of the business process may be subject to a specific multi-criteria analysis in order to determine the optimum process paths for the selected criterion.

Real life uses cases could be optimized by an in-depth analysis of individual tasks (time consumption, costs) comprising business processes makes it possible to identify those areas which may generate savings upon modification. Modification could be prompted by adopting multi-objective algorithms like SPEA2.

Reader could also be interesting in further article. Next part introduces conversion of XPDL workflows into Petri Network Modeling Notation for optimization in category of time consumption.

4 References

- [1] Zakarian A., 2001. Analysis of process models: A fuzzy logic approach. *The International Journal of Advanced Manufacturing Technology* 17, 444-452.
- [2] Grigori D., Casati F., Castellanos M., Dayal U., Sayal M. and Shan M.C., 2004. Business Process Intelligence. *Computers in Industry* 53, 321-343.
- [3] Yu C-S. and Li H-L., 2000. A robust optimization model for stochastic logistic problems. *International Journal of Production Economics* 64, 385-397.
- [4] McKay A. and Radnor Z., 1998. A characterization of a business process. *The International Journal of Operations and Production Management* 18 (9/10), 924-936.
- [5] Hofacker I. and Vetschera R., 2001. Algorithmical approaches to business process design. *Computers & Operations Research* 28, 1253-1275.
- [6] Zitzler, E. (1999). *Evolutionary Algorithms for Multiobjective Optimization: Methods and Applications*. Ph. D. thesis, Swiss Federal Institute of Technology (ETH) Zurich, Switzerland. TIK-Schriftenreihe Nr. 30, Diss ETH No. 13398, Shaker Verlag, Aachen, Germany
- [7] Zitzler, E., K. Deb, and L. Thiele (1999, December). Comparison of multiobjective evolutionary algorithms: Empirical results (revised version). Technical Report 70, Computer Engineering and Networks Laboratory (TIK), Swiss Federal Institute of Technology (ETH) Zurich, Gloriastrasse 35, CH-8092 Zurich, Switzerland.
- [8] K. Vergidis, A. Tiwari, B. Majeed *Optimisation of Business Process Designs: An algorithmic approach with multiple objectives.*
- [9] J. M. Pinto and I. E. Grossmann, "Assignment and sequencing models for the scheduling of process systems," *Ann. Oper. Res.*, vol. 81, pp. 433–466, 1998
- [10] F. Soliman, "Optimum level of process mapping and least cost business process re-engineering," *Int. J. Oper. Prod. Manage.*, vol. 18, no. 9/10, pp. 810–816, 1998
- [11] Tiwari, K. Vergidis, and B. Majeed, "Evolutionary multi-objective optimisation of business processes," in *Proc. IEEE Congr. Evol. Comput.*, Jul. 2006, pp. 3091–3097.
- [12] W. M. P. van der Aalst, A. H. M. ter Hofstede, and M. Weske, "Business process management: A survey," in *Lecture Notes Computer Sciences*, Springer-Verlag, 2003, vol. 2678, pp. 1–12.
- [13] WIL M. P. VAN DER AALST ET AL, *Pattern-Based Analysis of BPML and WSCI*, 2004

SESSION

SOFTWARE MAINTENANCE, REUSE, DEVELOPMENT CONCEPTS + COMPILER TECHNOLOGIES

Chair(s)

TBA

Documenting Java Database Access with Type Annotations

Paul L. Bergstein

Dept. of Computer and Information Science, University of Massachusetts Dartmouth
285 Old Westport Rd., Dartmouth, MA 02747
pbergstein@umassd.edu

Abstract – *Enterprise applications typically include a relational database layer. Unfortunately, the current generation of IDE's (Integrated Development Environments) do not adequately capture the interaction between the database management system and other layers of the application. For example, current Java IDE's do not evaluate the relationship of classes with the database, or how a particular java method interacts with database tables and columns. We report here our recent progress in extending an Eclipse plug-in to use programmer supplied documentation of database access in the form of type annotations for providing a visual map of interactions between Java code and relational databases. A primary motivation is to facilitate code maintenance in the face of database modifications.*

Keywords: Software maintenance, software visualization tools, java annotations.

1 Introduction

Modern tools have simplified the development of enterprise applications. However, the current generation of IDE (Integrated Development Environment) does not sufficiently capture the interaction between application code and the database layer. For example, the Eclipse IDE does not evaluate how a particular java method interacts with database tables and columns. This makes it difficult to maintain and enhance applications, since a change in code may create problems that will become known only after extensive testing. We report here our progress in developing an Eclipse plug-in that helps the programmer by providing a visual map of interactions between Java code and relational databases.

The obvious benefit of the mapping is to facilitate code maintenance in the face of database modifications by identifying the code-to-database couplings. A second, and equally important, benefit involves the easy detection of code-to-code couplings that arise when different java methods access the same database elements. For example,

suppose the programmer wants to make some changes to a method and would like to know the effects of this change on rest of the code. Ordinarily the programmer could use the “Call Hierarchy” feature of the Eclipse IDE to get the dependencies of other methods and classes on this method. But suppose the method uses an SQL statement to store a string in the *address* column of the *customer* table, and the developer wants to change the format of the address. This is not easy because there may be many other methods which are dependent on the address format but are not related through the call hierarchy.

2 Background

We have previously reported [1,2] our development of a tool to provide a visual mapping of java code-to-database and code-to-code (via database) couplings. We have implemented our tool as a fully integrated plug-in to the popular Eclipse development environment. Developers can view the database and the project at various levels of granularity and easily find the types of coupling they are most interested in. For example, users can choose to view couplings of code to anywhere in the database, to a particular table in the database, or to a specific column in a table. Similarly, they can adjust the granularity of their project view between the project, class, and method levels. Search facilities enable users to quickly identify important dependencies. For example, when a method that stores information in the database is modified, it is easy to find all of the methods (or classes or projects) that retrieve the same information and might be affected.

Until now, the database access of an application was discovered by our tool through a combination of static and dynamic code analysis. Each of these two approaches has its relative advantages and disadvantages, but share some common characteristics. Both approaches have the advantage of automatic discovery that makes them suitable for

maintaining legacy applications where the database access is not well documented. However, neither is 100% effective in finding all possible interactions between the application and the database.

The static code analyzer uses the Sun java compiler API and the Compiler Tree API to parse the java source and walk the abstract syntax tree. It looks for string literals that are included either directly or after assignment to String variables in calls to the *execute*, *executeQuery*, and *executeUpdate* methods of the *JDBC Statement* class. The analyzer attempts to identify column and table names occurring in select, from and where clauses and record these dependencies in the coupling repository.

The code analyzer considers certain string concatenations including some concatenations that are built from a combination of string literals and variables to detect simple cases of dynamic SQL generation in the code. However, static analysis in general is a hard problem and it will never be possible to detect all couplings to the database that may occur at runtime, possibly dependent on user input.

The main component of the dynamic analyzer is a JDBC bridge driver that logs the database accesses to the coupling data repository. Our driver acts as a bridge between the application and the "real" driver that communicates with the user's database. The implementation is conceptually simple. Most of the methods in our driver classes simply pass requests on to the underlying "real" driver and return whatever data is returned from the real driver. The main exception is in the Statement class methods (e.g. *execute*, *executeQuery*, *executeUpdate*) that take SQL statements as arguments. These methods receive only complete, valid, fully formed SQL statements as arguments (unless there are errors in the application) even if they have been built dynamically. The SQL statements processed in the JDBC driver are parsed to determine the database elements that are being accessed and the coupling information is recorded in the repository.

The main drawback to the dynamic analysis approach is that it will only find database accesses that occur during testing with the bridge driver in place. Therefore the success of this technique is highly dependent on the developer's ability to generate adequate test cases.

In this paper, we describe an extension of our previous work to allow the developer to explicitly supply the database access information in the form of type annotations, as detailed in section 3. For new projects, this can overcome the limitations of code analysis techniques. In legacy projects, or when the developer hasn't completely documented the database access, code analysis can still be used to supplement this information. In a future enhancement, access information that the visualization tool obtains through static and dynamic code analysis may be injected into the source code in the form of annotations to automatically document the code.

3 Results

Database access information is stored in a repository that is used internally by our tool. For every code-to-database coupling that is detected by either the static or dynamic code analyzer, there is an entry in the repository. Each coupling entry in the repository includes the code location (class, method, file, and line number), the database element (database, table, and column), the SQL statement type (select, insert, update, etc.) and the type of access (read, write, or read/write). The statement type does not necessarily determine the access type. For example, a field occurring in the *set* clause of an update statement indicates a write access, but a field occurring in the *where* clause of the same statement indicates a read access.

In order to detect changes over time, the repository also records the first time and last time that a coupling is detected. Also, each time the tool is run, the structure of the database is checked using the JDBC metadata API, and any structural changes are recorded in the repository.

Ideally, we would like to document individual statements where database access occurs. However, since there is currently no support for statement level java annotations or javadoc comments, we utilize type annotations for the documentation. Type annotations are a backward-compatible extension of java annotations defined in the Java Specification Request, JSR 308 [3]. Type annotations allow the programmer to annotate types wherever they occur and are scheduled to be included in the release of JDK 8. They are supported now by the Checker Framework's [4] type annotation compiler.

We have defined type annotations to document the way in which java applications use `Statement` and `PreparedStatement` objects to access database elements. Each `Statement` or `PreparedStatement` variable should be annotated to document the kind of SQL statement that is to be executed (select, insert, update, etc.), the database elements being accessed (database, table, and column) and the type of access (read, write, or update). The tool will associate this information with the lines of code where these variables are used to invoke `execute`, `executeQuery`, and `executeUpdate` methods.

Note that a given statement object can be accessed through multiple reference variables each with its own annotation in case the programmer wishes to reuse the same statement object for different types of database access.

The `Database` annotation definition given in Listing 1, uses Strings to specify the database and SQL statement type. Because types in annotation definitions are restricted to primitives, String, Class, enums, annotations, and arrays of those types, we have defined a second custom annotation, `DataAccess`. Each `Database` annotation contains an array of `DataAccess` annotations to specify the types of access to specific data elements. Notice that the definitions specify a default statement type of `select`, and a default access type of `read`, so that those values can be omitted where appropriate. Also, the wildcard, `*`, is the default value for the column list.

Listing 2 is a code fragment showing examples of how the annotations are used. On lines 2-3, we obtain a `Connection` from the `DriverManager`, and use it on line 11 to get a `Statement` object, `stmt1`. Lines 5-10 contain the `Database` annotation that indicates this statement variable will be used to read the `eid`, `salary`, and `dept` columns of the employee table from the personnel database in a select statement. Our tool will map this database access to line 14 where the `stmt1` variable is used to invoke the `executeQuery` method of the `Statement` class.

Lines 15-27 demonstrate how the same `Statement` object can be reused for a different type of database access, which is also correctly annotated. On line 23, we declare a new variable, `stmt2`, that refers to the same object as `stmt1`. The `stmt2` variable is annotated on lines 16-22 to indicate that it will be used to read the `salary` and `dept` columns, and update the `salary` column of the employee table in

the personnel database. Since there are two types of access to the employee table (read and update), there are two `DataAccess` annotations within the `Database` annotation. Both of these database accesses will be mapped by our tool to line 26 where `stmt2` is used to invoke the `executeUpdate` method.

Figure 1 shows the overall architecture of our tool. The tool uses developer supplied annotations in addition to static and dynamic analysis of the java code to find database couplings. The results of all analysis methods are combined in the coupling data repository which is also used to track changes over time. The user interface, implemented as an Eclipse plug-in displays the results to the user and allows easy navigation to code based on its database coupling.

The screenshot in Figure 2 shows the tool interface in an Eclipse pane. On the Database View tab the database structure is shown as a tree with database, table, and column information arranged in a hierarchal structure. Selecting an element from this tree brings the associated couplings into view along with the controls to select sorting options. In the example in Figure 2, the tree is collapsed to a single node (which is selected), and all couplings to the database are displayed. Selecting a code reference from the coupling list brings the corresponding java source into view in an editor pane with the appropriate line of code highlighted.

When two or more methods are coupled to the same database element, there is a suggestion that these methods may be coupled through the database. The nature of the coupling can be seen from the statement type and access type information. For example, one method might read data that is written to the database by another method.

```
public @interface Database {
    String db();
    String statement() default "select";
    DataAccess[] access();
}

public @interface DataAccess {
    String type() default "read";
    String table();
    String[] columns() default {"*"};
}
```

Listing 1.

```

1. // Open a DB Connection
2. Connection dbConnection =
3.     DriverManager.getConnection(url,username,password);

4. // Create an annotated Statement
5. @Database (
6.     db = "personnel",
7.     statement = "select",
8.     access = {
9.         @DataAccess (table="employee", columns={"eid", "salary", "dept"})
10.    })
11. Statement stmt1 = dbConnection.createStatement();

12. // Execute a query to get the eid, salary, and dept of each employee
13. ResultSet results =
14.     stmt1.executeQuery ("select eid, salary, dept from empolyee");

15. // Annotate a variable for an update reusing the same Statement object
16. @Database (
17.     db = "personnel",
18.     statement = "update",
19.     access = {
20.         @DataAccess (table="employee", columns={"dept","salary"}),
21.         @DataAccess (type="update", table="employee", columns={"salary"})
22.    })
23. Statement stmt2 = stmt1;

24. // Execute a query to give sales department employees a 10% raise
25. ResultSet results =
26.     stmt2.executeUpdate (
27.         "update employee set salary = salary*1.1 where dept = 'sales'");

```

Listing 2.

The interface also allows the user to browse in the opposite direction, i.e. from code to database. The project view tab provides a hierarchal view of the projects where the user can select a project, class, or method to find the database elements that it accesses. A developer can find methods that are coupled through the database to a method that has been added or modified by using both views. First, the database elements accessed by the new or modified method can be found in the project view tab. Then, other methods that access the same database elements can be found in the database view tab. In future versions, we plan to improve the interface to automate this task.

4 Related Work

Java annotations have been widely used for object relational mapping (ORM) in order to automate the process of persisting java objects in relational databases. In this case, classes and fields are annotated to indicate the corresponding tables and columns in a relational database. This is the approach used in the Java Persistence API [5] and in products like Hibernate[6] and ORMLite [7].

There is also a large body of work on software visualization [8-12] and on database visualization. There is also a good deal of work on reverse engineering of databases and CASE tools that support reverse engineering with visualization techniques. However, we are not aware of any other

system designed to support the development and maintenance of software through the visualization of program code dependencies on the database.

5 Limitations

A significant limitation of the work described in this paper is the current lack of support for type annotations in Java 7. This means that code written with type annotations must be compiled with a non-standard compiler such as the Type Annotations compiler that is part of the Checker Framework. Alternatively, the annotations can be written in comments. In this case, the Type Annotations compiler will still recognize them, and the code can also be compiled with any standard compiler without modification. This limitation will disappear with the release of Java 8, assuming that the planned support for type annotations is not dropped before the final release.

Another limitation is that the type annotations are quite verbose, so that it may be difficult to convince developers that it is worthwhile to take the time to include them during development. This

problem will be largely mitigated when we have incorporated functionality to automatically inject annotations into the source code based on data access patterns that are detected by the static and dynamic code analyzers. When this is complete, developers should only need to add annotations for those few accesses that are missed by the code analyzers, or make minor corrections to the ones that have been generated automatically.

At first, we considered it a disadvantage that we were not able to annotate JDBC *execute* statements directly and were forced to annotate the Statement variables instead. While this has made the implementation of the annotation processing slightly more difficult, our experience has shown that this is actually an advantage for users since it requires fewer annotations to be written. In cases where the same type of database access occurs in multiple places within the same method or class, we can annotate a Statement variable once, and use it for each access. This is another mitigating factor in the verbosity problem.

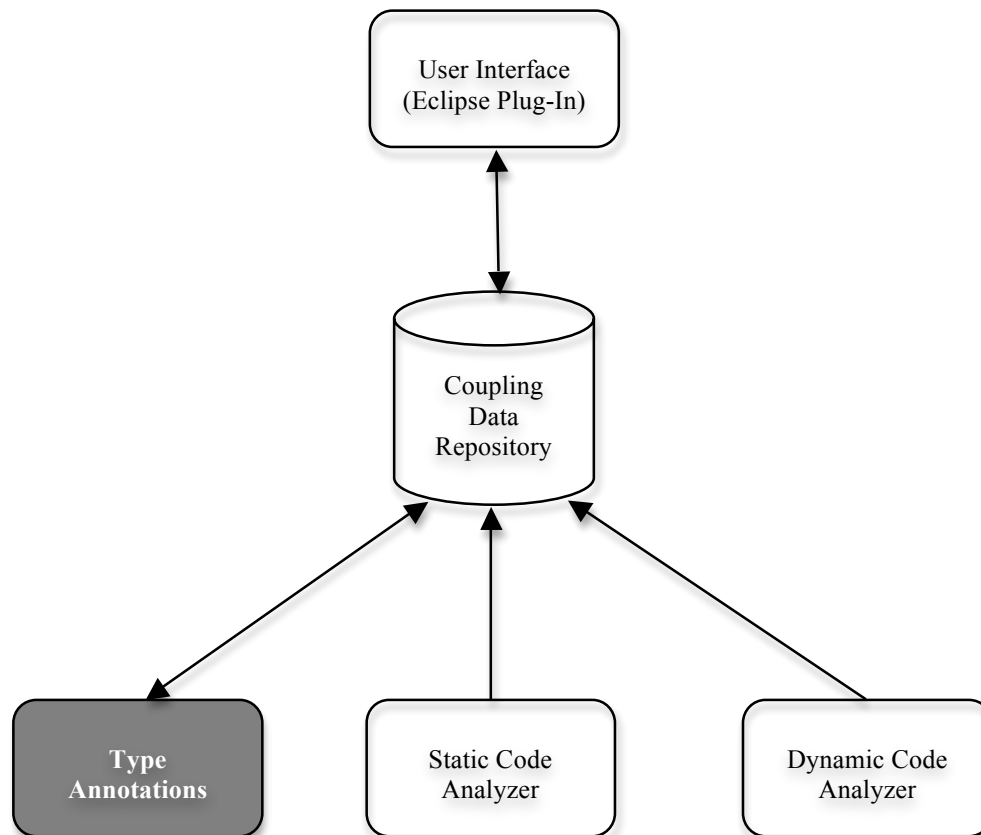


Figure 1.

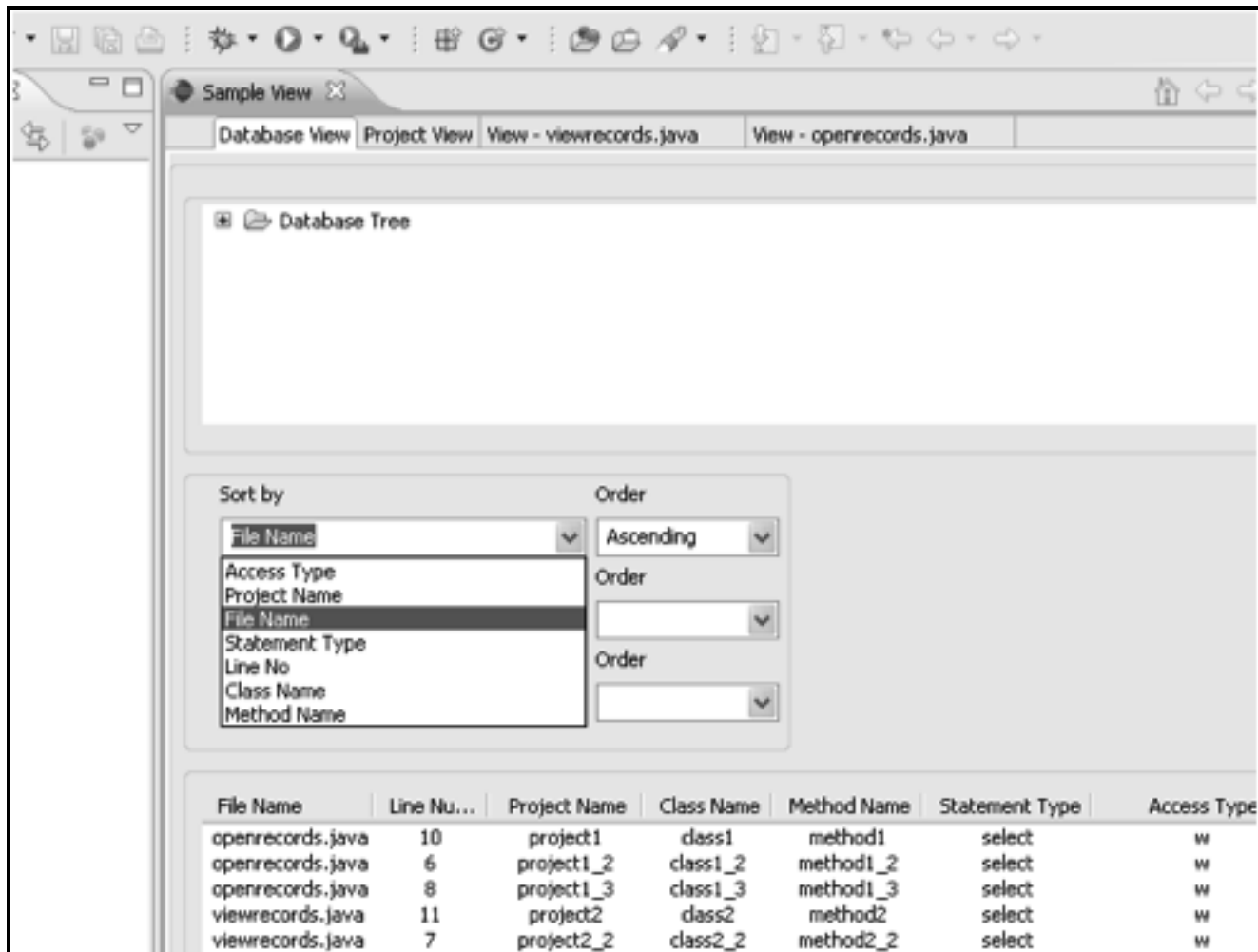


Figure 2.

6 Conclusions and Future Work

Many researchers have investigated to resolve the dependencies between different technologies involved in an enterprise application. Our tool significantly enhances visibility between java and relational databases. The principal benefit is the ability to easily detect the code-to-database couplings and couplings of code-to-code via the database. This ability makes it easy to maintain application code in the face of structural changes to the database, or changes in the format of data stored in the database.

Static and dynamic analysis of java code to discover database couplings each have their advantages and disadvantages. Dynamic analysis is easier to implement and will find all couplings that occur during

testing, but will not find couplings that aren't covered by the test cases. Static analysis may identify couplings that are missed during the testing phase, but is harder to implement and cannot identify couplings that only occur dynamically (e.g. based on user input).

Explicit documentation of code to database couplings by the developer in the form of type annotations is intended to fill any gaps left after static and dynamic code analysis. By combining all of the coupling data in a repository, we get the combined benefits of each. The repository also allows for tracking of changes over time so that areas of code that may be affected by a change could be flagged for the developer. While we are focused on using the coupling information in our visualization tool, we expect that having this kind documentation included in the source

code (whether hand written or injected by the code analysis tools) will prove useful for a variety of reasons during code maintenance.

Our top priority for future work is to implement injection of annotations into the source code from the code analysis tools. However, we are also actively working on improvements to the user interface based on user feedback. In particular, our users are most interested in automating the discovery of code-to-code couplings through the database, and in automatic flagging of potentially affected code when structural changes to the database occur. We are also working on improving the static code analyzer to reduce the number of couplings that are only detected dynamically.

In the long term, we plan to extend this tool to handle additional languages and technologies. For example, we plan to extend our java code analyzers to support JSP by analyzing the java snippets embedded in JSP pages, so that we can show couplings of JSP pages to the database. This would also allow the visualization of couplings between the presentation layer (JSP) and business logic code that occur through the database in a typical J2EE environment. If all these dependencies between the various layers of a J2EE application can be shown through a visual tool, the task of maintaining and enhancing such applications would be greatly facilitated. Eventually, we would also like to support additional programming languages such as C# and C++ and add support for ODBC applications.

7 References

- [1] Paul L. Bergstein, Priyanka Gariba, Vaibhavi Pisolkar, and Sheetal Subbanwad. An Eclipse Plug-In for Visualizing Java Code Dependencies on Relational Databases. In *Proceedings of the 2009 International Conference on Software Engineering Research and Practice (SERP'09)*, Pages 64-69, July 13-16, 2009, Las Vegas, Nevada. CSREA Press ISBN 1-60132-129-5.
- [2] Paul L. Bergstein and Ashwin Buchipudi. Coupling Detection to Facilitate Maintenance of Database Applications. In *Proceedings of the 2011 International Conference on Software Engineering Research and Practice (SERP'11)*, Pages 289-94, July 18-21, 2011, Las Vegas, Nevada. CSREA Press ISBN 1-60132-201-1.
- [3] JSR 308
http://download.oracle.com/otn-pub/jcp/annotations-2012_01_23-edr2-spec/annotations-2012_01_23-edr2-spec.pdf
- [4] Checker Framework
<http://types.cs.washington.edu/checker-framework/>
- [5] Java Persistence API
<http://docs.oracle.com/javaee/6/api/javax/persistence/package-summary.html>
- [6] Hibernate
<http://www.hibernate.org/>
- [7] ORMLite
<http://ormlite.com/>
- [8] G. C. Roman and K. C. Cox. A taxonomy of program visualization systems. *IEEE Computer*, Vol. 26(12), Pages 11-24, 1993.
- [9] Blaine A. Price, Ian S. Small, and Ronald M. Baecker. A Principled Taxonomy of Software Visualization. *Journal of Visual Languages and Computing*, Vol. 4, Pages 211-266, 1993.
- [10] Jonathan I. Maletic, Andrian Marcus, and Michael L. Collard. A task oriented view of software visualization. In *Proceedings of the First International Workshop on Visualizing Software for Understanding and Analysis (VISSOFT)*, Pages 32-40, 2002.
- [11] Christian Collberg, Stephen Kobourov, Jasvir Nagra, Jacob Pitts, and Kevin Wampler. A system for graph-based visualization of the evolution of software. In *Proceedings of the 2003 ACM symposium on Software visualization*, Pages 77-86, 2003. ACM Press.
- [12] M. D. Storey, K. Wong, F. D. Fracchia, and H. A. Müller. On Integrating Visualization Techniques for Effective Software Exploration. In *Proceedings of IEEE Symposium on Information Visualization*, Pages 38-45, 1997.

Mining RCS Data

Catherine Stringfellow Swetha Myneni Raaji Vedala-Tirumala Sreya Reddy

Department of Computer Science, Midwestern State University, 3410 Taft Blvd, Wichita Falls, TX 76308

Abstract - *As software systems evolve, it becomes important to know and to track the evolution of a system and its components. Data mining tools helps us to find the relationships between these components from patterns in the attributes of the data. The purpose of this paper is to describe how relationships between the attributes in the RCS (Revision Control System) change data can be found. These relationships may include which components of the system undergo repeated maintenance, how many lines of code are changed, the span of time in which files undergo change, as well as the type of maintenance (according to certain keywords used in the log). The technique is illustrated on a flight simulator system consisting of 409 RCS files.*

Keywords: *maintenance, software evolution, RCS data mining.*

1 Introduction

It is important to track the evolution of a system and its components as changes are made repeatedly. Components become increasingly difficult to maintain if these changes are not managed. The identification of these components and the classification of their changes help in maintaining a system or building an entirely new system properly.

This information can be used when we need to reengineer the components in the future. Problems will increase, if the changes are not understood in early stages. Late changes typically require changes to code in multiple components, which leads to problems in software architecture of the system [1, 7]. Software architecture problems are far more costly to fix and thus it is better to identify them as soon as possible. The change reports provided by RCS files help in finding these changes in the components. Change reports are written whenever developers change some part of the code. The information for each change is stored in the logs of each RCS file of the system. A change may affect many components. In addition, the time, author and the nature of the change are also recorded.

Data mining analysis is a tool to find patterns between attributes. Most existing analysis approaches employ quantitative data (statistical or series techniques) and qualitative data (such as details about the people who fix defective components). Data mining techniques analyze both qualitative and quantitative data using association rule mining. Decision tree learner is one of the most prominent machine learning techniques that is successful in classification problems, where attributes are discrete values [1]. Thus, discretizing attribute values enables association mining techniques to be more easily used.

This paper is organized in the following way. Section 2 describes prior research, section 3 describes the method followed in this case study, section 4 gives the results of applying data mining on RCS data of an industrial application system, and finally section 5 summarizes the paper.

2 Background

Data mining is commonly used in a wide range of profiling practices, such as marketing, surveillance, fraud detection and scientific discovery [6, 10]. It is a technique that can also be quite helpful in effectively managing the software development process [13]. Software defect data are typically used in reliability modeling to predict the remaining number of defects in order to assess software quality and make release decisions [8]. Recent work using data mining techniques analyze data collected during different phases of software development, including defect data [3].

A great deal of software engineering data exists today and continues to grow, and already the helpfulness of that data in improving software development and software quality has been demonstrated. Xie, Thummalapenta, Lo and Liu describe categories of data, various mining algorithms and the challenges in mining data [13].

Sahraoui, Boukadoum, Chawiche, Mai and Serhani [4] use a fuzzy binary decision trees technique to predict the stability among versions of an

application developed using object-oriented techniques. This technique gives an improved performance in deriving association rules over the classical decision tree technique, as well as solving the threshold limit problem and the naive model problem, in which decision support is not available *during* development.

As software gets more complex, testing and fixing defects become difficult to schedule. Rattikorn, Kulkarni, Stringfellow, and Andrews [3] presented an empirical approach that employed well-established data mining algorithms to construct predictive models from historical defect data. Their goal was to predict an estimated time for fixing defects in testing. The accuracy obtained from their predictive models is as high as 93%, despite the fact that not all relevant information was collected.

Association rule mining can help identify strong relationships between different attribute values [3]. These relationships can also help to predict any or many attribute values, hence, it is easy to be flooded with association rules even for a small data set [11]. Association rules are an expression of different regularities implied in the datasets, and some rules found imply other rules. Some of these rules can be meaningless and ambiguous: as a result it becomes necessary to constrain these rules to a minimum number of instances (e.g. 90% of the dataset) and set a minimum limit for accuracy (e.g. 95% accuracy level). The coverage or support of an association rule can be defined as the number of instances for which the rule can be predicted correctly. The accuracy or confidence of an association rule can be defined as proportion of correctly predicted instances to that of all instances for which the rule applies.

In rule mining, an attribute-value pair is called an item and combinations of attribute-value pairs are called item [1, 11]. Association rules are generally sought for item sets. There are two steps in mining association rules 1) identifying all the item sets that satisfy the minimum specified coverage and 2) generating rules from each of these item sets that satisfy the minimum support and minimum confidence. It is possible that some items sets may not generate any rules and some may generate many rules.

Much current software defect prediction work focuses on the number of defects remaining in a software system. This is to help developers detect software defects and assist project managers in allocating testing resources more effectively. Song, Sheppard, Cartwright and Mair had results that show for defect association prediction, the accuracy is very

high and the false-negative rate is very low [5]. They also found that higher support and confidence levels may not result in higher prediction accuracy and a sufficient number of rules are a precondition for high prediction accuracy. Srikant, Vu and Agrawal describe a technique to find only rules that meet certain constraints, building them into the mining algorithm, in order to get a subset of rules of interest in less execution time [6].

Several recent studies have focused on version history analysis to identify components that may need special attention in some maintenance task. Nikora and Munson found that measurements of code churn in an evolving software system can serve as predictors for the number of faults inserted in a system during development, and that this predictive ability can be improved with a clear standard for the definition of a fault [2]. Stringfellow, Amory, Potnuri, Georg and Andrews use RCS change history data to determine the level of coupling between components to identify potential code decay [7]. They looked at grouping changes according to a common author and a small time interval for checking in the changed files, as well as grouping changes to files within and between components.

This paper uses RCS files as input. RCS systems have many advantages: they manage multiple revisions of files and automate the storing, retrieval, logging and merging of revisions [7]. The automatic logging makes it easy to know what changes were made to a module, without having to compare source listings. Revision numbers aid in retrieving the changes. Included in the RCS data is the author, date, and log message summary in files.

3 Method

The method in this study follows a mostly data-driven approach, although the authors are focused on the task of maintenance. It also follows the steps in the mining methodology outlined in Xie et al. [13]. Preprocessing involved extracting data from RCS files, scrubbing the data, importing the data into arff files. Mining involved finding association rules using WEKA software [10]. Witten and Frank's book and WEKA toolkit are excellent resources for data mining [12].

A python program extracts attributes, from each RCS log. The results are stored in the form of text file that is then imported to a spreadsheet and stored in a csv file. That file is then converted to an arff file using

a online conversion tool. Finally, the arff file is opened in WEKA to obtain rules showing different relationships between attributes with association rule mining [9].

3.1 RCS data extraction

Figure 1 shows the first part of an RCS file. The head indicates that the last log made to the file (in this case 1.4, indicating there are 4 logs of changes made to the file). Each log has the date, time, and author of the change made.

```
head 1.4;
1.4
date 99.08.30.21.27.54;
author mikeh; state Exp;
next 1.3;
1.3
date 99.07.14.22.32.48;
author mikeh; state Exp;
next 1.2;
```

Fig 1: Change data in an RCS file with attributes date, author, etc.

The RCS data comes from a large flight simulation system, consisting of 409 RCS files of type .c,v and .h,v [8]. The RCS logs were first analyzed to determine the ten most frequently occurring keywords concerning changes in the RCS files. The keywords in logs that are indicative of change were determined to be add, delet, fix, bug, chang, fails, modify, error, correct, mov, fault, debug, updat, problem, delet and replac. File attributes, such as span of time from the first to the last change and the number of logs for a file were of interest.

Figure 2 shows the descriptions of two changes in the log. The descriptions are found at the end of the RCS file after the code, after the string Desc. Log 1.3, for example, has @d2 1, a2 1 in it, which means that on line 2 there was one statement deleted and one added. A programmer types in a description of the changed in comment (inserted between two @'s).

The python program extracts attribute values from each change log, including log number, date of change, author, number of deletes, number of adds and the frequency of each keyword occurring in the log associated with change. These extracted attribute values are stored in a csv file.

```
Desc @@
1.4
log
@fixed ccip/ccrp ct/hl code
@
text
@/*
1.3
log
@first checkin after tactics
@
text
@d2 1
a2 1
d61 2
a62 2
```

Fig 2: Example of two RCS logs.

3.2 Converting to arff

The resulting csv file output from the python program is converted in to arff file using an online conversion tool, as Weka supports only arff files [10]. The arff file format is becoming an increasingly important tool to transform these data into useful information.

3.3 Data mining for association rules

Data mining is the process of extracting rules from data. The WEKA software application is used in this study to find a set of rules and relationships between the change attributes extracted from the RCS files. WEKA requires data to be *nominal*. So filters are chosen to discretize these numeric attributes. After applying bins (grouping) and filters the rules are found and the relationships between attributes are determined.

Discretizing is done by selecting a particular attribute and then supplying the number of bins (and their range of values) into which the attribute is to be divided or grouped. Grouping data for a numeric attribute into bins converts it into a nominal attribute. Once all the attributes are nominal, association is performed using an apriori filter.

Figure 3 shows a WEKA screenshot of a list of the attributes extracted. The attribute SpanDays is highlighted (SpanDays refers to the span of days that a file underwent changes). It shows the six bins the numeric counts fell into and a frequency graph. It also shows minimum, maximum, mean, and standard

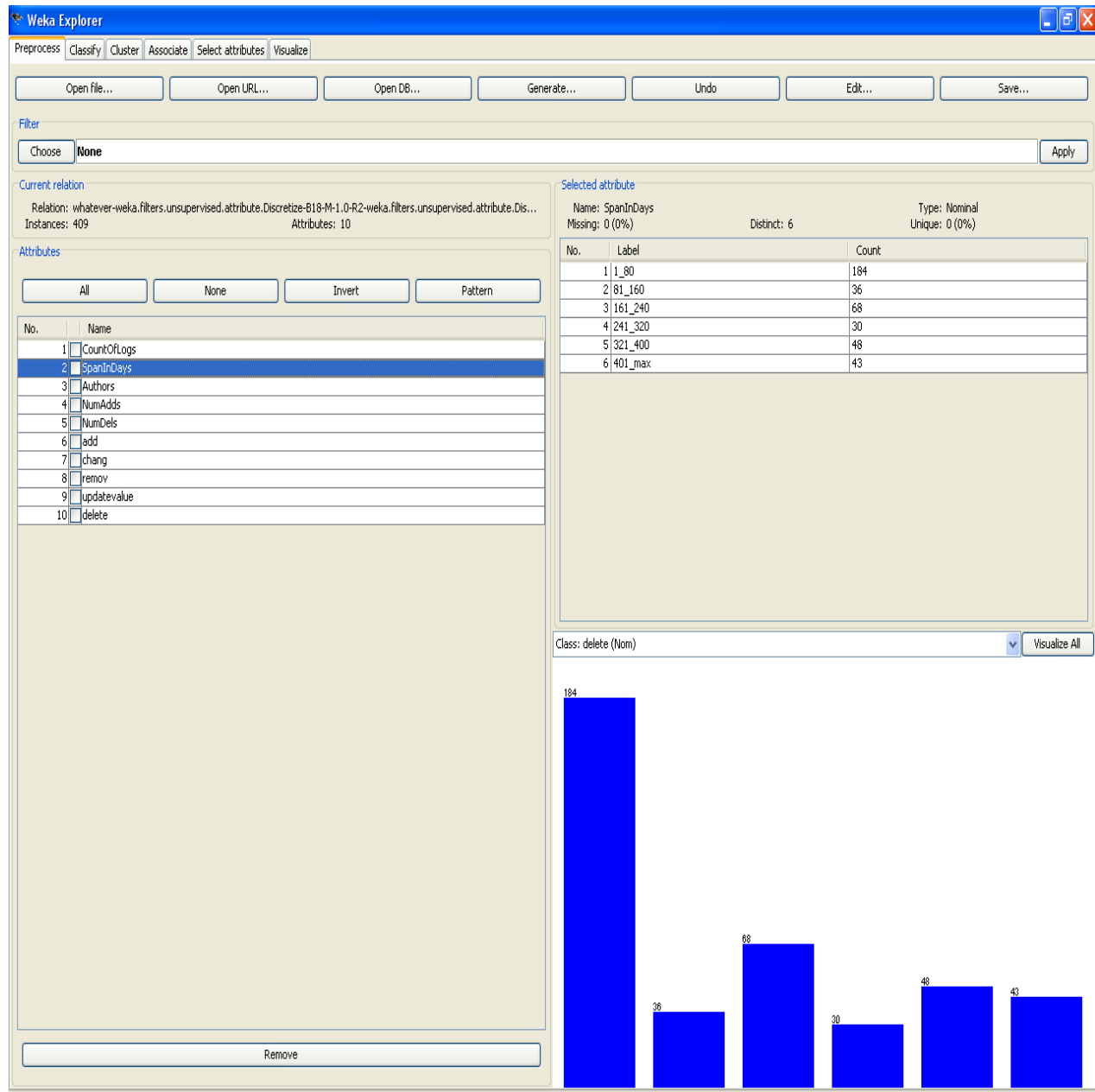


Fig3: Screenshot of the output from WEKA showing the graph for the attributes per log [7].

deviation values for those attributes. The maximum span, for example was 481 days.

Some data scrubbing was performed. One of the RCS files, for example, was ignored as it were not formatted correctly for data extraction. In addition, some data in the csv files when uploaded had zeros in some of the columns, and these were deleted. As already mentioned, data imported into WEKA had to be converted from numerical format to nominal format for data mining analysis.

4 Results

Using association rule mining and an apriori filter, 100 rules were found between the attributes. From these rules, we consider those rules that have a minimum confidence of 0.9 or above to find relationships between the attributes. The most significant of these rules are shown in figure 4.

Rule1: SpanInDays=1_80 184 ==>
 CountOfLogs=1_4 add=0_4
 updatevalue=0_1 181 conf:(0.98)

Rule2: SpanInDays=1_80 184 ==>
 CountOfLogs=1_4 chang=1_6
 updatevalue=0_1 181 conf:(0.98)

Rule3: fail=0_max 409 ==> bug=0_max
 409 conf:(1).

Rule4: mov=0_max 409 ==> bug=0_max
 409 conf:(1).

Rule5: debug=0_max 409 ==>
 fail=0_max 409 conf:(1).

Rule6: bug=0_max 409 ==> fail=0_max
 correct=0_max 409 conf:(1).

Rule7: Author=chris 388 <==>
 DateTime=529_max NumAdds=1_6 354
 conf:(0.91).

Rule8: NumLogs=0_5
 DateTime=529_max 257 ==> LogNo=0_4
 NumAdds=1_6 250 conf:(0.97).

Fig4: Rules found from association rule mining.

Figure 4 shows eight rules which have confidence 0.9 and above. Rules 1 and 2 (and several other of the 100 rules found) indicate that for files that undergo change in short span of time (here the bin was less than 80 days), there are few occurrences of the keyword add or chang (-e, -es, -ing, -ed). (In addition there were few occurrences of the keywords updat, remov or delet.) Also for files with a shorter span, these few changes were done by few authors for few logs. Rules 3-5 shows there is a strong correlation between keywords fail, bug, move and debug – these words can be almost taken as synonyms and probably counted together, as a result.

The added lines by author Chris are done mostly during the second half of the development schedule, and he mainly added only a few lines of code (Rule 7). Files with a few number of logged changes, had mostly only a few number of lines added (Rule 8).

5 Conclusion

This study shows that it is possible to mine RCS or other subversion control data in a system to identify certain change relationships. Identifying relationships between the attributes in RCS data could help in improving the performance of the system architecture and determining the components that need repeated maintenance. Data mining is one of the prominent tools that could help in finding these relationships and rules with both qualitative and quantitative data. Mining for change relationships may improve a system's performance early stages of development and save money. If the changes are found in latter stages, changing the whole system architecture could lead to a disaster.

6 References

- [1] Anand, R., "Association rule mining for a medical record system using WEKA," File paper, Midwestern State University, Fall 2006.
- [2] Nikora, A., Munson J., Developing fault predictors for evolving software systems. *Proc 9th Intl Software Metrics Symposium*, Sydney, Australia, Sept 2003, 338-349.
- [3] Rattikorn, H., Kulkarni, A., Stringfellow, C., Andrews, A., Software defect data and predictability for testing schedules. *Proc 18th Intl Conf on Software Engineering and Knowledge Engineering, SEKE'06*, San Francisco Bay, USA, Jul 2006, 715-717.
- [4] Sahraoui, H., Boukadoum, M., Chawiche, H., Mai, G., Serhani, M., A Fuzzy Logic Framework to improve the performance and interpretation of rule based quality prediction models for OO software, *Proc 26th Annual Intl Conf on Computer Science Software and Applications*, England, Aug 2002, 131-138.
- [5] Song, Q., Sheppard, M., Cartwright, M., Mair, C., Software defect association mining and defect correction effort prediction. *IEEE Trans on Software Engineering*, 32(2), Feb 2006, 69-82.
- [6] Srikant, R., Vu, Q., Agrawal, R., Mining association rules with item constraints. *Proc 3rd Intl Conf on Knowledge Discovery and Data Mining (KDD'97)*, Aug 1997, 67-73.
- [7] Stringfellow, C., Amory, C., Potnuri, D., Georg, M., Andrews, A., Deriving change architectures from

RCS history, *IASTED Conf. Software Engineering and Applications*, Cambridge, MA, Nov 2004, 210-215.

[8] Stringfellow, C., Mayrhauser, A., Applying the SRGM Selection to Flight Simulation Failure Data, Tech Report, Colorado State University, Fort Collins, CO, 2000, 1-16.

[9] Tkalcic, M., Online conversion tool, Ljubljana,slavnik.fe.unilj.si/markot/csv2arff/csv2arff.php.

[10] Waikato Environment for Knowledge Analysis (WEKA). *Data Mining Software in Java*, Nov 12, 2006 <http://www.cs.waikato.ac.nz/ml/weka/>

[11] Williams, C., Hollingsworth, K., Automatic mining of source code repositories to improve bug finding techniques. *IEEE Trans on Software Engineering*, 31(6), Jun2005, 466-480.

[12] Witten, H., Frank, E., *Data mining: practical machine learning tools and techniques*, 2nd ed., Elsevier Publications, 2005.

[13] Xie, T., Thummalapenta, S., Lo, D., Liu, C., "Data mining for software engineering," *Computer*, Aug 2009, pp. 55-62.

On The Construction Of A LaTeX To gDPS Compiler

X. Chen¹

¹Department of Information and Computer Science, University of Hawaii at Manoa, Honolulu, HI, USA

Abstract - *The Latex2gDPS compiler is a domain-specific compiler designed to translate the specification of a Dynamic Programming Functional Equation (DPFE) from LaTeX script into gDPS language. This eliminates the need for a D2P compiler user to learn another intermediate language (gDPS), and builds a bridge between the power of the petri net solution and the ease of using the popular text processing system of LaTeX. This paper describes the concept, design and implementation issues of the Latex2gDPS compiler, introduces its usage, development package, and how to extend to include new DPFE types. The translation of the Matrix Chain Multiplication DPFE is given as an example to demonstrate how the latex2gDPS compiler works .*

Keywords: Domain-Specific Compiler, LaTeX, gDPS, DPFE.

1 Introduction

Dynamic programming [1][2][3] is a category of optimization techniques initially studied by Richard Bellman since 1950s. Petri net [4][5][6] is a graphical modeling tool for parallel and independent events invented by Carl A. Petri. Recently, a general-purpose dynamic programming problem solver utilizing petri net was designed and implemented by Mauch and Lew [7][8]. In the problem solver, a DPFE (Dynamic Programming Functional Expression) is specified in the gDPS (general Dynamic Programming Specification) language. Then a D2P compiler translates the DPFE expressed in gDPS to Bellman net. The Bellman net, which is represented as a $N \times N$ adjacency matrix, is further translated into solver code in three formats: spreadsheet, java and PN code. These are eventually solved by spreadsheet program, java program and Petri Net simulator respectively. This structure is shown in Figure 1.

To use this general DP problem solver, a user needs to learn the gDPS language. There is a learning curve associated with it. To remove this learning curve, a solution is to allow the user to specify the DPFE in a familiar language. A domain-specific compiler then can translate from this language into gDPS, and make the gDPS language layer transparent to the user. Figure 2 shows how the Latex2gDPS compiler fits into this system. This original project is based on this idea. No related work has tackled this problem.

The reason we choose LaTeX as the source language, is because LaTeX is the most widely used text processing system in the scientific and computational field. It is easy to write DPFE formula in LaTeX, and LaTeX source code can be conveniently processed by a third party translation system. In comparison, some other text processing softwares, such as MS Word, do not possess all these nice features.

A compiler in general translates from a high level language into a low level language, and carries a series of tasks including lexical analysis, syntax analysis, semantic analysis, preprocessing, code generation and optimization. The Latex2gDPS tool translates between two high level languages, and there is not much need for code optimization so far. It may be more appropriately called a translator at this time, but we just call it a compiler in the discussion.

2 The Latex2gDPS Compiler

2.1 Overview

The Latex2gDPS compiler is built on the Lex/Yacc system, and is written in ANSI C. In short, a Lex file is used to provide the tokens used in the LaTeX input file, and a Yacc file is used to provide a grammar for the DPFE formulas in LaTeX input format; for some production rules of the grammar a semantic action is associated, so when such a production is recognized, the corresponding semantic action is written to the output gDPS file.

A two-pass parse process is used for the compiler. In the first pass the DPFE in the LaTeX input file is scanned into a container data structure of DPFE, and each of the functions, variables, operators and literals (numbers, constants) gets an entry in the symbol table. Types (number, variable or set) of the variables and literals are determined from context information or by using default value. In the second pass, which is the code generation pass, the obtained information of the DPFE is written into the gDPS output file.

In this work, the essential functions and framework of the Latex2gDPS compiler have been established. The processing include: 1) translation of the DPFE formula itself, 2) translation of the declarations, 3) fill in the other sections needed by the gDPS output. The DPFEs of fourteen types of dynamic programming problems can be translated by the current Latex2gDPS compiler. An API is provided which allows easy addition of new DPFE types.

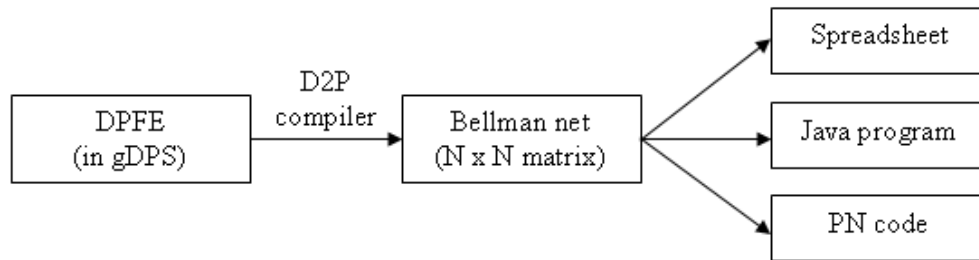


Figure 1. Architecture of the dynamic programming problem solver by Mauch and Lew

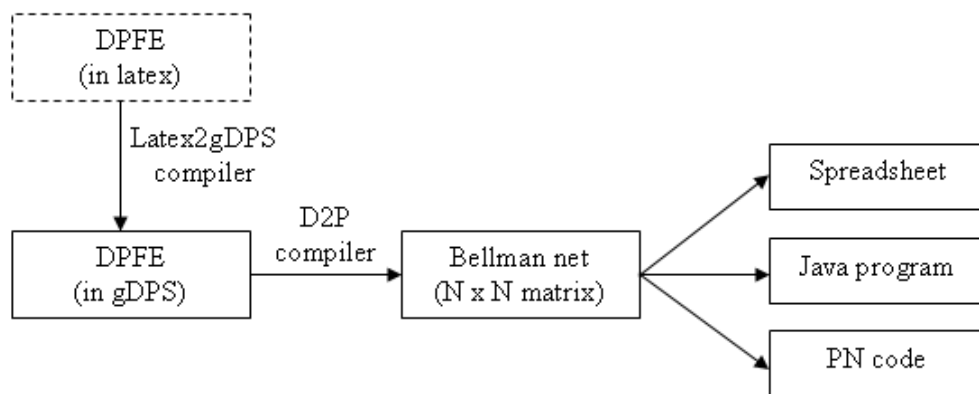


Figure 2. Add Latex2gDPS compiler to the architecture of the dynamic programming problem solver

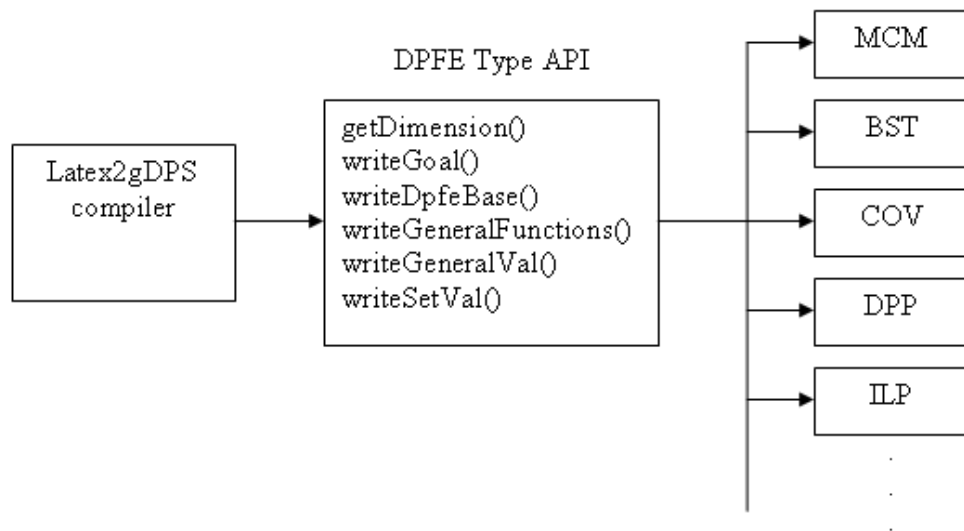


Figure 3. DPFE Type API of the LaTeX2DPS compiler

2.2 Data structures

2.2.1 Symbol table

A symbol table needs to be provided, which is set up as a hash table in the implementation.

The hash table data structure is a routine for a compiler project like this. It is appropriate and efficient in this case, because only *INSERT* and *FIND* operations are need, and both can be done in $O(1)$ time on a hash table. No *DELETE* operation is needed here.

Each entry of the symbol table contains the symbol's variable name, variable type and hash value. It can store the locations on which a symbol appears in the DPFE, but not at this time.

The collision can be solved by either open hashing (chaining) or closed hashing (open addressing). Actually most DPFEs only have a limited number of symbols. This is a small symbol table. Open hashing is actually used.

2.2.2 DPFE data structure containers

An appropriate data structure is needed to represent the DPFE formula. We need to organize the scanned DPFE formula in such a way that it is easy to translate it to the output gDPS file. The relevant data structure containers are: 1) the DPFE itself, including associated name and type, 2) DPFE optimization function, 3) DPFE base condition, 4) the goal function of the DPFE. The data structure of the DPFE itself is a container of the other three.

The goal function needs to keep track of 5) the function name, and 6) the parameter list.

The DPFE optimization function needs to keep track of 7) the optimization type (MIN, MAX or others) used, 8) the decision variable (s), 9) the decision operator, 10) the decision space, 11) the terms of the function, including both the operands and the operators, 12) the constraint condition.

The DPFE base condition needs to keep track of 13) the base value, 14) the constraint condition for this base value. It is possible that more than one base condition is specified, so the DPFE base condition data structure needs to be implemented as a collection (a list or an array, or something similar).

Figure 4 depicts all these elements and their relationship.

Data structures also are used to represent components of the gDPS file. When the DPFE formula in LaTeX format is scanned in, the components of the DPFE formula are translated into the components corresponding to each section of the gDPS file. Then in the code generation step, the parsed information will be written to the gDPS file.

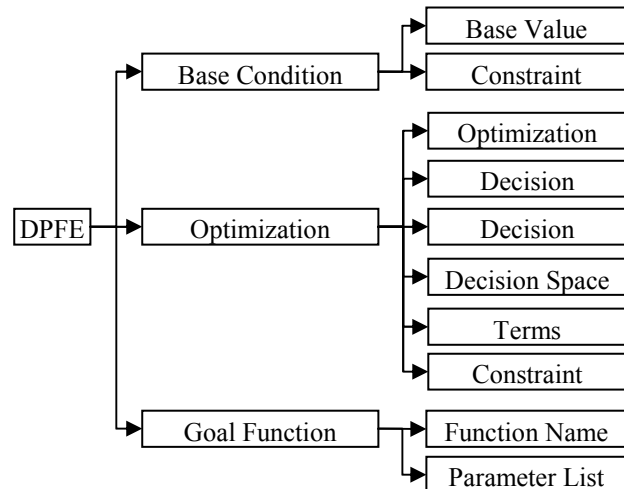


Figure 4. DPFE Data Structure Containers

2.3 The lexical analyzer

The lexical analyzer (or lexer) breaks the input file into a stream of tokens and feeds it to the syntax analyzer. The lexer generation tool Lex is used. Flex works too.

The lexer reports lexical errors on the input file. If a symbol is not recognized as any of the tokens, the program should report its location including line and column numbers. The current Lex file keeps track of the line and column count to report these information.

Another task of the lexer is to provide the actual value for literal tokens. For example, a token "5" is a NUMBER, it's value should be provided since it may be used in calculations. This information is provided by the current Lex file as well.

2.4 The syntax analyzer

The syntax analyzer determines whether a stream of input tokens can be recognized by the specified grammar and outputs the generated compiler file. Some of the grammar production rules have associated semantic actions. These associated actions will be taken if such a production is accepted. Parser generator Yacc or Hyacc [11][12] can be used to generator the syntax analyzer here.

The current Latex2gDPS grammar specification contains 65 terminals, 58 nonterminals and 133 production rules. When processed by Hyacc, the generated parsing machine contains 263 states after combine compatible states. It is a LALR(1) grammar.

The Latex2gDPS compiler first prints the header of gDPS file, next parses the input LaTeX file, then writes intermediate sections to the output gDPS file based on the parsing, and finally writes the footer information. A debug

option PNDEBUG is provided, the value of which can be set in the head of the syntax analyzer file pn.y. If you set PNDEBUG to 1, the generated Latex2gDPS compiler will print intermediate information during the parsing to standard output for debugging purpose.

2.5 Implementation challenges

In our work, we identify the following major implementation challenges to address, and describe our solutions so far.

2.5.1 Type recognition

There is no explicit declaration of variable type in the LaTeX DPFE formula input. So the type of a variable is determined by: 1) default type, and 2) context information. For example, in the Matrix Chain Multiplication problem, the default type is integer. But it is also possible the default is a real number, like in the continuous knapsack problem. So it is a question whether the default type should be included as part of the LaTeX DPFE specification, i.e., add a third choice: 3) explicit declaration of type. Use of context information is mostly in case of operators that connect two operands. If an operator is a set operator, then at least one of the two operands is a set. For example, from “a NOTIN b”, we can deduce b is a set, and a is an element of the set b. But we still need more context information to determine the type of set b’s element. From “a UNION b”, we can deduce that both a and b are sets. However, from “a + b”, we can deduce that both a and b are numbers. Currently, a term used in DPFE can have these types: ID, Number, Real, Infinity, Greek Symbol, Set, Index, Operator, Function and None.

2.5.2 Complete grammar specification

One thing special about dynamic programming problems is that each type is very different from the rest. There are big variations among the formats of the DPFE formulas. There is no common solutions and individual attention is needed for each type. There are two ways of constructing a comprehensive grammar specification:

1) Top down approach. By doing this we have an overview of all the dynamic programming problems, and come up with a set of general rules that can produce a grammar that fits all the DPFEs. This in general can be a clean solution. The hard part is to come up with such a set of general rules. We don’t know all the types of dynamic programming problems. Even if we do finish such a grammar specification for all the currently know DPFE types, once we come into a new type of dynamic programming problem, we will need to modify our grammar. For this reason, a comprehensive top-down approach is hard in practice since there are too many exceptions to handle. Consequently, a bottom-up solution is taken as below.

2) Bottom up approach. We handle different types of dynamic programming problems one by one. After we finish the work

on current available types of dynamic programming problems and make our grammar specification compatible with their DPFEs, we add another type and look at its DPFE, modify our grammar to fit it in. This is a doable incremental process.

Currently, fourteen DPFE types have been collected and included into the Latex2gDPS compiler [7][8][9][10]. Their acronyms and full names are listed in Table 1.

An API to add new DPFE types is given. Some macros are ready to use to ease the work of writing API functions. For unhandled cases the user needs to write a small piece of customized code. When incorporating a new DPFE type, just provide a C source file and a header file specific for this DPFE type, and compile again. See section 2.7 for details.

Table 1: DPFE Types In Current Implementation

Type	Full name
BST	Optimal Binary Search Tree Problem
COV	Optimal Covering Problem
ILP	Integer Linear Programming Problem
KS01	0/1 Knapsack Problem
LCS	Longest Common Subsequence Problem
LSP	Longest Simple Path Problem
MCM	Matrix Chain Multiplication Problem
ODP	Optimal Distribution Problem
RAP	Production: Reject Allowances Problem
SCP	Stagecoach Problem
SPA	Shortest Path in an Acyclic Graph Problem
SPC	Shortest Path in an Cyclic Graph Problem
TSP	Traveling Salesman Problem
WLV	Investment: Winning in Las Vegas Problem

2.5.3 Handling base conditions.

The DPFE formula may specify base conditions for the purpose of actually solving it for given parameters. A DPFE usually contains more than one equation, for example, one for the recursive optimization condition, and one for a base condition. If we determine that an equation is for the base condition, we can translate it into the gDPS DPFE_BASE section. This sometimes means we need to know the range of variables x and y. E.g., for “ $F(x, y) = 0$, if $x = y$ ”, if we know $0 \leq x < 2$ and $0 \leq y < 3$, we can put “ $F(1,1) = 0$ ” and “ $F(2,2) = 0$ ” to the DPFE_BASE section.

As the solution, three methods are used: 1) The user specifies base condition in the declaration section, 2) base condition is given in the DPFE formula, 3) use default specification.

2.6 The Latex2gDPS development package

Latex2gDPS works under any Unix/Linux/Cygwin/MacOS or similar system. These tools should be available: Gcc, Lex (or Flex), Yacc (or Hyacc [11][12]). The Latex2gDPS open source project package is available for download at [13]. Unzip the package to get the source files.

The *doc* folder contains documentation files in the development. The *gen_func* folder contains the definitions of general functions and set values needed by some types of DPFEs. The *io* folder contains input LaTeX files and output gDPS files for all the DPFE types. The *testcase* folder contains the output files of the test cases. The *test.pl* perl script does automatic regression test on the Latex2gDPS compiler. Basically it runs the Latex2gDPS compiler *pn* over all the *io/*/*.test.i* files, and generate *io/*/*.test.o* files, then compare these to the *testcase/*/*.test.o* files. If the comparison is the same, the *testcase* is passed, otherwise it's not. The *update.pl* perl script is used to update all the *io/*/*.o* files by running the Latex2gDPS compiler on all the *io/*/*.i* files. The rest of the files are the source files of the Latex2gDPS compiler.

The executable file *pn* is the compiled Latex2gDPS compiler, which can be rebuilt using *makefile* in the package by typing the *make* command in a unix-like environment. If the user makes some change and wants to make a new distribution, type the command: *make dist*. The Latex2gDPS compiler can also be built as a DOS executable in MS Visual Studio on the windows platform. This is done by first processing with flex and Hyacc, then compile with MS Visual Studio. A package containing DOS executable and example input/output files is also available for download at [13].

2.7 Add a new DPFE type

The Latex2gDPS compiler has an open design that allows the integration of new DPFE types easily. The API is shown in Figure 3, which include these functions:

- *GetDimension()*. This can be for a variable, an array or a matrix.
- *WriteGoal()*. For the goal function of the DPFE.
- *WriteDpfeBase()*. For the base condition(s) of the DPFE.
- *WriteGeneralFunctions()*. For other general function(s) needed by the DPFE specification. E.g., Mathematics functions.
- *WriteGeneralVal()*. For miscellaneous variables used by the DPFE. Each variable has associated type, name and value.
- *WriteSetVal()*. For set variables used by the DPFE.

To add a new DPFE type, say for the LCS (Longest Common Subsequence) problem, follow these steps:

- 1) Copy from existing DPFE module *.c* and *.h* files to create source files for the new module. For example, copy from *mcm.c* and *mcm.h* to *lcs.c* and *lcs.h*, and then modify them.
- 2) Modify these files: *const.h*, *const.c*, *dpfe_api.c*, *dpfe.h*, *makefile* accordingly. We are trying to minimize the amount of changes needed. More optimization can possibly be done to this part.
- 3) In the *io* directory, add a new subdirectory *lcs* to store input/output files of this module. This is not an essential part of the Latex2gDPS compiler, but used to include standard examples for different DPFEs.
- 4) Add the following files for regression test later. In the *testcase* directory, add an output file of the new DPFE type. Change *test.pl* to add testing code for this new module. Create *newmod.test.i* file in the *io* directory, and *newmod.test.o* in the *testcase* directory. This also is not an essential part of the Latex2gDPS compiler.

The *lcs.c* file is shown in Table 2 below. Three macros *MACRO_getDimension()*, *MACRO_writeGeneralVar()* and *MACRO_writeDpfeBase()* are provided to ease the task of writing these functions. These macros are defined in *dpfe.h*. If any of these is not needed, just provide an empty function, for example, *LCS_writeSetVal()* function is empty here. In *writeGeneralFunctions()* and *writeSetVal()*, the actual general functions and set values are defined outside the C code in the *gen_func* folder, and are copied here. This way the user can easily rewrite these and incorporate them into the output.

Table 2: The C source file for LCS DPFE: *lcs.c*

```
#include "lcs.h"

void LCS_getDimension(DPFE * dpfe, char * name, char * value,
    DimensionType type) {
    MACRO_getDimension(dpfe, name, value, type);
}

void LCS_writeGoal(FILE * fp, DPFE * d) {
    PARAMS_STRUCT * p = dpfe->params_struct;
    if (NULL == p) { return; }
    fprintf(fp, " %s(1, %d)\n", getAlias(d->gf->f_name), p->size - 1);
}

void LCS_writeDpfeBase(FILE * fp, DPFE * d) {
    MACRO_writeDpfeBase(fp, d);
}

void LCS_writeGeneralFunctions(FILE * fp, DPFE * d) {
    fprintf(fp, "GENERAL_FUNCTIONS_BEGIN\n");
    read_gen_func(fp, gen_func_LCS);
    fprintf(fp, "GENERAL_FUNCTIONS_END\n\n");
}

void LCS_writeGeneralVal(FILE * fp, DPFE * d) {
    MACRO_writeGeneralVar(fp, d);
}

void LCS_writeSetVal(FILE * fp, DPFE * d) {
    // empty
}
```

2.8 Usage of the Latex2gDPS compiler

This section describes the format of DPFE declarations to be used in the LaTeX input file, and the console commands used to run the compiler.

2.8.1 Declarations in the LaTeX input

Some context information are not available in the DPFE itself. To overcome this problem special declarations are used to specify such information. Seven declaration types are used:

- 1) `ModuleType`: a one-word identifier.
- 2) `ModuleName`: a one-word identifier.
- 3) `ModuleGoal`: `function_name([parameters])`.
- 4) `ModuleBase`: list of “`function_name([parameters]) = value [when (condition list)]`”.
- 5) `Dimension`: `dimension name : list of numbers` (separated by “,” in a single array; in case of a matrix, multiple arrays are used and are separated by “;”).
- 6) `DataType`: list of “type ID” separated by “;”.
- 7) `Alias`: list of “ID = ID alias” separated by “;”. An alias is another name used for the same variable. This is used such that a user can easily change the name of a variable without the effort of modifying the optimization function, which can be tedious and error prone.

The declaration part is quoted by `\begin{declaration}` and `\end{declaration}`. `{declaration}` is a LaTeX environment defined by user as: `\newenvironment{declaration}{}{}{}`. The order of these declarations does not matter. But only one of each declaration type can be used. In the declaration, the keyword is enclosed by \$ and following by a colon, then the declaration content.

2.8.2 Run the Latex2gDPS compiler

The executable is named “pn”. To run, type the command: `./pn infile [-ah]`

Parameter `infile` is the input file name. Optional parameters `-a` and `-h` are command line switches. `-h` shows help message. `-a` tells the compiler to use alias declared in the `$Alias$` declaration section.

In general, the input file should have suffix “.i”, although not mandatory. The output gDPS file name will replace the suffix with “.o”. If the “.i” suffix is not used by the input file, then the output file name just appends “.o” to the input file name.

3 An Example

An example for the MCM (Matrix Chain Multiplication) DPFE type is given below.

The Matrix Chain Multiplication problem aims to find the most efficient way of multiplying a sequence of matrices together. The efficiency here is defined by the number of multiplications and additions used. It is a classical optimization problem that can be solved by dynamic programming.

Given a matrix chain multiplication $A_1 * A_2 * A_3 * A_4$, which contains 4 matrices with dimensions of 3x4, 4x5, 5x2, 2x2 each. We want to find out the most efficient way of solving it using dynamic programming.

The MCM DPEF is shown below. Here we have goal = $f(1, n)$, $n = 4$. For $i = 1$ to 4, matrix A_i has dimension $d_{i-1} * d_i$, $D = \{3,4,5,2,2\}$.

$$f(i, j) = \begin{cases} \min_{k \in \{i, \dots, j-1\}} \{f(i, k) + f(k+1, j) + d_{i-1} d_k d_j\} & \text{if } i < j \\ 0 & \text{if } i = j. \end{cases}$$

The corresponding LaTeX specification and output are shown in Table 3 and Table 4.

4 Conclusion

In this paper we present our work on a domain-specific compiler Latex2gDPS, which translates a DPFE (Dynamic Programming Functional Equation) from its LaTeX source to the gDPS language. We review the background, go over problems and challenges, and describe the design and implementation.

Essential functions and framework of the Latex2gDPS compiler have been established. A pluggable design is given and relevant API interface functions are provided to accommodate new DPFE types. So far fourteen DPFE types are incorporated.

The future work will add more DPFE types, and further refine the Latex2gDPS compiler.

5 Acknowledgement

Thanks to the support and guidance of Dr. Art Lew, who provided abundant help in ideas and material throughout the project. His dedication to research has always been an inspiration.

Table 3: LaTeX source specification for a MCM problem

```

\begin{declaration}
$ModuleType$ : MCM
$ModuleName$ : MultipleChainMultiplication
$ModuleGoal$ : f(0, 0)
$ModuleBase$ : f(i, j) = 0 if (i = j)
$Dimension$ : d = {3, 4, 5, 2, 2}
$Alias$ : i = firstIndex, j = secondIndex, d =
dimension
\end{declaration}
\begin{equation}
f(i, j) =
\left\{
\begin{array}{ll}
\displaystyle \min_{k \in \{i, \dots, j-1\}}
\{ f(i, k) + f(k+1, j) + d_{i-1} d_k d_j \} & \& \\
\mbox{if } i < j \\
0 & \& \mbox{if } i = j.
\end{array}
\right.
\end{equation}

```

Table 4: gDPS specification translated from LaTeX source

```

BEGIN

NAME MultipleChainMultiplication;

GENERAL_VARIABLES_BEGIN
private static int[] d = {3, 4, 5, 2, 2};
GENERAL_VARIABLES_END

STATE_TYPE: (int i, int j);

DECISION_VARIABLE: int k;
DECISION_SPACE: decisionSet(i, j) = {i, ..., j - 1};

GOAL:
f(1, 4)

DPFE_BASE:
f(i, j) = 0 WHEN i = j;

DPFE:
f(i, j)
= MIN_{k IN decisionSet}
{
f(t1(i, j, k)) + f(t2(i, j, k)) + r(i, j, k)
};

REWARD_FUNCTION:
r(i, j, k) =
d[i - 1] * d[k] * d[j];

TRANSFORMATION_FUNCTION:
t1(i, j, k) = (i, k);
t2(i, j, k) = (k + 1, j);

END

```

6 References

- [1] Richard E. Bellman. (1957). *Dynamic Programming*. Princeton University Press.
- [2] Richard E. Bellman, Stuart E. Dreyfus. (1962). *Applied Dynamic Programming*. Princeton University Press, 1st edition.
- [3] Moshe Sniedovich. *Dynamic Programming: Foundations and Principles*, Second Edition. (2010). Chapman & Hall/CRC Pure and Applied Mathematics. 2nd Ed.
- [4] Carl A. Petri. *Communication with automata*. PhD thesis, University of Bonn and Darmstadt University of Technology, (1962).
- [5] R. Valette, B. Bako. *Software Implementation of Petri nets and Compilation of Rule-based Systems. Advances in Petri nets 1991, Lecture Notes in Computer Science 524*, Springer Verlag, 1991, pp.296-316.
- [6] K. Jensen and G. Rozenberg (eds.): *High-level Petri Nets. Theory and Application*. ISBN: 3-540-54125 X or 0-387-54125 X, Springer-Verlag, 1991.
- [7] H. Mauch. *Automated Translation of Dynamic Programming Problems to JAVA Code and their Solution via an Intermediate Petri Net Representation*. PhD thesis, University of Hawaii, March 2005.
- [8] A. Lew, H. Mauch. (2007). *Dynamic Programming: A Computational Tool*. Springer, 1st Ed.
- [9] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein. (2009). *Introduction to Algorithms*. The MIT Press, 3rd Ed.
- [10] F. S. Hillier, G. J. Lieberman. (2002). *Introduction to operations research*. McGraw-Hill Science/Engineering/Math, 7th Ed.
- [11] Xin Chen. LR(1) parser generator Hyacc. (2008). Available: <http://hyacc.sourceforge.net>
- [12] Xin Chen, David Pager. LR(1) Parser Generator Hyacc. In *Proceedings of International Conference on Software Engineering Research and Practice*, p.471-477. Las Vegas, July 18-21, 2011.
- [13] Xin Chen. Domain-specific compiler Latex2gDPS. (2008). Available: <http://code.google.com/p/latex2gdps/>

Software Cloning Detection Techniques: Comparison Criteria

A. Baqais¹, M. Ahmed²

¹College Of Computer Science and Engineering, KFUPM, Dhahran, Saudi Arabia

²College Of Computer Science and Engineering, KFUPM, Dhahran, Saudi Arabia

Abstract - Cloning code is becoming an increasing activity done by several programmers especially when there is a tight schedule to finish their tasks. The difficulty of detecting clone code lies in the intentional modification of some segments of the code by the programmers which may result in difficult-to-track debugs and increasing cost of maintenance. There have been different approaches and techniques proposed in the literature to solve this issue. However, these approaches either target a specific aspect of the issue or biased to some criteria rather than others. A comparison study of these techniques –based on some criteria- is proposed by this paper favoring approaches borrowed from artificial intelligence discipline. A final analysis is provided to layout the foundation for a new proposed solution that outweighs previous approaches.

Keywords: Software cloning, Code duplication, Code cloning, Code Similarity.

1 Introduction

Software cloning is an active research area in the field of software engineering. A considerable amount of papers has been published to address this issue from different perspectives. Some papers discuss the implication and the sequences of software cloning while others devote large sections to devise some techniques to identify the cloning fragments. Moreover, some papers provide a framework to compare the different techniques, tools and approaches targeting this domain.

Though it is considered as bad practice, Code duplication is quite popular in industrial software for many reasons. Due to the pressure of meeting deadlines, many programmers opt to copy some snippets of code and paste them somewhere in their program. Another reason is that the original code may have been fully tested and validated and as such some developers intentionally prefer to duplicate them especially when the code segment has advanced algorithms that consider different branches and computations. A third reason resides in the skills and the capability of the development team. Fresh or junior programmers tempt to duplicate a method or class if they feel they don't have the

necessary programming skills to code it themselves. Moreover, some code sections are not really intentionally duplicated, it's just the similar construct across different programming languages or the accidental duplication of functionality makes software cloning tools detect them as duplicated. For example, two for loops could be detected as clone segment even though it computes two different functions. As figure 1 illustrates, these are two functions that are mainly calculating the area of 10 objects and the square of 10 numbers. Clone detection will detect these as a cloning candidate code because they have almost the same number of lines, the same iterative variables; they only differs in the name of the returning variable. This is reported in the literature as false positive, that is, fragment of codes that look similar but actually semantically different and can't be classified as clones.

Researchers show strong interest in studying duplicated code because it helps in refactoring, evaluating code quality or reveal hidden bugs. Refactoring refers to the activity of reconstruction code structure without altering its intended behavior which conceptually similarly to software cloning where different codes perform the same function with different code structure. Code duplication is a strong indication of a design flaw and affects the code quality since it hinders other design techniques (such as abstraction or inheritance) of being implemented. In addition, duplicated code exhibits the same errors that its original has. Hence, a bug in an original code will be transferred to all duplicated code. For example, using CP-Miner has uncovered 28 bugs in Linux and 23 in FreeBSD[14].

It has been asserted that cloning increases maintenance time. Readability [1] is an interesting issue of studying software cloning. Another researcher Says [2] that duplicating code make it difficult to be understood, while states that it helps to understand the system since it provides sufficient information about the domain. There is no contrary in the above two views. The practice of code duplication reduces the readability of the program per se; however, it gives information on the system as a whole since it points out to important segments of code where duplication occurs more frequently.

<pre>For (int i; i<10; i++) {Area = i^2;}</pre>	<pre>For (int i; i<10; i++) {Square = i^2;}</pre>
--	--

Figure 1: Two loops that falsely could be detected as clones.

2 Previous Work

There have been many surveys investigating and comparing the different techniques and solutions addressing the area of code cloning. Each one has a different purpose and methodology to set up the comparison framework, and so does this paper. Bellon et al [3] have taken six approaches and tested them. Roy et al [4] provides a scenario and apply different techniques on that scenario while in [5] they provide a lengthy comprehensive comparison based on qualitative judgment. A recent systematic review has been undergone by [6] aiming to determine the role of cloning code during software evolution. In this paper, our attempt is to study different approaches aiming to discover hidden patterns that could guide interested researchers in their quest for solutions to the code cloning problem. This survey is different than previous studies in the sense that it's goal-guided to the solution. It simulates Artificial intelligence approach in finding heuristic function to determine the goal area of a certain problem by just exploring few branches of the problem rather than extensively and exhaustively exploring all the branches as in breadth first or going deeply to one aspect of the problem as in depth first. Here, the problem is software cloning, several approaches and criteria has been chosen carefully to get enough insight to the problem hoping that by studying and comparing these approaches, some interesting solutions may emerge and stem out of this discussion.

3 Software Cloning Metrics:

In this section, different methods (metrics) of detecting clones in code are presented. A short summary of the approach, algorithms and techniques used in each paper is presented. The metrics are:

- FIT (Frequent Itemset Techniques) [7].
- Tree-based [8].
- Near-Miss (Flexibly pretty printing and code normalization) [9].
- AST (Abstract Syntax Tree) [2].
- Function Metrics [10].
- Dependence Graph [11].
- Language independent [12].
- IR [13].
- Structural Clone (Data Mining for structural Clones) [1].
- FSM (Frequent Subsequence Mining)[14]

FIT (Frequent Itemset Techniques) [7]: The process starts by converting from source language to XML, apply mining frequent items while supplying configuration file to the result. It works by using frequent Item set technique, a popular technique in data mining. The process works by merging two frequent subsets to generate a combined frequent set of header and nested body. This can be clear in deleting common cloning constructs that have some headers but different bodies.

Tree-based: The paper provides a novel algorithm for clone detection. It has two steps: characteristic vectors and locality sensitive hashing. Characteristic vectors of a sub tree represented as $\langle c1, c2...cn \rangle$ refer to the occurrence of relevant nodes under the root of this sub tree. Afterwards, vector merging is applied among certain sub trees to generate merged vector for the combined code segments. Then we cluster the merged vectors using distance measure on vectors and report two vectors having same characteristics as clones.

AST: It uses abstract syntax tree (AST) to reconstruct the text into tree-like shape. The technique is conducted via a prototype tool called *Asta* that finds the number of occurrence of different patterns due to large outcome generated, thinning and ranking is performed. Thinning is a technique to reduce number of holes and nodes by providing an option list of parameters. Hole refers to the leaves of any sub trees pattern that can be resembled by wildcard“?” Ranking can be controlled based on some criteria: size, frequency or similarity.

Language independent: The paper presents a language independent approach for detecting software clones. It transforms the source code into condensed format where spaces and content lines are removed. The different algorithm steps are applied on the new file. First, an optimized string matching mechanism is performed and the result of modeling is recorded on a matrix. A diagonal line in the matrix denotes a cloning line. For the algorithm to effectively extract clone segments where some lines are changed, a post processing extraction method is used to represent the clone in either report-like or visual format.

FSM: It's based on frequent subsequence mining technique, where it calculates the number of occurrences of a subset (or subsequent) across the whole database (sequence database). It must be noted that this algorithm doesn't require contingency in performing its matching. An optimized version of this algorithm called *cloSpan* is used to identify only closed subsequence. (Its support is different than its super sequence). The steps are as follow: parsing some codes to build a sequence database, applying mining and prune false positive, further, *cloSpan* algorithm is adjusted by specifying the gap constraint to accommodate the problem of software cloning. It provides some

techniques of pruning false positive result and detection bugs emerged from cloning practice.

Dependence Graph: The paper introduces a new approach for detecting software cloning that – as the authors claim – doesn't suffer from the tradeoff between recall and precision. Their approach doesn't only consider syntax duplication but it dives deeply to extract any potential cloning based on the semantic. It reports that code as a graph consisting of vertices and associated arrows between these vertices. These arrows indicated dependency among vertices connected to them. The algorithm runs by computing two graphs to see level of similarity among different segment of code. Since such an algorithm is considered NP-complete, another approximation is applied where maximal sub matching is used.

Function Metrics: Assessment metrics of a source code for the purpose of detecting potential clone is proposed in this paper. An analyzer tool is used to transform the code into an intermediate form where metrics can be captured and illustrated graph of the code is shown. This approach is not aiming at detecting syntactical similarity only but it looks further to any similarity in the control flow of the two source codes. There are around 21 metrics used which are divided into four categories and being assessed using a scale of 1-8 for level of code similarity. This scale represents the level of similarity where level one indicates exact copy and level 8 refers to distinct control flow.

Near-Miss: It's based on TxL Transformation mechanism that applied on source code directly. It's performed on three steps:

- Parsing where the source code is converted into context free-grammar.
- Transform using a specified set of rules.
- Unparsed to provide a text output of the code.

The proposed idea of this paper is to address near-miss intentional cloning code. Near miss intentional cloning refers to the behavior of copying a code on purpose and changes it slightly for the sake of being appropriate to the new context. The solution proposal is based on three major steps: flexible pretty-printing, code normalization and filtering. Flexible pretty-printing is an enhanced version of pretty-printing which allows for further breaking of code segments into single lines and allow line by line comparison. Normalization refers to converting a segment of code into an abstract form. The three preprocessing steps above are fed into LCS (Longest common subsequence) algorithm. Two items are considered cloned if they appear in the LCS of two sequences. This algorithm is computing – intensive, so a further dynamic clustering is used to reduce

the number of comparison to only items contained in one class or cluster.

IR: It's based on Latent semantic indexing (IR) algorithm which is popular in information retrieval field. This algorithm basically finds any relationship between original documents and the terms contained within these documents. The paper uses a third-party tool to detect clones and categorize them into classes. Then, all redundancy of subclasses that appear in super classes is removed. Then another tool is used to transfer all the terms into an XML and generate term-document matrix (name of identifier and clone classes) and compute SVD using MATLAB. The results are clustered using *Cluto* tool and manual post processing analysis of the clusters is attempted.

Structural Clone: It introduces a formalization of structural clone, applies data mining techniques and implements a clone miner tool. Structural clones refer to the structure of many simple clones that may itself be replicated. Further, it classifies structural clones into different levels. Basically, the algorithm of this paper is detecting simple clones and increasingly finds relationship with other clones that may lead to structural clones. Two data mining techniques are used which are closed frequent set mining and clustering. The approach attempts to find different simple clones across various methods and files.

4 Framework of Comparison

In this section we will provide different criteria that set the framework of our comparison to the metrics discussed beforehand. The criteria are chosen carefully to test for competency and novelty of these metrics. Competency of metrics is referred to the common criteria that most of the metrics achieve but with different scale. This provides an insight to the superiority of some metrics over others. Novelty in this context refers to the uniqueness or features that some metrics provide while others don't. Brief explanation of these criteria is laid out below:

Recall: this criterion indicates the whole number of clones detected by various metrics whether the result is correct or false positive.

Accuracy: refers to the number of correct cloned codes detected and output by each metric.

Clone types: this refers to the types of clones detected by each approach. There are three types of software cloning as presented in the introduction section. Some metrics clearly state which types of clones are able to detect, others may not state it clearly. To compare different metrics fairly under this criterion, a human review by the author is conducted to determine unstated detection of cloning types.

Sensitivity to software size: this criterion refers to the fact that some tools are able to detect clones in small sections of code. The ability of the proposed approach of each metric to scale to larger software is an important criterion that must be considered.

Sensitivity to clone density: some tools work perfectly if the two codes under testing are slightly cloned. However, if the two codes are almost identical with many different types of clones, then the performance degrades drastically and may even lead to crashing the tool.

Use of AI Technique: this criterion specifies whether any of the artificial intelligence techniques has been applied in the proposed metrics.

Preprocessing & post-processing: preprocessing in this context refers to the sophistication of the approach under study in reducing the complexity of the problem. Post processing refers to the step each approach performs in order to get a better result or to introduce hid-den pattern.

Language & environment: this indicates the languages targeted by each metric and the environment where different approaches have been tested within.

A tool implementation: this indicates whether an implementation tool of the proposed metric has been provided or not.

5 Comparison Framework

Setting the framework of comparison is a crucial step as many discoveries and patterns can be revealed just by the structure of the framework. If the framework is structured and presented very well, hidden patterns will flash out quickly in front of the eyes of interesting readers. For example, the framework that can distinguish approaches very well based on some criteria resembles a decision tree where the best criteria are the best attribute with a highest information gain. On the other hand, framework that has too little information to give a decision on the superiority of approaches in getting to the solution would be as similar as a tree with many branches with same strength and no clue on which path is better in leading to the solution.

There has been different attempted to provide sophisticated framework that provides a strong insight to the capabilities and limitations of each approach. A side by side comparison where all approaches are evaluated against one criterion is commonly used in the literature of comparisons. Another approach would be to give cascaded view of the comparison by showing the criteria of each approach on its own. Both of the previous comparison framework strategies has some advantages and disadvantages relating to the objective of using any of them. The objective of this paper

is to point out the dimension of the problem that an interested researcher should focus on with some hints on which techniques has been used to attain this goal. Hence, a creative strategy has been developed in accordance with the aforementioned objective as figure 2 illustrates.

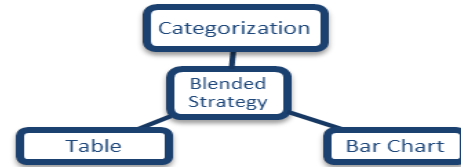


Fig 2: An overview of the developed comparison framework strategy

The developed strategy composes of three components: Categorization, Table and Bar Chart where each component is used to represent the suitable criteria. Table is used for criteria where a short description of how the approach implements these criteria is sufficient. For Example, tool implementation, programming languages and environment is very suitable to be represented by tables. Bar chart is more appropriate for criteria that can be quantified or leveled which can be used effectively in the two criteria: Pre & post processing and the use of AI Techniques as the bar chart provides us with the relation of the strength of these criteria and its effect in providing better solutions. For Instance, the paper that have higher value in the aforementioned two criteria reported to have a better impact on the overall solution. The third component called categorization is useful when the interest is not on the quantity of performance each approach scores under a specific criterion; but rather on which category of performance each approach falls in relatively to other approaches. For example, precision is divided into three categories: Medium, high and Higher and each approach are placed on the most appropriate category based on the level of convincing and the level of empirical validity it shows. Five criteria are represented by categorization which are: Recall, Precision, Clone Types, and Sensitivity to size and density.

5.1 Table

5.1.1 Language & environment:

Approach	Description
Function Metrics	Applied for procedure languages only.
Language independent	C, Smalltalk, Python and Cobol
Dependence Graph	C Program
FIT	Java, C++ and Prolog.
FSM	On Linux, FreeBSD, Apache, PostgreSQL. C & C++
Tree-based	On C and Java and it targets Linux OS.
AST	Java and C#, can't work with logical paradigm
Near-Miss	On C.
Structural Clone	J2SE 1.5 & Case Studies
IR	Microsoft NT Kernel, C

5.1.2 A tool implementation:

Approach	Description
Function Metrics	-
Language independent	-
Dependence Graph	-
FIT	-
FSM	CP-Miner.
Tree-based	DECKARD.
AST	ASTA.
Near-Miss	NICAD
Structural Clone	Enhanced Clone-Miner
IR	-

5.2 Bar Chart

5.2.1 Pre Processing & Post Processing

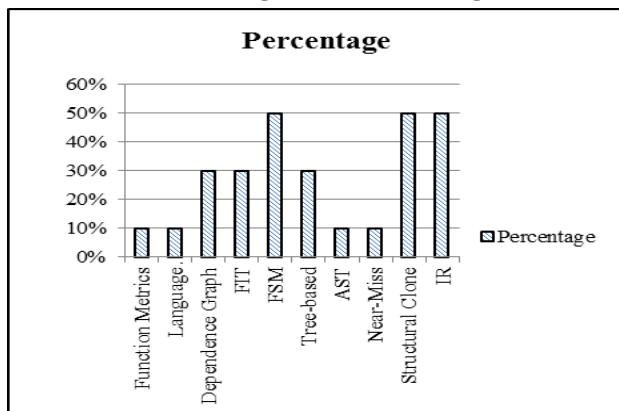


Figure 3: Preprocessing & Post processing

Figure 3: Preprocessing & Post processing

5.2.2 Use Of AI Techniques

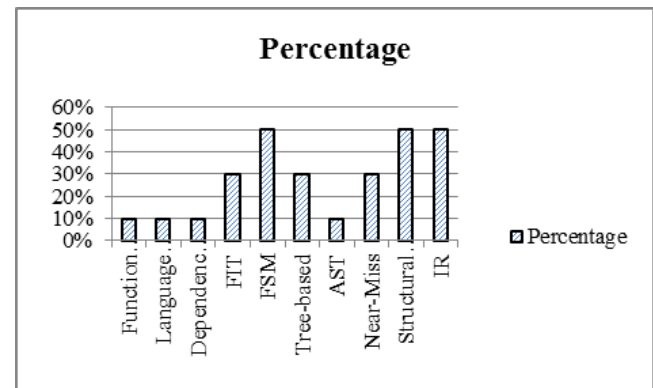


Figure 4: Use of AI Techniques

5.3 Categorization

5.3.1 Recall:

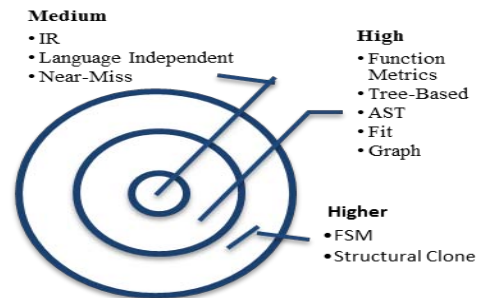


Figure 5: Categorization of Recall Criterion

5.3.2 Accuracy:

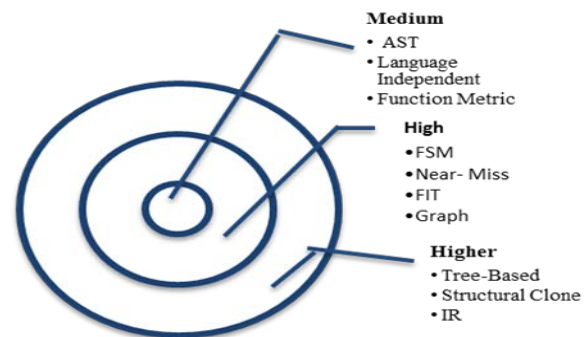


Figure 6: Categorization of Accuracy Criterion

Discussion: The level of accuracy is also categorized under this criterion comparatively among the approaches under study.

5.3.3 Clone Types:

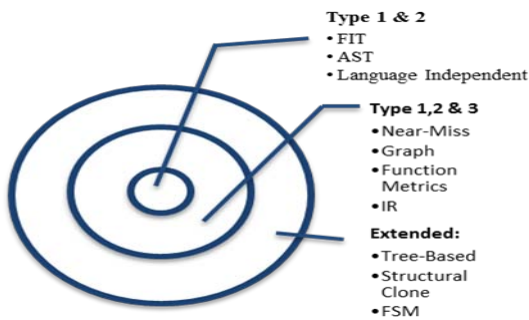


Figure 7: Categorization of Clone Type Criterion

Discussion: Some Approaches can detect clone types 1 and 2, while others are very sophisticated to detect type three as well. Moreover, some approaches are able to provide better insight at clone types and extend these types to include structural clone or bug related clones.

5.3.4 Sensitivity to software size:

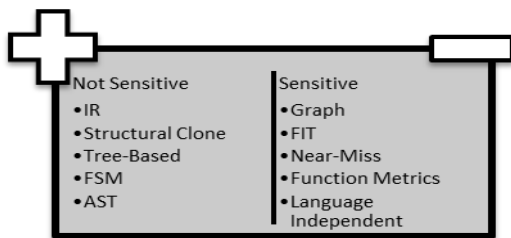


Figure 8 : Categorization of Sensitivity to Software Size

Discussion: Some Approaches are very scalable and able to detect clones even when the granularity of the code is very large. On the other hand, some approaches are using some algorithms that are very efficient working on a small size of code. When the code is getting larger, the algorithm either collapses or don't perform as it's supposed.

5.3.5 Sensitivity to clone density:

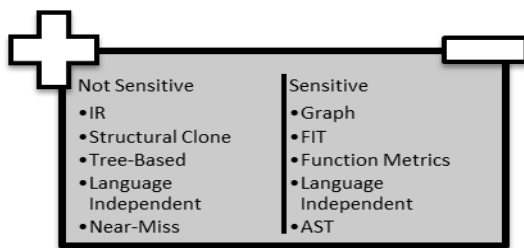


Figure 9: Categorization of Sensitivity to clone Density

Discussion: Some approaches are able to detect clones even when the density of the clones are very high and overlap

each other. This criterion is related mostly to accuracy and clone types since approaches who are able to identify overlap, redundancy and structural clones are more capable to be accurate.

6 Analysis:

The previous sections open our eyes to various techniques of clone detecting applied by different authors. These techniques are designed to target some aspects of the issue and to be optimal under certain context parameters. The criteria provided in the last sections illustrate the capabilities and limitations of these approaches. Interestingly, it illuminates the researchers to the different perspectives of the clone code problem while providing an extending definition to the issue. The types of clone criterion justify the above argument where some authors only consider identical clones with some renaming only. Subsequent authors noted that clones can be composed together to form a picture of structural clone. The idea of structural clone is very helpful in removing redundant overlapping clones. Accuracy and recall represents a challenging tradeoff that many authors attempt to achieve optimally. The number of clones that can be generated by any tool is apparently to the level of accuracy these tools provide. The use of pre and post processing techniques accompanied by the emergence of structural clone concept reduce the severity of this issue. The introduction of the criterion (sensitivity to larger code) implies the scalability of the approach to be used in large software contexts. Sensitivity to the clone density sets a rigorous test for these approaches. Many of the techniques are based on specific exemplary source code implemented in a certain environment, where the number of clones is known to be of moderate size. However, introducing this criterion shows the limitations of some approaches to work with source codes that are overwhelmed by copy-paste practice. Though such artificial set-up can be argued that isn't imaginary or not reflecting real world application, it still provides an insight to robustness and adaptability of the approach. Artificial intelligence attempts to optimize the way machines find solutions and its application has been well-received with wide recognition and acceptance. Approaches based on different AI branches such as machine learning, data mining, information retrieval have been favored by authors as it increases the performance and accuracy. AI techniques extend the detection of syntactical clones to semantic and structural clone. An omission of applying fuzzy logic is surprising though. As presented in the following section, the comparison survey leads to the necessity of soft computing techniques.

The remaining criteria (pre and post processing, languages and tool implementation) give descriptive view on the context and mechanism of the various approaches. Their impact on achieving the solution is of lower value.

Any successful approach can be ported with some modification to a different environment. These criteria may enlighten the interesting researcher to limitation and abstraction level of each metric, so a further enhanced version of algorithm can be implemented and then published. However, it must be noted that the main contribution is to provide high precision and recall coupled with scalability and robustness whether it requires pre and post processing step or not and whether the research was guided by a certain language or not.

7 Conclusion and Future Works

The previous sections provide an overview of the gaps and holes where many approaches fail to achieve. It's evident that our understanding of code clone has evolved through the years from exact copying into ability of detecting structural clones. There is a distinct difference between copy-paste practice in the sense of reusability and with the purpose of plagiarism. Plagiarism is considered bad under any argument, but the code cloning issue has shown an interesting feature that urge researchers to study it in depth. Though the previous approaches target similarity in source code, it should be understood that clones may crept to the designing and testing phase as well.

Interesting unattempt approaches to the issue of software clones has not been discussed in the literature and set as objectives to interesting researcher to investigate deeply. One dimension of the problem is how to determine that similar code are cloned and how to avoid the overlapping of code clones particularly when it's taken on larger software size. Fuzzy logic with its capability to provide a range of membership for all instances to be categorized might address this issue. The ability of an algorithm to survive in a larger code can be addressed by machine learning techniques. Machine learning is significant in finding patterns of code clone in a certain context and able to follow that pattern in a larger context. The issue of the tradeoff of recall/precision can be solved by using neural network allowing the errors of detecting code clone to be kept to a minimum.

8 Acknowledgement

The authors wish to acknowledge King Fahd University of Petroleum and Minerals (KFUPM) for the use of various facilities in carrying out this research.

9 References

- [1] H. A. Basit and S. Jarzabek, "A Data Mining Approach for Detecting Higher-Level Clones in Software," *IEEE Transactions on Software Engineering*, vol. 35, no. 4, pp. 497–514, Aug. 2009.
- [2] W. S. Evans, C. W. Fraser, and Fei Ma, "Clone Detection via Structural Abstraction," in *14th Working Conference on Reverse Engineering, 2007. WCRE 2007, 2007*, pp. 150–159.
- [3] S. Bellon, R. Koschke, G. Antoniol, J. Krinke, and E. Merlo, "Comparison and Evaluation of Clone Detection Tools," *IEEE Transactions on Software Engineering*, vol. 33, no. 9, pp. 577–591, Sep. 2007.
- [4] C. K. Roy and J. R. Cordy, "Scenario-Based Comparison of Clone Detection Techniques," in *The 16th IEEE International Conference on Program Comprehension, 2008. ICPC 2008, 2008*, pp. 153–162.
- [5] C. K. Roy, J. R. Cordy, and R. Koschke, "Comparison and Evaluation of Code Clone Detection Techniques and Tools: A Qualitative Approach," *SCIENCE OF COMPUTER PROGRAMMING*, p. 2009, 2009.
- [6] J. R. Pate, R. Tairas, and N. A. Kraft, "Clone evolution: a systematic review," *Journal of Software Maintenance and Evolution: Research and Practice*.
- [7] V. Wahler, D. Seipel, J. Wolff, and G. Fischer, "Clone detection in source code by frequent itemset techniques," in *Fourth IEEE International Workshop on Source Code Analysis and Manipulation, 2004, 2004*, pp. 128–135.
- [8] Lingxiao Jiang, G. Misherghi, Zhendong Su, and S. Glondu, "DECKARD: Scalable and Accurate Tree-Based Detection of Code Clones," in *29th International Conference on Software Engineering, 2007. ICSE 2007, 2007*, pp. 96–105.
- [9] C. K. Roy and J. R. Cordy, "NICAD: Accurate Detection of Near-Miss Intentional Clones Using Flexible Pretty-Printing and Code Normalization," in *The 16th IEEE International Conference on Program Comprehension, 2008. ICPC 2008, 2008*, pp. 172–181.
- [10] J. Mayrand, C. Leblanc, and E. M. Merlo, "Experiment on the automatic detection of function clones in a software system using metrics," in *International Conference on Software Maintenance 1996, Proceedings, 1996*, pp. 244–253.
- [11] J. Krinke, "Identifying similar code with program dependence graphs," in *Eighth Working Conference on Reverse Engineering, 2001. Proceedings, 2001*, pp. 301–309.
- [12] S. Ducasse, M. Rieger, and S. Demeyer, "A language independent approach for detecting duplicated code," in *IEEE International Conference on Software Maintenance, 1999. (ICSM '99) Proceedings, 1999*, pp. 109–118.
- [13] R. Tairas and J. Gray, "An Information Retrieval Process to Aid in the Analysis of Code Clones."
- [14] Z. Li, S. Lu, S. Myagmar, and Y. Zhou, "CP-Miner: a tool for finding copy-paste and related bugs in operating system code," in *Proceedings of the 6th conference on Symposium on Operating Systems Design & Implementation - Volume 6, Berkeley, CA, USA, 2004*, p. 20–20.

A Multiplicity Approach for Equilibrium-Driven Complexity Control

K. O. Chow

City University of Hong Kong
Hong Kong
cspchow@cityu.edu.hk

Abstract - *This paper uses a software technology approach to tackle the problem of complexity control. A multiplicity of system models, abstraction levels and development primitives are employed, with special attention paid to the use of relationship in the tackling process. The paper explains the steps needed to locate the balance point between competing factors in the search of the final equilibrium state. The benefits include extracting the part and relationship ideas found in software techniques and applied into controlling complexity.*

Keywords: Complexity control, equilibrium, balance, relationship, software method.

1 Introduction

Tackling complexity is an important study with a satisfactory solution remaining elusive. The issue of complexity exists in many subject areas and its study is multi-disciplinary [1, 2]. In the computer science discipline, complex computer systems abound after the proliferation of electronic computers in the past fifty years. Significant progresses have been made, although variation in success exists in different sub-areas such as the rapid hardware advance versus desired software productivity improvement.

Software study is an area that awaits significant progress in productivity, as in Brooks's paper on 'No Silver Bullets' [3]. Two important ingredients in complexity discussion are parts and relationships. Software engineering has studied these two items extensively in an effort to develop quality software products through a sound development process. Examples are entity and relationship [4] and, object and relationship [5-9]. In tackling software complexity, the key concern is to control complexity. A number of key concepts evolved, like abstraction, decomposition and classification. These ideas share similarities with other disciplines, though differed in multi-modeling and relationships.

Due to the complicated intricacy and the subsequent effort to seek a better understanding through a systematic methodology, the study of complexity is often tied to system study, forming complex systems. In this paper, we adapt findings from computer science and take a system methodology approach to analyze, understand and solve the intertwining relationship in the task of complexity control. Important ideas include multi-modeling, abstraction hierarchy and development primitive. System modeling makes it easier

to see connections, relationships and patterns of interaction, as multiple perspectives put things in better context. We will show that these ideas have generic attribute and can be applied to other disciplines.

This paper is organized as follows. Section 2 states the key concepts related to complexity control, equilibrium and balance. Section 3 discusses the software engineering ideas used in controlling complexity. Section 4 presents the proposed approach. The last section evaluates the proposal and gives the conclusion of the paper.

2 Complexity and Equilibrium

This section presents the key concepts relevant to the paper proposal, namely complexity control, balance and equilibrium.

2.1 Complex System

A complex system is commonly defined as a system with numerous parts. And many times these parts are arranged intricately. Although parts and intricacy have been the conditions to be investigated, they tend to emphasize on the structural arrangement of parts, with less attention given to interaction, transition and processing. A comprehensive consideration should include both structure and behavior, as behavior is affected by structure and both are closely related. Structure-based behavior includes object interaction, state transition and function processing.

2.2 Equilibrium and Balance

An important consideration in the study of complexity is system equilibrium. Equilibrium is a system condition in which competing forces are balanced. Balance point is the desirable point among opposing forces. For example, the balance point in decomposing a function is seven or less sub-functions, as we want to avoid ending too many sub-parts and become over-decomposition. The ability to maintain balance is important to achieving system equilibrium. System balance is a state condition enabled by interaction behavior and function processing, and is obtained by:

- Locate factors to be included in balance considerations
- Identify balance point

- Act on factors to move closer to balance point

2.3 Complexity Control

Complexity control is meant to prevent from aggravating the degree of complexity in a task. A number of techniques have been evolved. The key idea centers in limiting the part number and consequently the scope of consideration. The techniques include:

- Classify into types of units
- Decompose into levels of units
- Compose units into groups

A number of control effects are resulted in the process of controlling complexity. They include:

- A leveled structure formed, with levels of parts
- The number of parts reduced, via composition into larger units
- Relationships between parts become explicit

These results can be viewed from a perspective of relationship and translated into having generalization, aggregation and composition relationships.

In sum, software development methodology is suitable to the complex system study. Because of the similar attributes exhibited, useful system properties include modeling, nesting, system structure, interaction behavior and conditional state. The followings are derived from this section:

- Include behavior into considerations
- Incorporate relationship and state condition
- Adopt the use of multiple techniques

3 Software Development Concepts

This section states the key concepts in the system methodology approach, with the aim to extract relevant ideas for an equilibrium-driven complexity control.

3.1 System Perspectives

Abstraction In computer science, an abstraction is a simplified view of a system which contains selected system details important for a particular purpose. In other disciplines, such as humanity and business, the concept of abstraction has also existed for a long time. Like stepwise refinement and information hiding, abstraction is an important concept in computer science. In software engineering, it is the issue of system complexity, equivalent to Brooks's essential complexity, which motivates the search for more effective solution. Model-driven software development is the key idea in both the classical and modern structured and object-oriented methodologies, and is a central tool in complexity control. This importance is due to the raising of the abstraction

level in software development. A weakness in abstraction is that it is not clear when should abstraction be stopped. This is so because the nature of abstraction implies that the process should be complete when essential features are obtained. However, the task of separating the essentials from non-essentials is a lengthy iterative process and necessitates continuous refinement. Other weakness includes imprecise abstraction.

Multiple Perspectives Abstraction is closely related to modeling. The essence of abstraction is to identify essential properties and ignore unnecessary details. In other words, abstraction involves selective attention and selective ignorance. This focus of concern is based on the modeling perspective. The historical progression of the software technology shows a transition from a serial execution emphasis to one of function processing and later towards oriented towards objects. These emphases reflect perspectives taken in understanding and modeling the system. The current trend in software development is to migrate from a single perspective to multiple perspectives, that is, model a system with different perspectives. Multi-modeling is the prevailing trend in current software development, as is shown in UML [10]. There are different proposals in terms of 2-model, 3-model and others. The more common one is a dual model, comprising structural model and behavioral model. Multiple models create a shortcoming of inconsistencies among system models. The inclusion of multiple models complicates the abstraction process. Other weaknesses are domain specificity and domain dependency.

3.2 Hierarchical Levels

When a system is decomposed into levels, a hierarchy of system is formed. These abstracted hierarchical levels can be grouped into high and low levels, and interpreted as shown in Figure 1 below:

High Level	What, General, Abstract, More information hidden
Low Level	How, Detailed, Concrete, Less information hidden

Figure 1. Abstraction Level

Using the concept of levels, software method devises the techniques of decomposition and composition and uses them to organize parts. However, no relationships exist between the hierarchical levels. This applies to the structured approach as well as the object-oriented approach. The standardized UML incorporates meta-model, using a more generic abstraction. This has the benefit of domain-independent characteristic. But the weakness is that it still has to involve domain-specific data. On the other hand, traversing between levels may involve changing of views. This implies that we can unify views through traversal of abstraction levels.

In addition to the lack of relationships, there are three other

shortcomings. It is not clear what constitute high and low levels. Abstraction level is a relative term, with no absolute value. Inconsistencies exist between abstraction levels. In addition, inconsistencies may also exist between models and abstraction levels.

3.3 Relationships

Relationship is also an under-investigated area in complexity study. The inclusion of the relationship concept in software development is a relatively new idea in computer science and happens only with paradigm shifts. It is only with the coming of the data modeling with entity relationship [4] and the object modeling [5-9] that the concept of relationship is explicitly included. The preceding paradigm of structured approach only embodies the hierarchy concept and does not have any relationships.

Similar to abstraction, the concept of classification exists in many disciplines such as biology, economics. Classification uses the idea of commonality to group similar things into types, forming the a-kind-of relationship. At the same time, the hierarchical level concept is employed to produce an organized depiction of parts. In the design of a complex system, classification is used to control complexity by reducing the amount of details that a designer needs to consider in similar groups. To manage a system more abstractly, the idea of subsystem is used. A subsystem may recursively include other subsystems.

The object-oriented methods in the 90s usually prescribe the basic relationships of association, aggregation and inheritance. Currently, the UML 2 [10] includes the following relationships, namely aggregation, association, composition, dependency and generalization. Using the dual model concept, a more complete list of relationships is shown in Figure 2 below.

A weakness is the lack of relationships between the structural and behavioral models. This paper proposes to use relations to maintain balance in complex systems.

Structural relationship	Aggregation, association, composition, generalization
Behavioral relationship	Sequence, dependency, aggregation, realization

Figure 2. List of Relationships

3.4 Process Primitives

It will be more effective in complexity control if activities are organized in terms of path and direction, together constituting basic activity primitives. A pattern of activities gives rise to a progression path. There are different patterns. They can be linear, fixed, dynamic or iterative. There are different directions of movement, such as top-down, bottom-up, middle-out. It is the combination of the development process stages and the process primitives that gives rise to a

variety of process models. A full list of the primitives is given in Figure 3 below.

Path	Sequence, iteration, linear, singular, parallel, fixed, dynamic
Direction	Top-down, bottom-up, middle-out, forward, backward, inside-out, outside-in

Figure 3. Activity Primitive

4 Multiplicity for Equilibrium

This section links the ideas described in Sections 2 and 3 and explains the proposed equilibrium-driven complexity control.

4.1 Multiplicity Approach

As has been described in Section 3, the key idea is in the exhibition of a multiplicity characteristic. They become the building blocks, or parts, in complexity control. The key multiplicity elements are summarized follows:

- Multiple models
- Multiple hierarchical levels
- Multiple relationships
- Multiple paths and directions

4.2 Proposal

As has been described in Section 2, the key ideas to achieve equilibrium are to obtain the followings:

- Condition state
- Balance point
- Competing factor

We propose to map and unify these ideas to form the building blocks for the proposal. We use the software development methods as exemplary elements in the proposal. Figure 4a gives a mapping between the structural and behavioral models. Figure 4b describes the structure-dependent behavior model by organizing the dual models into two hierarchical levels.

Structure	Behavior
Class, relationship	Use case scenario
Property	Interaction sequence
Method	Activity processing

Fig 4a. Structural/Behavioral Models

Behavior
Structure

Fig 4b. Dependency

The structural model is of primary importance in software development. Subsequently it takes a primary role, a major

share and a leading order. It will first give the earliest meaning. It also forms the lower level in a hierarchy of model dependency. The behavioral model is based on, and dependent upon, the structural model, serving as a guide in development direction and a dependency relationship. Figure 5 shows the mapping among equilibrium criteria, competing consideration factors and relationships employed.

Equilibrium Criteria	Competing Factors	Relationship Used
Importance	Primary vs. secondary	Generalization, aggregation
Share/weight	Major vs. minor	Ordering sequence
Directional guide	Based on, driven by, oriented to	Dependency

Figure 5. Building Blocks

The competing factors are used as opposing forces to locate the balance point in searching for equilibrium state. For example, the balance strategy between primary and secondary factors will be primary first. The relationships of generalization and aggregation are employed as structure with leveled parts classified, decomposed and composed. The behavior includes interaction between parts, state transition and function processing. All these are based on the structural model built previously. Equilibrium is the final state where the primary and secondary factors are balanced. Complexity is controlled in the process.

4.3 Complexity Control Steps

From the relationships derived in previous section, the sequence of modeling is to first create a structural model, and then a behavioral model. This sequential order conforms to the recommendations given in most object-oriented methods. Figure 6 lists the procedural steps.

Modeling	Extract essentials
	Select emphases
	Create models
	Identify parts
	Form hierarchies
	Establish relations among parts
	Relate dual models
Design	Sequence activities with primitives
	Set condition states
	Consider competing factors
	Establish balance point for equilibrium

Figure 6. Steps

5 Conclusions

We have presented an approach to tackle the question of complexity control. In our proposal, we advocate a

multiplicity approach, encompassing multiple dimensions of system modeling and development activity. The paper explains the steps needed to locate the balance point between competing factors in the search for final equilibrium state. The benefits include extracting the part and relationship ideas found in software techniques and applied into controlling complexity. There are a number of outstanding issues. These include:

- Work towards domain-independent proposal by resolving the problem of domain specific and domain dependence
- Derive a detailed list of relationships
- Establish relationships between structural and behavioral models to remove inconsistencies, and relationships between hierarchical levels to ensure correct mapping

The proposal will also require realistic applications for further verification. Future works include incorporating modeling language and developing a software tool.

6 References

- [1] Weaver, Warren, Science and Complexity, American Scientist 36:536, 1948. Retrieved from <http://www.ceptualinstitute.com/genre/weaver/weaver1947b.htm>.
- [2] Johnson, Neil F., Two's Company, Three is Complexity: A simple guide to the science of all sciences. Oxford: Oneworld, 2007.
- [3] Brooks, F.P., "No Silver Bullet – Essence and accident in Software Engineering", Computer 20, 4, pp. 10-19, 1987.
- [4] Chen, Peter, "The Entity-Relationship Model - Toward a Unified View of Data", ACM Transactions on Database System, 1(1):9-36, 1976.
- [5] Booch, Grady, "OO Analysis & Design with Applications, 2nd Ed. Addison-Wesley, 1994.
- [6] Rumbaugh, James et al., Object-Oriented Modeling and Design, Prentice-Hall, 1991.
- [7] Shlaer, S and Mellor, S., Object Life Cycles: Modeling the World in States, Yourdon Press, 1992.
- [8] Jacobson, Ivar, et al., Object-Oriented Software Engineering: A Use Case Driven Approach, Addison-Wesley, 1992.
- [9] Martins, James, Principles of Object-Oriented Analysis and Design, Prentice-Hall, 1993.
- [10] Booch, Grady, Jacobson, Ivar & Rumbaugh, James, OMG Unified Modeling Language Specification, Version 1.3 2000.

SESSION

WORKSHOP ON NONLINEAR SOFTWARE ENGINEERING REVOLUTION BASED ON COMPLEXITY SCIENCE

Chair(s)

Jay Xiong

NSE Extracts Software Models from Source Code - Software Modeling Revolution Based on Complexity Science

Liana Ye¹, Jay Xiong²

PeaceNames.com, USA¹, NSESoftware, LLC, USA²

Abstract

Software specification and implementation are intertwined. Model driven software development is considered harmful if the models are outcomes of reductionism and superposition of linear software development processes. Successful software products are outcomes of a non-linear approach. This paper introduces a nonlinear, holistic, and dynamic software modeling approach based on complexity science, driven by platform-independent Java source code or platform-dependent programming language source code, called NSE modeling. NSE modeling enables software design automation from stubs in top-down pre-coding design, and enables the coding process to be bottom-up to further design, and effectively incorporates model-driven software design into a non-linear process.

Keywords: software modeling, MDE, MDA, MDD, software requirement engineering, software design, coding, testing, quality assurance, maintenance

1. Introduction

“A model of a system is a description or specification of that system and its environment for some certain purpose” (OMG). “A **model** is an abstraction of a (real or language based) **system** allowing predictions or inferences to be made.” [1]

Claimed by Jim Arlow and Ila Neustadt in their book, [2] MDA, a set of large software system standards, “The future of UML may be a recent OMG initiative called Model Driven Architecture (MDA). MDA defines a vision for how software can be developed based on models. In MDA, software is produced through a series of model transformations aided by an MDA modeling tool. An abstract computer-independent model (CIM) is used as basis for a platform-independent model (PIM). The PIM is transformed into a platform-specific model (PSM) that is transformed into code.”

Critic of MDA Harry Sneed pointed out [3]: “Model driven considered harmful:

- * Model-driven tools magnify the mistakes made in the problem definition;
- * Model-driven tools create an additional semantic level to be maintained;
- * Model-driven tools distort the image of what the program is really like;
- * The model cannot be directly executed. It must first be transformed into code which may behave other than expected;
- * Model driven tools complicate the maintenance process by creating redundant descriptions which have to be maintained in parallel;”

* Model driven tools are designed for top-down development;

* Top-down functional decomposition creates maintenance problems.”

Opposing MDA is Architectural Driven Modernization, ADM. It produces **Knowledge Discovery Meta-model** (KDM) for model-based reverse engineering of legacy systems. KDM describes information systems in terms of various assets (programs, specifications, data, test files, database schemas, etc.) shown in Fig. 1.

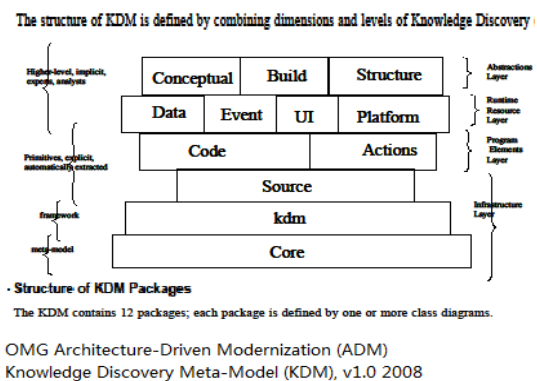


Fig. 1 KDM structure from OMG ADM group.

In summary, if a UML design can really replace the programming code as envisioned by some, then it becomes just another programming language. The question is, which is easier to change, the design documents or the programming language. Harry Sneed: “This depends on the nature of the problem and the people trying to solve it. If they are more comfortable with diagrams, they can use diagrams. If they are more comfortable with text, they should write text. Diagrams are not always the best means of modeling a solution. A solution can also be described in words. The important thing is that one model is enough – either the code or the diagrams. They should be reproducible from one another.”

2. ADM Approach Drives to NSM revolution

Many excellent software development approaches or methodologies have been developed to streamline a process. All these tools can be used by people for holistic examination for a software system in development. But people do forget some details. ADM group has provided KDM view from existing large systems, which are developed from many nonlinear processes in the past. ADM approach sets the direction for NSM revolution but extracts from machine code, creating another layer under existing source code. In fact, it throws away existing

source code, which is the result of successful past non-linear processes. Automating model creation from legacy source code systems, studying and improving these working models from the past, and incorporating new requirements into code generation can revolutionize software engineering.

From practical experience, the existing software engineering paradigm has always been divide and conquer, thus an outcome of linear thinking, reductionism, and superposition.

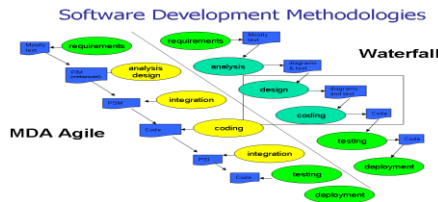


Fig. 2 MDA Agile software development process [5]

Products woven together with these approaches are un-maintainable. Any requirement change or code modification will make the products unstable as current software processes ever larger amount of dynamic data, as experience has already shown. Today, most critical software defects are introduced into a software product in the requirement development phase and then the design phase. Yet dynamic testing of the product is performed after coding. The National Institute of Standards and Technology's study [6] said: "Briefly, experience in testing software and systems have shown that testing to high degrees of security and reliability is from a practical perspective not possible. Thus, one needs to build security, reliability, and other aspects into the system design itself and perform a security fault analysis on the implementation of the design." Working models from legacy systems need to be re-examined to satisfy these new requirements.

2.1. ADM approach has problems to work out

According to ADM structure in Fig. 1, ADM has to rewrite the source code of a legacy system in KDM. If we use the code generated by MDA, it will take a large amount of resources to cover an existing legacy system in detail, unless the models are ugly enough with a lot of detailed information making these models hard to view.

Either MDA or ADM may provide two sources approach to software modeling (Fig. 3) with one in models or diagrams format for people to understand a complex software system, and the other in textual format, or source code, for computers to interpret the system. There is a big gap between the two sources.

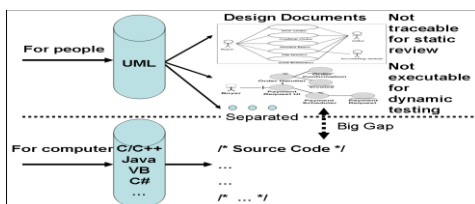


Fig. 3 Two-Source Approach in software modeling

Traditional code-driven engineering approaches do not support software modeling for high-level abstraction, making the developed software product hard to understand and hard to maintain.

Similar to ADM legacy system approach, a proposed Nonlinear Software Modeling (NSM) [4] takes only source code as the single source for both human and machine understanding, instead of avoiding the old code, as both MDA and ADM are doing. NSM creates models of a complex software system with colorful, dynamic, virtual, interactive, traceable, linkable, auto-convertible, accurate, precise display, assuring model consistency with the source code.

2.2. Requirements to satisfy NSM

The foundation of NSM is laid in Fig 4. To count for all the elements indicated in the framework and using existing technologies and best practice today, the following requirements to satisfy NSM are feasible.

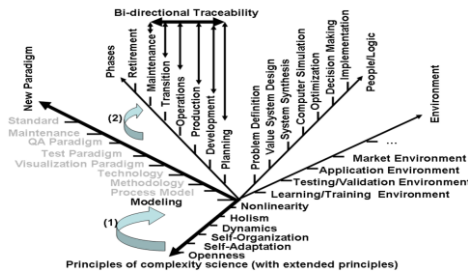


Fig. 4 The framework for NSM development

- (1) Models or diagrams should be meaningful for describing both high-level abstraction and low-level program logic from the same source.
- (2) Models or diagrams should be holistic, colorful, interactive, dynamic, and traceable.
- (3) Programming sources should be stable and platform-independent, such as Java-DSL or C++, without changes for high-level abstraction to generate new models or diagrams.
- (4) Tools should be fully automated to generate models or diagrams directly from such source codes.
- (5) Models or diagrams should not take a large amount of space in static storage.
- (6) Bi-directional traceability must be supported.
- (7) Developed software products should be truly maintainable, counting in all related models, diagrams, documents, test cases, and the source code.

3. NSM solution

Shown in Fig. 5 in NSM, one source is used for both human understanding and computer understanding of a software product.

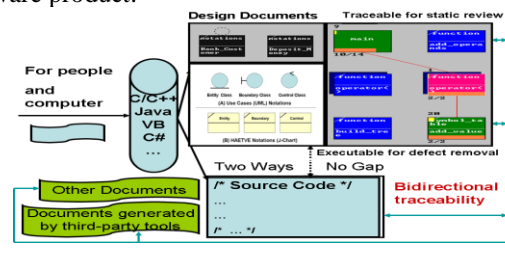


Fig. 5 NSM (Nonlinear Software Modeling approach)

The models and diagrams are automatically generated from their source code, either stub modules (having an empty body, or only a set of function call statements), or a regular program, through reverse engineering. The generated models, diagrams and the source code are fully traced. Further, dynamic design and coding are fully integrated in a non-linear way as shown in Fig. 6.

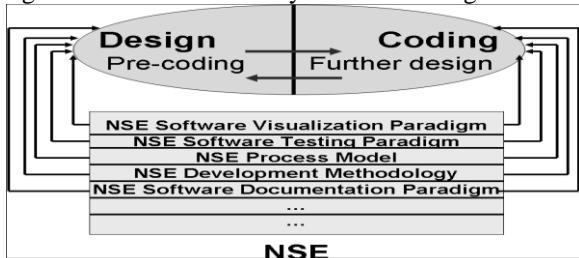


Fig. 6 Design and coding in NSE

NSE modeling methods are using 3J-graphics, and HAETVE techniques with Transparent testing box from running source code.

4. 3J-Graphics

Three types of models and diagrams are created called J-Chart, J-Diagram, and J-Flow.

Function	Non-member function	Press the right mouse button to pop up a function menu.
class	Member function	Press the right mouse button to pop up a member function menu.
Macro	Macro function	Press the right mouse button to pop up a function menu.
Function	Overloading non-member function	Press the right mouse button to pop up an overloading menu.
Class	Overloading member function and virtual function	Press the right mouse button to pop up a function menu.
Function	Overloading member function	Press the right mouse button to pop up a function menu.
Class	Virtual function	Press the right mouse button to pop up a function menu.
Class	Class	Press the right mouse button to pop up a class menu.
Class	Template Class	Press the right mouse button to pop up a class menu.

Fig. 7 J-Chart notations

4.1. J-Chart is not only used to represent class inheritance relationships, function call graphs, and the class-function coupling structure of a software product, but also is used to display the orders of incremental unit testing or related test coverage, where quality data in bar graphics is overlaid on each module-box. to show overall results of the test. J-Chart is easy for software modeling, system understanding, inspection, test planning, test result display, re-engineering, and software maintenance. J-Chart can be automatically generated from a stub program

of “Bone Programming” for high-level abstraction, or a regular program including legacy programs. J-chart notations are shown in Fig. 7.

4.2. J-Diagram notations are shown in Fig. 8. J-Diagram is automatically generated from source code in all levels, including class hierarchy tree, class structure diagram, and class member function logic diagram with un-executed class/function/segments/condition outcomes being highlighted. J-Diagram is automatically linked together for an entire software product to make the diagrammed code traceable in all levels. J-Diagram can be automatically converted into J-Flow diagram. J-Diagram is particularly useful in Object-Oriented software understanding, inspections, walkthroughs, testing, and maintenance.

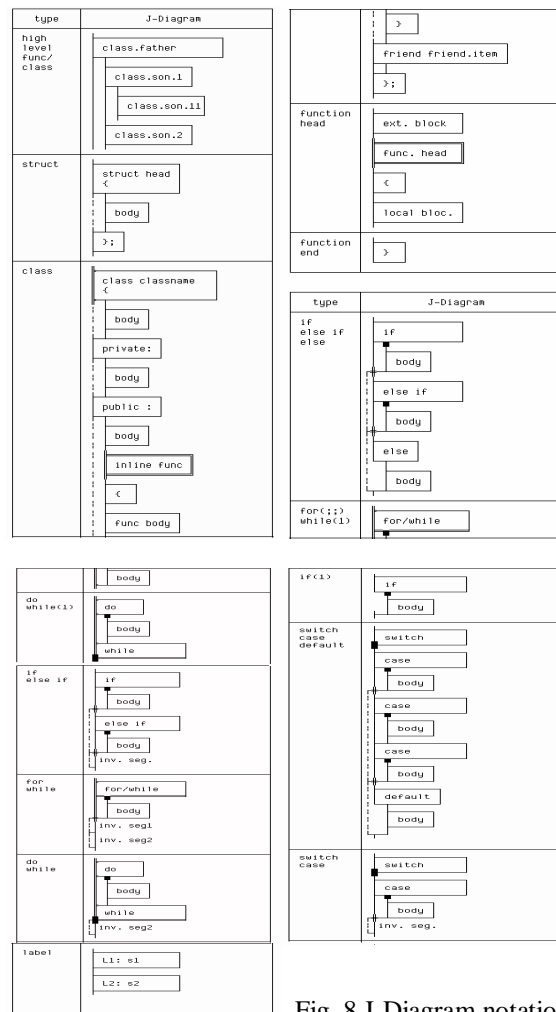


Fig. 8 J-Diagram notations

4.3. J-Flow The majority of traditional control flow diagrams are un-structured. They often use the same notation to represent different program logic, and cannot display logic conditions and source code locations. J-Flow diagram is Object-Oriented and structured. It uses different notations to represent different logic with capability to show logic execution conditions and corresponding source code locations. J-Flow is particularly useful in logic debugging, path analysis, test case, code correspondence analysis, and class/function level test. The test coverage

result is displayed with unexecuted elements (paths, segments, and unexecuted condition outcomes) highlighted.

J-Flow diagram in Fig. 9 can be converted to and from J-Diagram automatically. In NSM, J-Flow not only shows program control flow, but also shows the best path for testing on mostly untested branches. Its execution conditions are automatically extracted for semi-automatic test case design in unit testing.

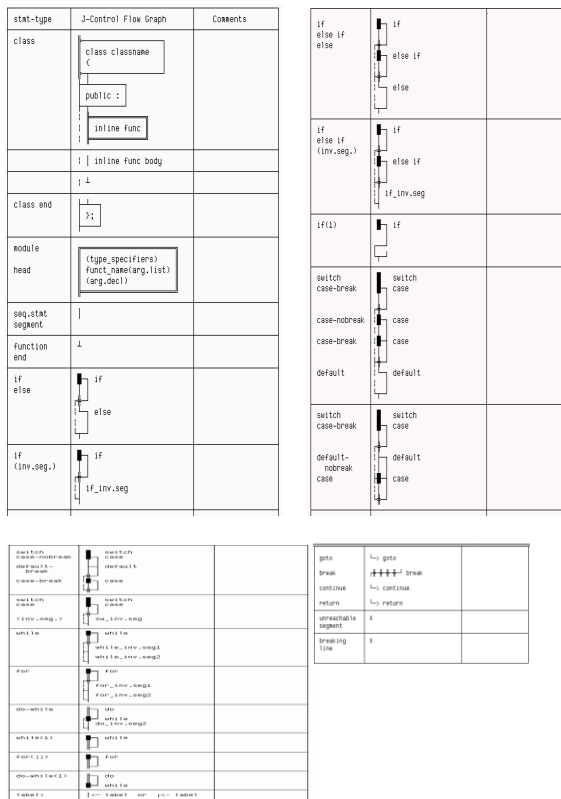


Fig. 9 J-Flow notations

J-Flow plays an important role in software traceability among all related documents, such as requirements specification, models, diagrams, test requirements, test cases and source code. All traceability operations use J-flow forwardly and backwardly. When a user selects a requirement and clicks a related test case in a window, that implementation is forwarded and that window will automatically show the test case in blue, while its corresponding test coverage result is shown in J-Flow in another window with its classes, functions, and branches are tested and highlighted in red.

The dynamic and interactive 3J-graphics, Object Oriented charts, logic diagram control flow diagram, and a chart generator, are a trinity. The trinity is always running when a chart is shown. 3J-Graphics are generated directly from the source code of the platform-independent Java programs or a platform-dependent program written in C, C++, or VB. With the 3J-Graphics and the corresponding tools and languages, high-level abstraction, such as Actor and Action notations similar to Use Case diagram of UML can be generated.

5. HAETVE [hayn-tiv] Technique

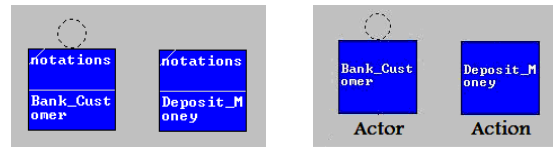
HAETVE means Holistic, Actor-Action and Event-Response driven, Traceable, Visual, and Executable techniques for dynamic software modeling. With HAETVE the graphical notations for representing an actor and its action using Java language are shown in Fig. 10 A, where the notation used for representing an actor is designed for representing a recursive program module below:

```
public class notations {
    public static void Bank_Customer ()
        { Bank_Customer (); }
    public static void Deposit_Money ()
        { } }
}
```

Java is a platform independent programming language. Results obtained in modeling from Java should be independent from target languages and platforms. If there is a need, the stub java source code can be transformed to a target language source code.

The notations for representing an actor and its action using C/C++ programs are shown in Fig. 10 B.

```
void Bank_Customer ()
{ Bank_Customer (); }
Void Deposit_Money ()
{ }
```



A. for Java

B. for C, C++

Fig. 10 Notations for representing an actor and its action model from different programming languages.

For the Actor-Action type applications, HAETVE is similar to Use Case approach [4], and is easy to map to Use Case notations as shown in Fig. 11 and Fig. 12.

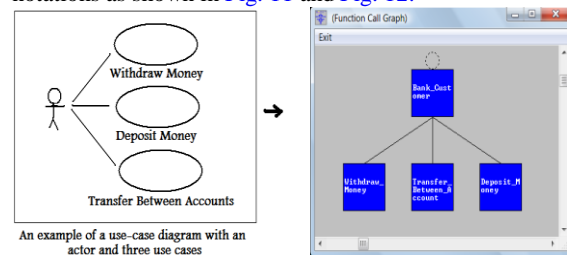
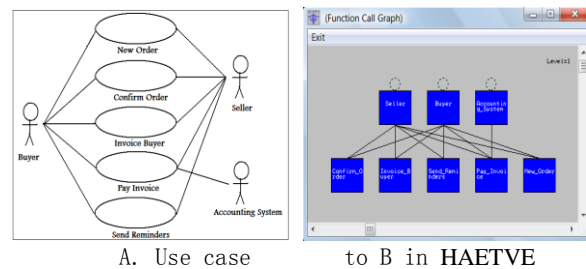


Fig. 11 Mapping from Use Case to HAETVE



A. Use case

to B in HAETVE

Fig. 12 Notation mapping from Use case A to HAETVE B.

The analysis result of Use Cases can also be mapped to HAETVE as shown in Fig. 13

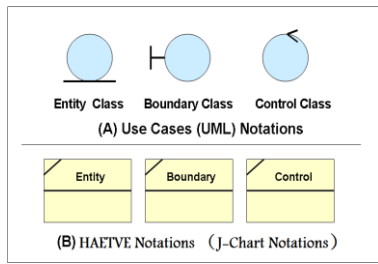


Fig. 13 Analysis notation mapping from Use Case (UML) to HAETVE.

A process example is shown in Fig. 14 and Fig. 15.

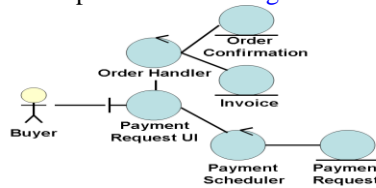
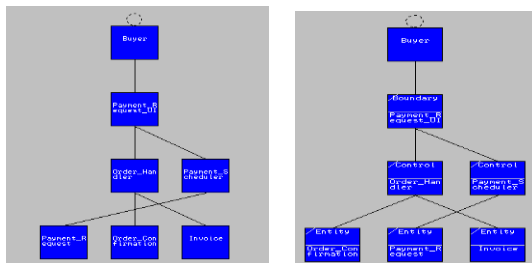


Fig. 14 A process example of Use Case Analysis



(A) Function notations (B) Class member notations
Fig. 15 Mapping to Use Case Analysis shown in Fig. 14

With HAETVE, event-responds notation is shown in Fig. 16.

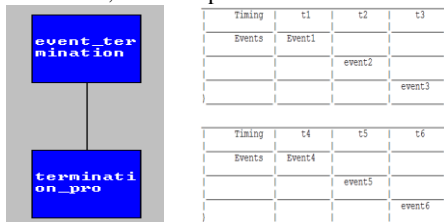


Fig. 16 Event-responds notation with event table in comments

A class is represented in J-Chart, J-Diagram, J-Flow diagram and Action-Plus diagram shown in Fig. 17.

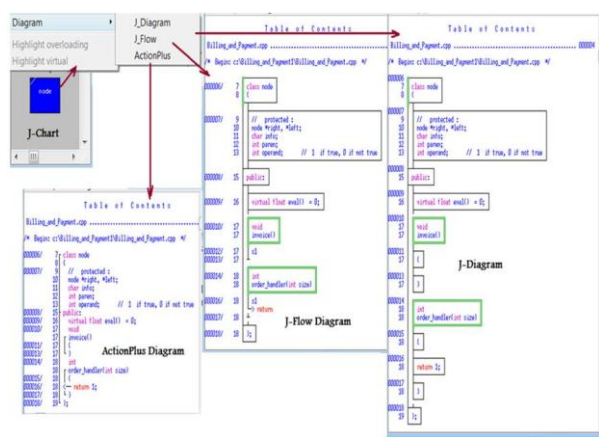


Fig. 17 A class is represented in four different notations.

A traditional Activity Diagram can be mapped to a combination of these notations and can be viewed in multiple windows.

Besides Actor-Action diagram, Event-Response diagram, and Activity diagram, all the models can also be automatically generated from a regular program or a legacy software product at all levels. It is a holistic solution.

5.1. Holistic HAETVE Models and diagrams can be generated from an entire software product with its source code written in platform-independent Java language or a platform-dependent programming language to show the product structure, class relationship, and overall static and dynamic properties of the product in Fig. 18 and comparative detailed views of two versions of the product shown in Fig. 19.

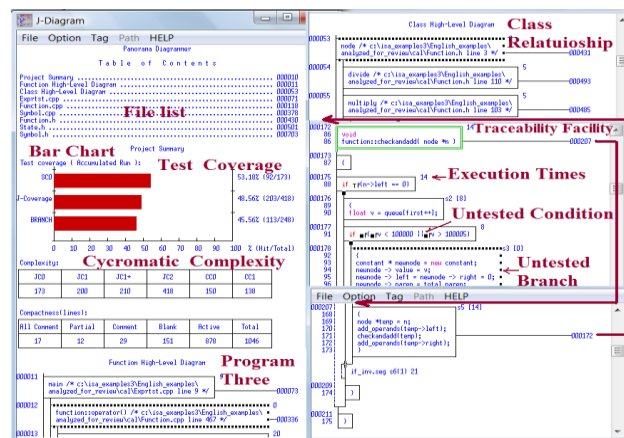


Fig. 18 A system level logic chart

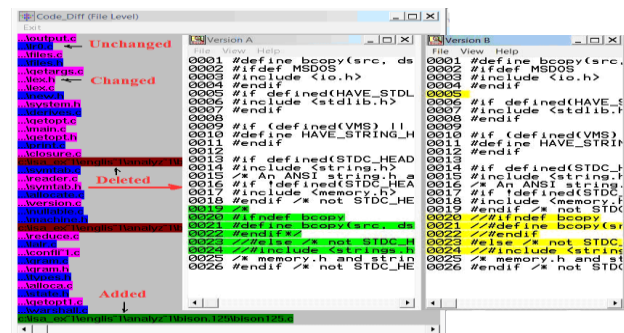


Fig. 19 A file level source code version comparison to guide code level understanding within the holistic system view.

5.2. Fully Automatable and Visual

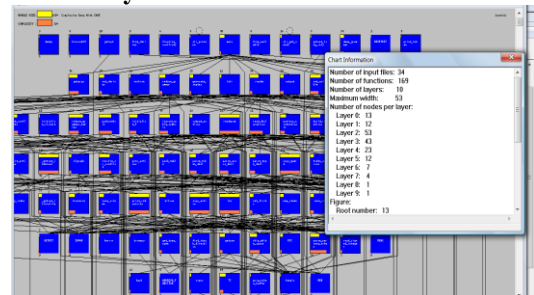


Fig. 20 A query on a model in a system level J-chart.

All models and diagrams can be automatically and completely generated from the source code as shown above in Fig.20. Many models and diagrams of UML are not automatable, such as Use Case diagram in Fig. 21, because it is not yet included in the software systems being developed.



Fig. 21 Actors of a Use Case diagram are outside the software product under development.

Unlike UML models, all parts of models and diagrams in HAETVE are inside the software system under development. Actors of the Actor-Action relationship diagrams are easy to take requirement validation and verification, and will not affect the program execution as shown in Fig.22, Fig. 23, and Fig. 24.

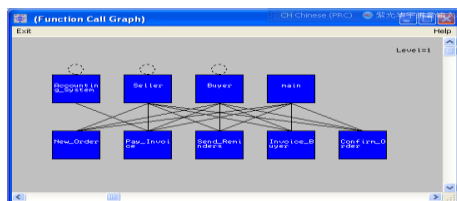


Fig. 22 HAETVE mapping to Use Case of Fig. 21.

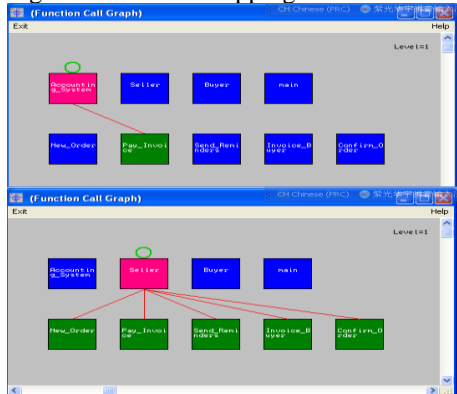


Fig. 23 Static check of the Actor-Action relationship.

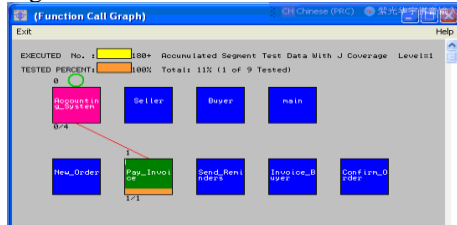
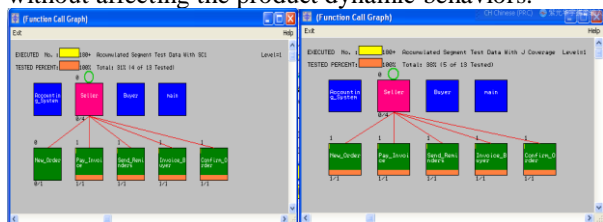


Fig. 24 Dynamic check on the actor-action relationship without affecting the product dynamic behaviors.



- (a)
 - (b)
- Fig. 25 Dynamic checks found an error and corrected:
- (a) An error – “New_Order” function does not executed
Source of error, a typing mistake:
`if(strcmp(argv[1], "New_Order")==0)`
`New_Order();`
 - (b) Corrected: Chang **New_Order** to **New_Order**.

HAETVE’s colorful screen representation includes model status and interactive response to user without penalty in speed.

5.3. Traceable and Executable models

Fig. 23-25 also show the model is traced back to source code to be able to identify and correct human errors after source code change. The traceability facilitates execution of the correction process.

With HAETVE, when a model is shown, the corresponding model generator is always working and waiting for users’ operation commands through the graphic interface. Users can request corresponding chart generator to show extra information such as the code test coverage, the percentage of the run time spent in each function. Fig. 26 shows how a user selects any function box as a new root to generate a sub-call graph from a J-Chart.

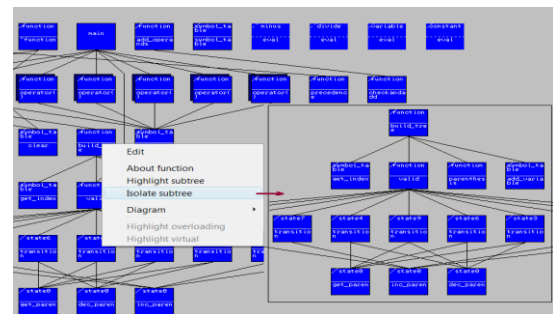


Fig. 26 An application example of the interactive J-Chart for generating a sub-call graph and shows the process to request the model generator to display the location where a runtime error happened (shown with an ‘EXIT’ word added) in the J-Flow diagram in Fig. 27.

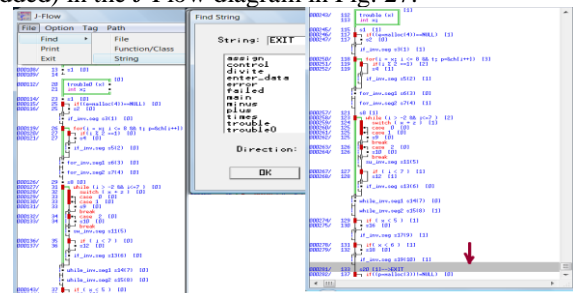


Fig. 27 The operation process for displaying the location where a runtime error happened.

If a diagram is better than a thousand words in describing a complex system, then an interactive and dynamic diagram will be much better than ten thousand words to represent a complex system.

6. Transparent-Box Method

As opposed to conventional black box or white box diagnosis method, NSE introduces a transparent testing box to combine functional testing and structural testing seamlessly and dynamically in the entire software development lifecycle. Fig. 28 shows bi-directional testing in every aspect of a software product.

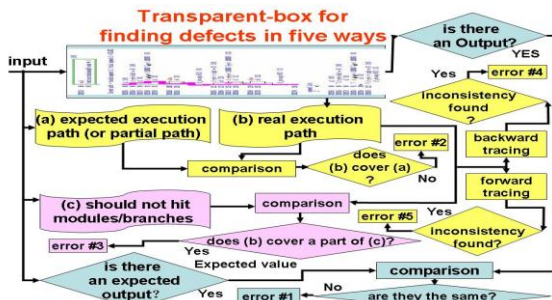


Fig. 28 Transparent-box testing method

The transparent testing box has made HAETVE implementation a reality. In addition, the fully implemented bi-directional traceability extends beyond a hosting product to incorporate a third party product with specific formats, or to play back captured GUI operations for regression testing. Fig. 29 and 30 shows a traceable UML model in Panorama++.

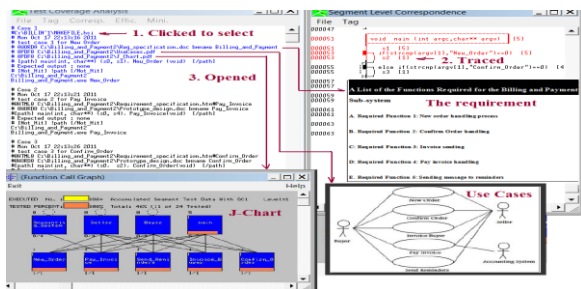


Fig. 29 An example of making UML models/Diagrams traceable from requirement

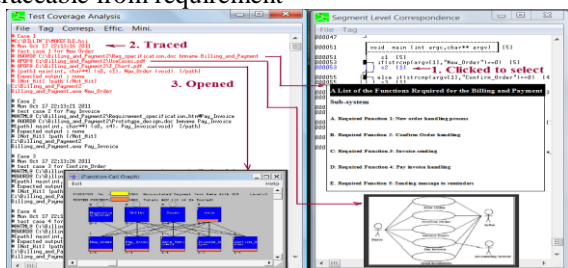


Fig. 30 An example of making UML models or Diagrams traceable from source code.

With bi-directional traceability, NSE supports software development, captures real time communication, manages multiple related projects, and fully supports UML modeling. Software maintenance can be performed nonlinearly, holistically, and globally, prevents side-effect from either requirement or code change using NSE Panorama++.

We have the right to be wrong, but we also have the right to be right: NSM makes design become pre-coding, and the coding becomes further design.

7. Conclusion

Existing linear software modeling approaches including MDE, MDA, and MDD are outcomes of linear thinking, reductionism, and the superposition principle that the whole of a complex system is the sum of its components. NSM automating software modeling brings revolutionary changes to software modeling by replacing manual modeling on legacy systems with automating model creation from the legacy source code, studying and improving these working models from the past, and incorporating new requirements into code generation, enabling software coding becomes further design.

NSM has been implemented, and fully supported by product Panorama++. The downloadable Panorama++ for C/C++ on Windows trial version, is on <http://www.NSEsoftware.com>.

"The next century will be the century of complexity" (Stephen Hawking, January 2000). Opportunities exist in developing platform-independent programming languages which are more suitable for high-level abstractions of a software product; Opportunities are also available in designing better tools to remodel legacy software products written in Cobol, ADA, or FORTRAN programming languages through nonlinear software modeling, etc. In NSM, it can be true, that "The Code is the Design".[7]

8 References

- [1] Thomas K`uhne, http://www.mm.informatik.tu-darmstadt.de/staff/kuehne_old/publications/papers/what-is-a-model-dagstuhl.pdf
- [2] Jim Arlow and Ila Neustadt in their book, "UML 2 and the Unified Process: Practical Object-Oriented Analysis and Design (Second Edition)", Person Education, Inc. 2006
- [3] Harry Sneed, *The Drawbacks of Model driven Software Evolution*, IEEE CSMR 07- Workshop on Model-Driven Software Evolution, Amsterdam, 20 Mar. 2007
- [4] Jay Xiong, *New Software Engineering Paradigm Based on Complexity Science*, Springer, March 2011 (<http://www.springer.com/physics/complexity/book/978-1-4419-7325-2>)
- [5] "An Analysis of Model Driven Architecture (MDA) and Executable UML (xUML)", http://www.powershow.com/view/30871-MjU5N/An_Analysis_of_Model_Driven_Architecture_MDA_and_Executable_UML_xUML_flash_ppt_presentation)
- [6] "Requiring Software Independence in VVSG 2007: STS Recommendations for the TGDC," Nov. 2006 <http://vote.nist.gov/DraftWhitePaperOnSInVVSG2007-20061120.pdf>
- [7] Jack W. Reeves, <http://developers.slashdot.org/story/05/03/01/2112257/the-code-is-the-design>

NSE Dynamic Software Documentation

- Software Documentation Revolution Based on Complexity Science

Liana Ye¹, Lin Li²

¹PeaceNames.com, USA

²NSESoftware, LLC, USA

Abstract

This paper introduces Nonlinear Software Engineering (NSE) Documentation Paradigm based on complexity science. By making software documents automatic, graphical, colorful, traceable, virtual, maintainable, and always consistent with source code dynamically, NSE brings revolutionary changes to software documentation.

Keywords: software documentation, software document, visualization, diagram, testing, maintenance

1. Introduction

eHow contributor O Paul pointed out that software documentation pervades the software life cycle. It is the visible part of the software process. Without it, software cannot be maintained, users are hard to train and the software is virtually unusable. Without it, new developers would have to re-invent the wheel in software development. Software documentation is the most important manifestation of software. It is the guide through the software maze [1].

Experience has shown that technical software documentation is unsatisfactorily maintained. The implications of outdated documentation are costly and can damage businesses (<http://www.sig.eu/en/Services/DocGen>).

Software documentation has been created manually with "cut and paste", or generated by a semi-automated tool for a large software product. Often the documents obtained with current software documentation techniques and tools are

- * not traceable,
- * not accurate,
- * not precise, and
- * not consistent with the source code, and
- * cannot be holistic.

This paper discusses a source code based software documentation approach based on complexity science solving all of the problems listed above efficiently, and bringing revolutionary changes to software documentations.

2. What Does a Revolution in Software Documentation Mean?

According to "The Structure of Scientific Revolutions" [2], a revolution in software documentation means:

- (1) Bringing drastic, complete, and fundamental change to the software documentation paradigm
- (2) Resolving some outstanding and generally recognized problems in software documentation;
- (3) There is no other way to efficiently resolve those outstanding and generally recognized problems in software documentation.

3. Documentation Paradigms

Documentation is a software production process, reflecting multiple levels of decision-making by people, especially software engineers. Fig.1 shows the mapping from requirements to source code production cycles in waterfall model on a reductionism base.

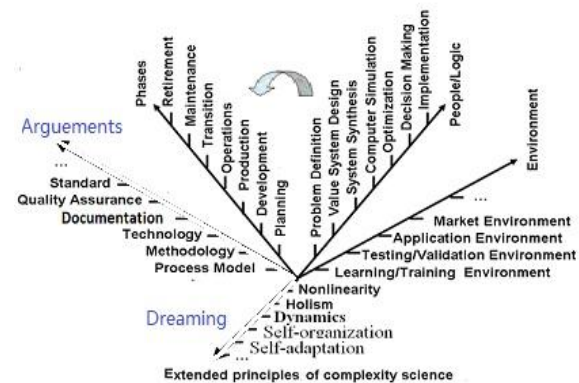


Fig.1. Three-dimensional software production cycles need documentation

Documentation is required for software development in the large arrow direction, where engineers are focused on coding and maintenance. Unconnected words indicate standardization efforts and numerous internal debates about software development.

3.1. Software documentation revolution condition (1)

NSE software documentation paradigm brings drastic, complete, and fundamental change to the software documentation paradigm by paradigm shift of the foundation of software documentation from the reductionism base of Fig 1, which is manually and partially traced, and its marketing, design and coding documents and specifications are strongly top-down typed and hard to

change. NSE is a complexity science based, innovative framework FDS shown in Fig. 1'.

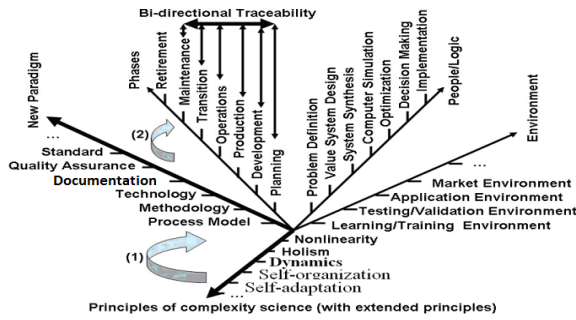


Fig. 1'. FDS (Five-Dimensional Structure Synthesis method) framework

3.2. Software documentation revolution condition (2):

The outstanding and generally recognized problems in software documentation are characterized by:

- (a) Reductionism and superposition principle: the whole of a complex system is the sum of its components, and almost all tasks and activities in software documentation are performed linearly, partially and locally.
- (b) Linear process models: where workflow goes linearly in one direction with only one track without counting upstream movement. It requires software developers to always document their software with no errors and no wrong decisions.
- (c) Constructive holism: the components of a complex system are completed first, then the system as a whole is assembled from its components.
- (d) Impossible to be holistic: many small pieces of documents are generated with no automated summation of the entire system, missing the big picture of the software product. Even if some tools can be used to document an entire software product, short of automated and self-maintainable traceability, its system level graphical documentation contain too many connection lines as shown in Fig. 2, making the documents hard to understand and almost useless.

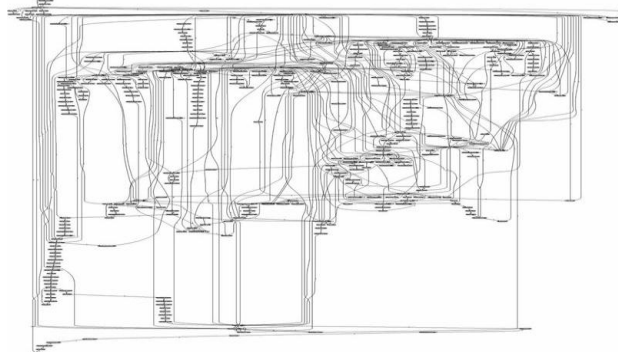


Fig. 2 A traditional call graph without traceability. Source: http://keithcu.com/bookimages/wordpress_html_m1e9af381.jpg

- (e) Graphic documents and source code are separated making it hard to keep them consistent, especially after product modification.

- (f) Documents are generally not traceable.
- (g) Documents obtained are stored statically as images in Postscript, XML, or other formats, requiring huge amounts of space and long loading times.
- (h) Manually or graphic editor created graphic documents are not automatically generated, time-consuming to draw, hard to change, and hard to maintain.
- (i) Documentation is error prone and may not be accurate.
- (j) The obtained documents are not precise. They can not directly and graphically show whether a code branch or condition combination has been tested or not.

3.3 Solution from NSE software documentation

- (a) Based on complexity science, particularly the nonlinearity principle and the holism principle, all software documentation tasks and activities are performed holistically and globally, as shown in Fig. 3.



Fig. 3 An example in NSE documentation on a complex software holistically

- (b) Nonlinear process model: Workflow goes nonlinearly through two-way interaction with multiple tracks, supporting both upstream and downstream movement as shown in Fig. 1.
- (c) Generative holism: The whole of a complex system comes first in embryonic form. It grows up with its components.
- (d) Holistic: Documents for an entire software product are automatically generated to make it easy to view the documents and understand internal connections, as shown in Fig. 4.



Fig. 4 An application example for NSE to trace a module and all its related modules upstream and downstream.

(e) Source code (either a stub program or a functional program) is the source for most graphical document generation. The graphic document is the visual face of the corresponding source code. They are always consistent with each other.

(f) Documents are traceable to and from the source code.

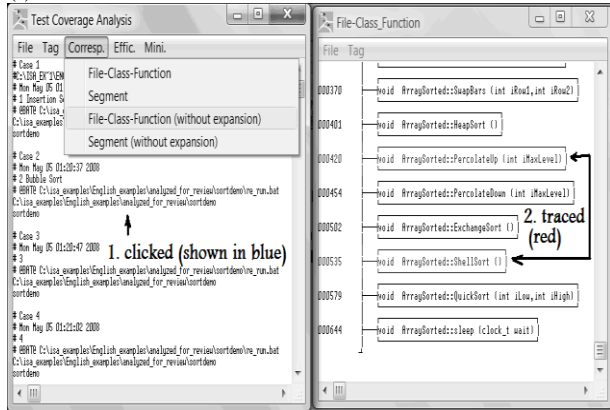


Fig. 5 Static visibility example – Forward tracing a test case to view which modules can be tested

(g) Documents are consistent with their source code after product modification with updates to database:

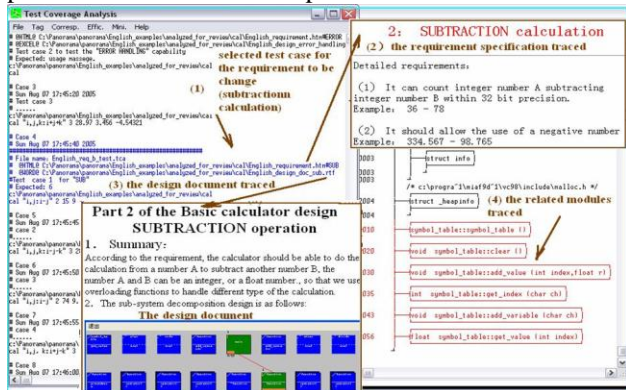


Fig. 6 Requirement Changes: Defect Prevention using forward tracing through the related test cases to determine which modules should be modified for a requirement change.

(h) Documents are dynamically generated directly from the source code. Corresponding databases are virtual without images in memory or on hard disk (with user option to print) to greatly save space and make the speed of display about 1000 times faster than established practice.

(i) Generated graphical documents are accurate to the code.
(j) Generated graphical documents are precise. They can directly and graphically show whether a code branch or condition combination has been tested or not.

NSE software documentation supports the entire software development process, from requirement development through software maintenance.

3.4. Software revolution condition (3): No other method can efficiently resolve those outstanding and generally recognized problems in software documentation except generating documents automatically from source code.

Although there are many software documentation tools available on the market, they are disjoint tool sets. Alistair Cockburn, characterizing people as non-linear, first-order components in software development: “We methodologists and process designers have been designing complex systems without characterizing the active components of our systems, known to be highly non-linear and variable (people).” <http://alistair.cockburn.us/>

From source code to generated documents, NSE produces automatically generated documentation from source code, displays coded decisions made by people keeping bi-directional traces for every symbolic connection in the system, updates function call statements or testing conditions dynamically with graphical representation for highly non-linear and variable people, with precise quantitative information about how many requirements may be affected and how many function call statements may need to be modified. **We see no other way to efficiently resolve previously mentioned outstanding and generally recognized problems but to take automatically generated documentation from source code.**

4. Documentation in NSE is Dynamic

Source code is the ultimate software system documentation. Complete graphical relationship representation of each component is determined when the source code is compiled. NSE collects all information on the software system, then generates holistic, graphical, interactive, and traceable documents automatically, including a stub program meeting development and product specifications and a functioning program for forward engineering or reverse engineering.

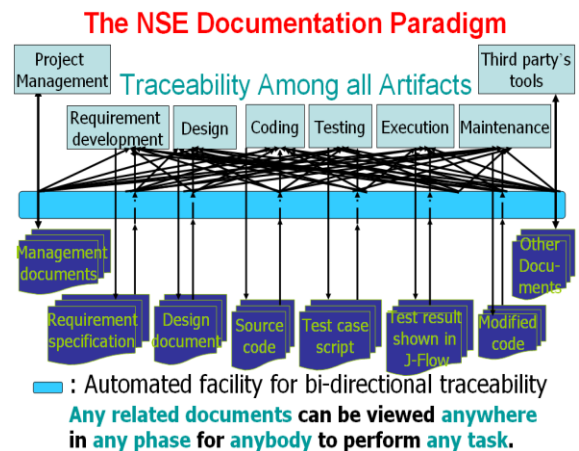


Fig. 7 Traceability among all related documents, test cases, and source code

As shown in Fig. 7, source code is the source for automatically generating graphic design documents, while those graphic design documents become visual faces of source code. The design becomes pre-coding, and coding becomes further design. The graphical documents are traceable for static review and executable for dynamic defect removal, as shown in Fig. 4.

Self-maintainable documentation facility is established through dynamic traceability among design documents, test cases and the source code, as shown in Fig. 7. After the source code is altered following a requirement change, the design documents and modified source code are automatically incrementally updated, facilitating defect prevention, as shown in Fig. 6.

4.1. Workflow in NSE Documentation Paradigm

Workflow for NSE software documentation paradigm consists of source code of regular programs and stub code of design documents:

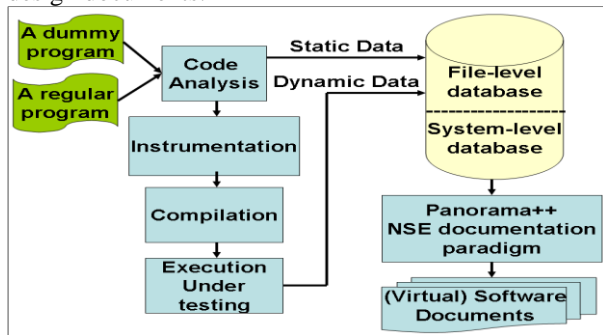


Fig. 8 The workflow of NSE documentation paradigm

The objectives of NSE documentation paradigm are:

- (a) combining software programming and graphical software documentation seamlessly;
- (b) making one source for both human understanding and computer “understanding” - through static review by people of the graphical documents and dynamic program execution to ensure the upstream quality of a software product.
- (c) realizing all kinds of documents (both manually drawn and generated by third party tools) traceable to source code to keep them consistent with each other through the established transparent-box method, combining functional testing and structural testing together seamlessly with the capability of establishing bidirectional traceability.
- (d) generating most software documents automatically, as much as possible.
- (e) making software documents visible, as much as possible.
- (f) making a software product truly maintainable and adaptive to the changed or changing environment.

4.2. NSE software visualization notations

A majority of NSE documentation paradigm is visualization in 3J (J-Chart, J-Diagram, and J-Flow diagram) graphics. For example, Classes, using HAETVE model, are represented in several graphical notations: J-chart,

ActionPlus Diagram, J-flow and J-Diagram as shown in Fig. 9.

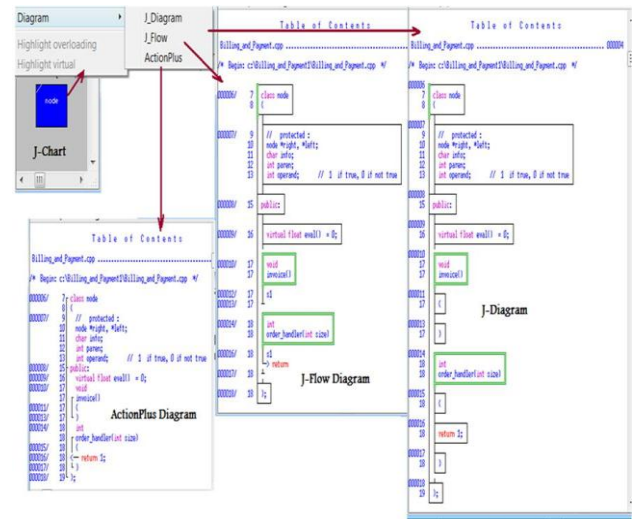


Fig. 9 NSE Graphical representation for a class

Time-Event table documents testing events

Time-event tables are written in the comment part of a stub program or regular program. An example is listed below:

/* Time-Event table:

Timing	t1	t2	t3
Events	Event1		
		event2	
			event3

Timing	t4	t5	t6
Events	Event4		
		event5	
			event6

*/

4.3. J-Diagram documents message sending & receiving

Graphically representing message sending and receiving in the automatically established “click-to-jump” facility as shown in Fig. 10.

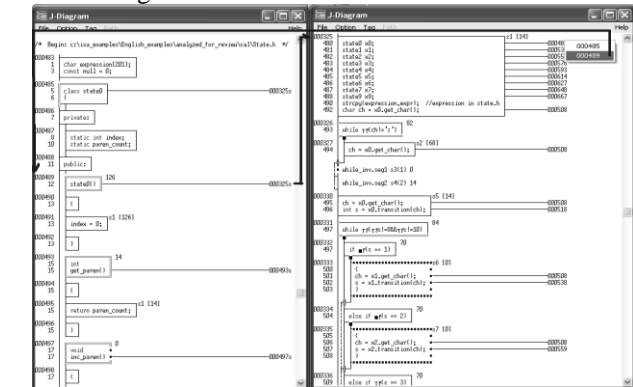


Fig. 10 Click-to-jump facility automatically established for showing message sending and receiving.

4.4. Software product is visible in multiple views

- (a) Static view of the cyclomatic complexity of a program.

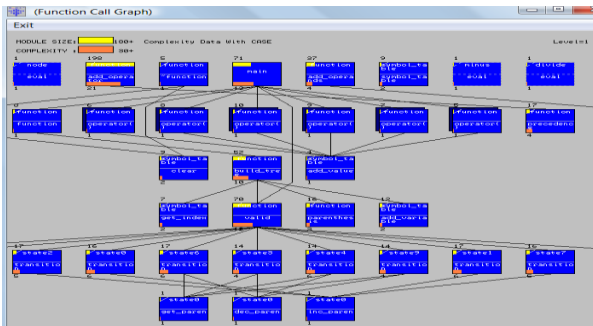


Fig. 11 An example of Cyclomatic complexity analysis

(b) Dynamic view of a program

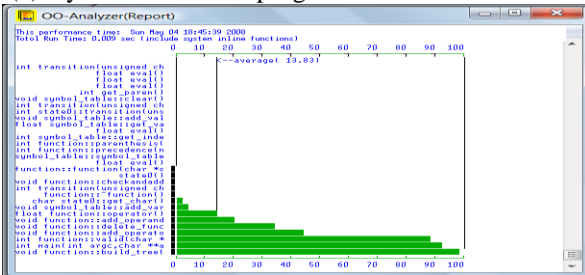


Fig. 12 An example of performance analysis

4.5. From macro to micro with MC/DC test coverage

(a) Holistic analysis for an entire product, Fig. 13:



Fig. 13 Holistic MC/DC (Modified Condition/Decision Coverage) test coverage analysis

(b) Detailed MC/DC test coverage analysis for an individual class or function, Fig. 14.



Fig. 14 An application example of detailed test coverage analysis of a module

4.6. From procedure to data

(a) Function cross-reference analysis, Fig. 15.

The screenshot shows two side-by-side windows from 'OO-Analyzer(Report)'. Both windows display a table of function cross-references. The left window is titled '34. Function Cross-reference: Call to' and the right is '35. Function Cross-reference: Called from'. Both tables have columns for 'Function Name', 'Call To', 'In File', and 'Line'.

Fig. 15 an example of function cross-reference analysis

(b) Data analysis, Fig. 16.

The screenshot shows two side-by-side windows from 'OO-Analyzer(Report)'. Both windows display a table of variable analysis. The left window is titled '36. Used Global Variable Report' and the right is '37. Used Static Variable Report'. Both tables have columns for 'Variable Name', 'Type', and 'Related File/Function'.

Fig. 16 An application example of variable analysis

4.7. From System level to file level to statement level

(a) System-level version comparison, Fig. 17.



Fig. 17 An example of holistic version comparison at system level.

(b) File-level version comparison, Fig. 18.

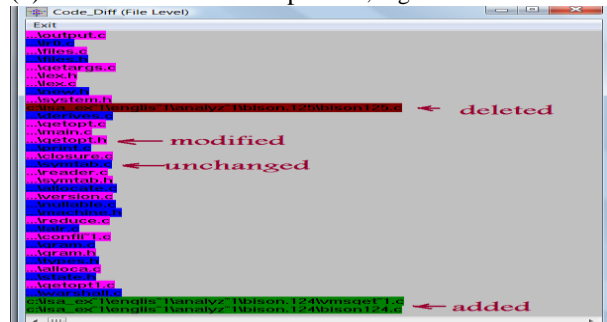


Fig. 18 An example of file-level version comparison

(c) Statement version comparison, Fig. 19.

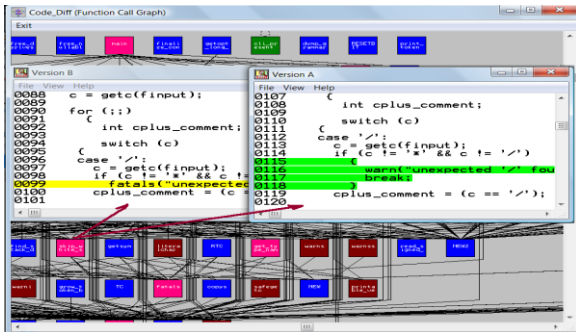


Fig. 19 An example of statement version comparison

4.8. Dynamic visibility: Tracing a test case not only to find what modules can be tested, but also to directly play the captured test operations back through the batch file (.bat) specified in the @BAT@ keyword within the test case description part – see Fig. 20.

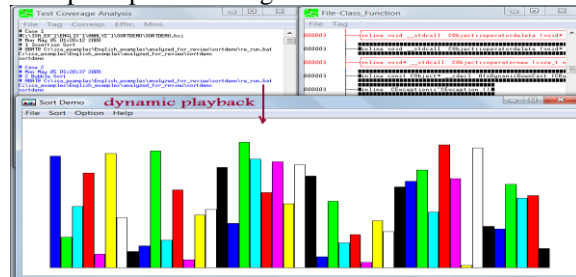


Fig. 20 Dynamic visibility – tracing a test case to play the captured operations back

4.9. Interactively traceable

With NSE the generated documents are interactive – for instance, the user can click on a module-box to use that module as a root to generate a sub-call-graph, Fig. 21.

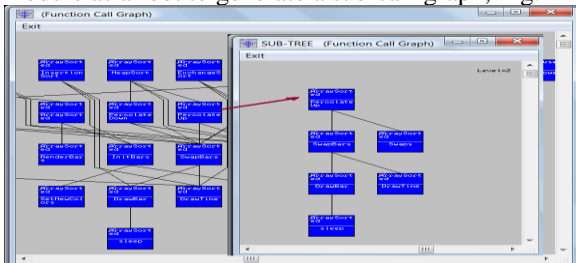


Fig. 21. Interaction example: click on a module-box to generate an isolated sub-call-graph

Most of the generated documents are traceable, Fig. 22.

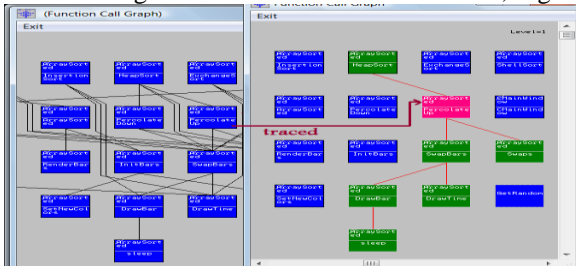


Fig. 22 Tracing a module to see all the related modules

4.10. Linkable and convertible

With NSE, different graphical documents can be linked together - see Fig. 23. Generated logic diagrams can be converted to control flow diagrams – see Fig. 24.

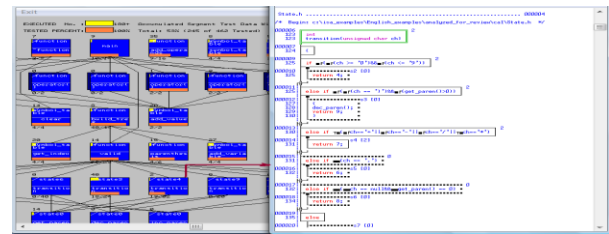


Fig. 23 An application example - linking a call graph to the logic diagram

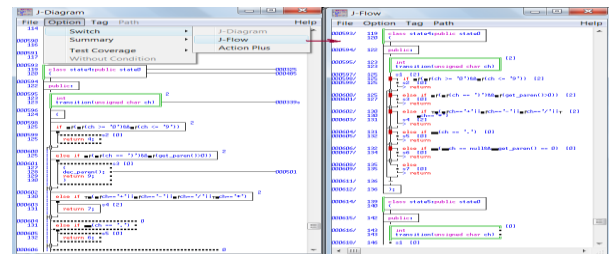


Fig. 24 An example of diagram conversion from a logic diagram to control flow diagram

4.11. Local documentation to Internet

With NSE, many static and dynamic analysis reports can be automatically generated as in Fig. 25.

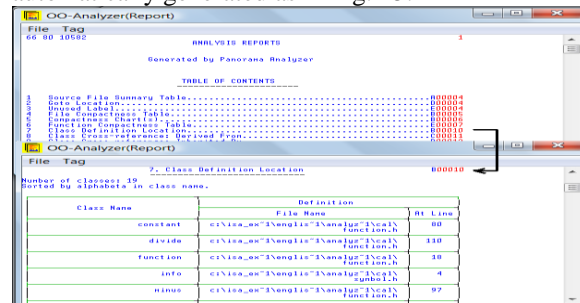


Fig. 25 An application example of static and dynamic program analysis and reporting

Generated reports for static and dynamic program analysis can be saved in HTML format to be used as web pages, Fig. 26.

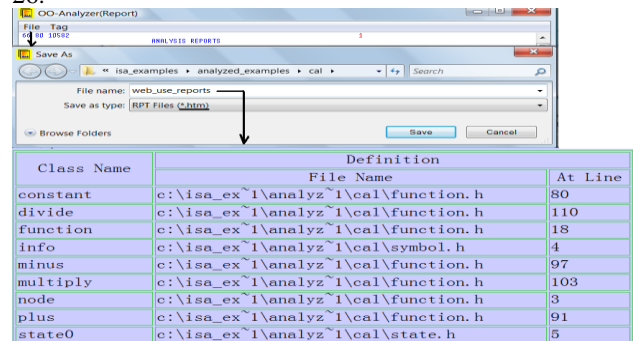
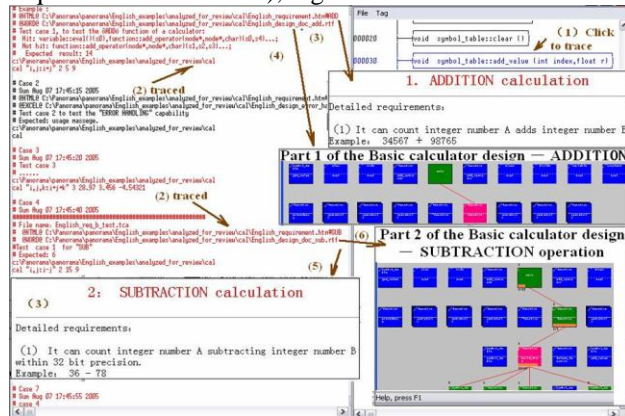
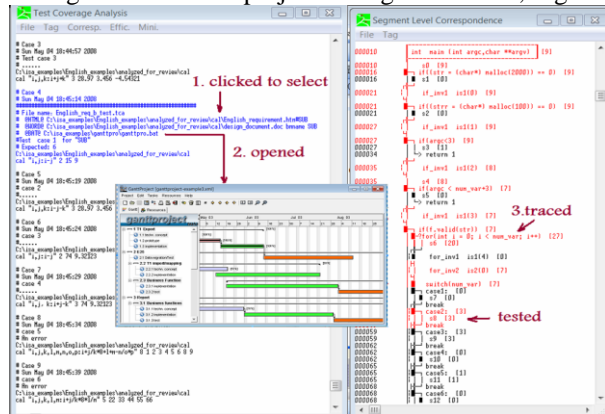


Fig. 26 Code analysis reports saved in HTML format to be used as web pages

4.12. Preventing requirement conflicts: Perform backward tracing to the modified modules (in this example, two requirements are related), Fig. 28



4.13. Project management: Through keywords used in testing traced back to project management view, Fig. 29



5. Major Features of NSE Documentation Paradigm

The graphical documents generated by NSE documentation paradigm are:

- **Holistic** – NSE documentation paradigm generates holistic charts and diagrams to document an entire software product.
- **Interactive** – the generated graphical documents are interactive, the generated charts/diagrams themselves are also the interfaces to accept user's commands.
- **Traceable** – with NSE most of the generated documents are traceable, useful for validation, verification, and semi-automated inspection and walk through,
- **Accurate** – Source code to NSE, including a stub program, is also the source to automatically generate most graphical documents, so that the generated documents are accurate and consistent with the source code.
- **Precise** – the generated graphical documents are precise, and the corresponding documents can show how many times a branch is executed, and what code branches and conditions have not been executed.
- **Virtual** – with NSE, most graphical documents are

dynamically generated from source code, so there is no need to save their hard copy images in memory or disk, so that a huge amount of space can be saved. The display speed is about 1000 times faster compared with established practice under comparable and similar operational environment. The generated holistic charts and diagrams are shown within one window, no more and no less. When scrolling occurs, the new portion of the chart will be generated dynamically. From a users' point of view, there is no difference between virtual charts and the regular charts occupying a huge amount of space in computer memory.

• **Massive** – the graphical documents are generally about 100 times the size in disk space of their source code. It can be automatically generated at system-level, file-level, and module-level. For each class or function, NSE documentation paradigm automatically generates the logic diagram shown in J-Diagram notation with untested branches and untested conditions highlighted, control flow diagram shown in J-Flow diagram notations, quality measurement result shown in Kiviati diagram, etc. The graphic display capability is massive.

6. Application

NSE documentation paradigm has been commercially available and supported by Panorama++. All screenshots shown in this paper come from real application examples.

7. Conclusions

The established software documentation paradigm based on reductionism and superposition is outdated. NSE software documentation paradigm resolves the outstanding and generally recognized problems in software documentation, and causes a paradigm shift to complexity science based principles, making the whole of a complex system greater than the sum of its components. The characteristics and behaviors of the whole emerge from interaction of its components.

Source code is not the best documentation of a software product, but **source code is the best source to directly and automatically generate holistic, interactive, traceable, consistent, accurate, precise, massive, and graphical documentations for the software concerned.** There is no way to solve the problem of inconsistency in documentation except to generate documentation directly from source code. Panorama++ demonstrates the feasibility of such an approach, turning software reverse engineering into a software documentation revolution.

8. References

[1] (http://www.ehow.com/about_6706857_importance-software-documentation.html).

[2] Kuhn T (1962) The structure of scientific revolutions. The University of Chicago Press, Chicago

SESSION

SOFTWARE ENGINEERING RESEARCH AND PRACTICE: NOVEL SYSTEMS AND METHODS

Chair(s)

Prof. Hamid R. Arabnia

Early Usability Evaluation in Model-Driven Video Game Development

Adrian Fernandez, Emanuel Montero, Emilio Insfran*, Silvia Abrahão, and José Ángel Carsi
ISSI Research Group, Department of Information Systems and Computation
Universitat Politècnica de València, c/ Camino de Vera, s/n 46022 Valencia, Spain
{afernandez, emontero, einsfran, sabrahao, pcarsi}@dsic.upv.es

Abstract—Usability is considered a relevant quality factor in video games. However, usability evaluations are usually performed too late in the game development lifecycle. We present a usability evaluation strategy that can be used in early stages of model-driven video game development approaches. The usability evaluation is based on a Video Game Usability Model, which extends the usability characteristic of the ISO/IEC 25010 (SQuaRE) standard by incorporating measurable attributes and measures related to the video game domain. The traceability established between the models that are produced in a model-driven development process and the corresponding source code allows performing usability evaluations on these models, facilitating the early detection/correction of usability problems that may appear in the final video game application. To show the feasibility of this approach, we have performed an early usability evaluation of a video game for the XBOX360 platform.

Keywords: Video Game, Usability Model, Usability Evaluation, Model-Driven Video Game Development.

I. INTRODUCTION

The video game development industry is a strong economic sector that deals with the development of highly interactive software, i.e., video games, for a wide variety of technology platforms such as PCs, consoles, Web browsers, and mobile devices. The interaction between the game and the players is a critical factor in the success of a video game.

Usability and playability are considered to be the most important quality factors of video games [15]. *Usability* is defined as the degree to which the video game can be understood, learned, used and is attractive to the user, when used under specified conditions [11]. *Playability* is defined as a collection of criteria with which to evaluate a product's gameplay or interaction [12]. Playability is often evaluated by using early prototypes and iterative cycles of playtesting during the entire video game development cycle. However, the evaluation of usability is deferred to late stages in the game development cycle, thus signifying that usability problems from early stages may be propagated to late stages of the development, and consequently making their detection and correction a very expensive task.

Traditional video game development approaches do not take full advantage of a usability evaluation of the game design artifacts that are produced during the early stages of the development. These intermediate artifacts (e.g., screen mock-ups or screen flow diagrams) are used to guide game

developers but not to perform usability evaluations. Moreover, since the traceability between these intermediate artifacts and the final video game is not well-defined, performing usability evaluations by considering these artifacts as input can be a difficult task. This problem may be alleviated by using a model-driven development approach due to its intrinsic traceability mechanisms that are established by the transformation processes. Platform-independent models (PIM) such as screen flow diagrams may be transformed into platform-specific models (PSM) that contain specific implementation details of the underlying technology platform. These platform-specific models may then be used to generate the source code of the video game (Code Model – CM), thus preserving the traceability among platform-independent, platform-specific and source code artifacts.

A model-driven video game development approach therefore provides a suitable context for rapid iteration early in the development cycle. Platform-independent (or platform-specific) models can be evaluated during the early stages of video game development to identify and correct *some* of the usability problems prior to the generation of the source code of the final video game application. We are aware that not all the usability problems can be detected based on the evaluation of models since they are limited by their own expressiveness and, most important, they may not predict the user behavior and preferences. However, studies such as the one by Hwang and Salvendy [10] claims that usability inspections, applying well-known usability principles on software artifacts, would be capable to find around 80% of usability problems. In addition, as suggested by previous studies [4], the use of inspection methods for detecting usability problems in product design (models in our context) can be complemented with other evaluations performed with end-users before releasing a video game to the public.

In this paper, we present a usability evaluation strategy that can be used in early stages of model-driven video game development. This strategy is based on a Video Game Usability Model which decomposes the usability characteristic proposed in the ISO/IEC 25010 (SQuaRE) standard [11] with new usability attributes for the video game domain. These attributes are quantified through their association with generic measures that can be operationalized by establishing a mapping between their generic definition and the specific modeling primitives of the software artifacts to be evaluated. This allows our Video Game Usability Model to be used not

*Contact author. This paper was submitted to the Int. Conference on Software Engineering Research and Practice (SERP 2012)

only in model-driven video game development processes but also in any other video game development process (e.g., traditional, agile).

This paper is organized as follows. Section 2 discusses usability evaluation techniques for video game development. Section 3 describes the Video Game Usability Model. Section 4 proposes a strategy to apply this model for performing early usability evaluations in model-driven video game development. Section 5 presents a case study to illustrate the approach. Finally, Section 6 presents our conclusions and further work.

II. RELATED WORK

The state of the art for game development in software engineering has been recently summarized in a systematic literature review [3]. The results of this review show a significant lack of studies in the key dimensions of video game quality: playability and usability. However, some efforts have been made to integrate current usability evaluation techniques into the game development industry and game research, and a brief review of current game usability techniques has been provided in [15]. Usability evaluation techniques can principally be classified into two groups: empirical techniques and inspection techniques.

Empirical techniques are based on capturing and analyzing usage data from real players. Some representative examples are *think-aloud* techniques and *focus group* techniques [8]. In *think-aloud* techniques, the player sits down to play the video game and narrates his experiences while a user experience evaluator sits nearby listening and taking notes. In *focus group* techniques, game developers gather a small group of potential game players together to discuss their opinions of the design of the interface, along with the game mechanics and story.

Inspection techniques, which have emerged as an alternative to empirical methods, are performed by expert evaluators or game designers and are based on reviewing the usability aspects of software artifacts (which are commonly game user interfaces) with regard to their conformance with a set of guidelines. The most representative example is *heuristic evaluation*, which is a common inspection method for evaluating the usability of video game interfaces in both early and functional game prototypes. Examples of heuristic evaluation techniques were presented in the work of Federoff [6] and Pinelle *et al* [17], in which a set of guidelines for creating a good game were defined, based on the experience of a game development case study, and PC game reviews, respectively.

In this paper, we focus on usability inspection techniques since they do not involve the players' participation and can be employed during the early stages of the game development process. In addition, current approaches that are based on heuristic evaluations are too generic and dependent on the evaluator expertise, and in most cases, result solely in a plain checklist of desired features with no specific guidelines on how they can be applied. In order to minimize, at least to some

extent, the degree of subjectivity that appears in the majority of inspection methods for video games, we propose a usability inspection technique based on the use of a Video Game Usability Model in a model-driven development context. In this way, we provide specific video game attributes and measures that can be quantified automatically by means of model-transformations. Model-driven development provides a suitable context for early usability evaluations since traceability between high-level software artifacts (models) and source code is maintained throughout the development process [1]. The evaluation of these high-level artifacts during the early stages of development is a means to detect and correct problems that may appear in the final software product.

Finally, approaches based on usability models have been successfully employed as inspection techniques with which to evaluate software artifacts in other domains, such as model-driven software development [2] and model-driven Web development [7]. However, as far as we know, no usability model has been applied to model-driven video game development.

III. DEFINING THE VIDEO GAME USABILITY MODEL

Since the usability concept has not been homogeneously defined in the literature, we use the ISO/IEC 25010 (SQuaRE) standard [11] as the basis for defining our Video Game Usability Model. In the SQuaRE standard the usability of a software product can be decomposed into the following sub-characteristics: *Appropriateness*, *Recognisability*, which refers to how the software product enables users to recognize whether the software is appropriate for their needs; *Learnability*, which refers to how the software product enables users to learn its application; *Ease of Use*, which refers to how the software product makes it easy for users to operate and control it; *Helpfulness*, which refers to how the software product provides help when users need assistance; *Technical Accessibility*, which refers to how the software product provides help when users need assistance; and *Attractiveness*, which refers to how appealing the software product is to the user.

However, these sub-characteristics are too abstract to be directly measured in a video game development context. We therefore propose the decomposition of these sub-characteristics into more representative and measurable attributes of video games, and the subsequent decomposition of each one of these attributes into specific measures, which can be calculated depending on the characteristics of the artifact to be evaluated.

A. Usability Attributes for Video Game Usability

The decomposition of the sub-characteristics into attributes is presented as follows, and is summarized in the second column of Table I. These attributes have been defined by considering and adapting both the knowledge gained from other domains such as Web development [5],[7], and the underlying usability principles from game development knowledge [13],[16].

The attributes defined for the sub-characteristics are:

- **Appropriateness Recognisability** contains all the attributes of the video game that ease the understanding of the game. This sub-characteristic is decomposed into the following attributes: *Visibility*, which focuses on visual recognisability, and legibility by measuring the ease of perception of the game's graphic information; *Interface Simplicity* and *Control Simplicity*, which evaluate the complexity of the graphical user interface and the game controls, respectively; and *Consistency*, which focuses on the degree of similitude and coherence between the elements of the video game.
- **Learnability** contains the attributes of the video game that allow players to learn how to play the game. This sub-characteristic is decomposed into the following attributes: *Feedback support*, which focuses on the game capability to provide information about the current state of the game and its players; and *Tutorial Support*, which verifies whether the game offers a tutorial to teach the players how to play it.
- **Ease of Use** contains all the attributes of the video game that facilitate players' control and operation, both inside and outside gameplay. This sub-characteristic is decomposed into the following attributes: *Control Consistency*, which refers to the degree of semantic similitude of the players' actions with regard to the game controls (i.e., mapping similar concepts onto the same control element to facilitate learning); *Internal Navigational Simplicity*, which refers to how to navigate between the menu options of a single screen; and *External Navigational Simplicity*, which concerns how to navigate between game screens.
- **Helpfulness** contains all the attributes of the video game that provide help when the players need it. Most video games lack a help option, and players must rely solely on eventual hints and goals. This sub-characteristic is decomposed into the following attributes: *Hint Support*, which refers to the game's capability to provide useful hints with which to guide the players; and *Goal Support*, which refers to the video game's capability to provide clear goals for the players to pursue.
- **Technical Accessibility** contains all the attributes that allow physically impaired users to play the video game. This sub-characteristic is decomposed into the following attributes: *Subtitle Support*, which refers to the game's capability to provide adequate subtitles for hearing impaired players; and *Magnifier Support*, which concerns the game's capability to provide adequate sized subtitles for visually impaired players.
- **Attractiveness** contains all the attributes that make a video game more appealing to the players. This sub-characteristic is decomposed into the following attributes: *Customization*, which refers to how players

can alter the game's graphical user interface and controls to fit their preferences; and *Wait Reduction*, which refers to the degree of inactive waiting the players are forced to undergo.

TABLE I. DECOMPOSITION OF THE SQUARE INTO MEASURABLE ATTRIBUTES AND GENERIC MEASURES

Sub-characteristics	Attributes	Measures
Appropriateness Recognisability	Visibility	Percentage of Screen Usage
	Interface Simplicity	Total Number of GUI Elements
	Control Simplicity	Total Number of Control Mappings
	Consistency	Ratio of Similitude Between Screens
Learnability	Feedback	Total Number of GUI Elements Displaying State Changes
		Ratio of GUI Elements Highlighting State Changes
		Ratio of Meaningful Messages
	Tutorial Support	Tutorial Interactivity Tutorial Coverage
Ease of Use	Control Consistency	Ratio of Similitude Between Colliding Game Actions
	Internal Nav. Simplicity	Internal Menu Navigation Depth Internal Menu Navigation Breadth
	External Nav. Simplicity	Shortest Path To Gameplay
		Shortest Path To Exit Shortest Return Path To Gameplay
Helpfulness	Hint Support	Availability of Hints Hint Understandability
		Goal Support
	Technical Accessibility	Subtitle Support
Magnifier Support		
Attractiveness	Customization	Control Remapping Interface Customization
		Wait Reduction

B. Generic Measures for Video Game Usability

Once the measurable usability attributes have been identified, generic measures are then associated with these attributes in order to quantify them. The measures are generic in order to ensure that they can be operationalized in different software artifacts (from different abstraction levels) from different video game development methods. The values obtained from the measures will allow us to determine the degree to which these attributes help to achieve a usable video game.

Due to space constraints, a subset of the proposed measures from the Video Game Usability Model is presented in the third column of Table I. Then, some of these measures are described in more detail in Table II.

IV. APPLYING THE VIDEO GAME USABILITY MODEL

In order to apply the Video Game Usability Model to a specific video game development, we propose a usability evaluation strategy. A typical video game development process consists in the following activities: requirements specification, game design, implementation, and playtesting, along with the usability evaluation.

The usability evaluation is conducted by applying the following three steps:

TABLE II. SUBSET OF PROPOSED MEASURES FROM THE VIDEO GAME USABILITY MODEL

Measure	Percentage of Screen Usage (PSU)
Attribute	Appropriateness Recognisability / Visibility
Description	Percentage of screen covered with GUI elements
Formula	Sum of all GUI elements size / screen size
Scale	Real value between 0 and 1
Interpretation	Values near 1 indicate that the GUI covers the entire screen, leaving no room for gameplay elements, thus making the video game difficult to understand

Measure	Total Number of GUI Elements (TNGUIE)
Attribute	Appropriateness Recognisability / Interface Simplicity
Description	Total number of elements of the graphical user interface (UI) on a game screen
Formula	Sum of all GUI elements on the screen
Scale	Integer greater than or equal to 0
Interpretation	Lower values indicate that the GUI has fewer elements, resulting in a simple UI.

Measure	Total Number of Control Mappings (TNCM)
Attribute	Appropriateness Recognisability / Control Simplicity
Description	Total number of control elements that players can use to perform an action in the game
Formula	Sum of all the controls in the game
Scale	Integer greater than or equal to 0
Interpretation	Lower values indicate that the control mapping has fewer elements, resulting in a simple control schema which is easy to understand

Measure	Shortest Path To Gameplay (SPTG)
Attribute	Ease of Use / External Navigational Simplicity
Description	Minimum number of screens that players have to navigate in order to start playing.
Formula	Minimum number of steps between the initial screen and the gameplay screen
Scale	Integer greater than or equal to 0
Interpretation	A value of 0 signifies that the game has no menu screens, and begins directly at gameplay. Higher values indicate that players have to navigate various screens before the game starts. If the value is too high, players may get anxious before reaching gameplay as a result of the navigational complexity

Measure	Shortest Path To Exit (SPTE)
Attribute	Ease of Use / External Navigational Simplicity
Description	Minimum number of screens that players have to navigate in order to exit from the game when playing
Formula	Minimum number of steps needed to exit from the game via the gameplay screen.
Scale	Integer greater than or equal to 0
Interpretation	A value of 0 signifies that the game has no menu screens, and players can exit directly from gameplay. Higher values indicate that the players have to navigate many screens before leaving the game. If the value is too high, players may get anxious before reaching the game exit as a result of the navigational complexity

1. The establishment of evaluation requirements. All the factors that will condition the evaluation of the game are determined in this phase. Evaluation profiles are chosen in order to specify which game development method is employed, which type of video game is developed, what the target technological platform is, and at which target players the game is aimed. Given a specific game development method, software artifacts (models) and attributes from the Video Game Usability Model are selected to perform early usability evaluations. The measures associated with the selected attributes are operationalized to provide both an instantiation of the generic formula for a specific software artifact (model) and thresholds for the measure values in accordance with the specific evaluation profile.

2. Early usability evaluation. In this phase, each selected video game software artifact (model) is evaluated with a set of measures. Each measure returns a numeric value within a specific threshold that indicates whether there is a usability issue in the video game. A usability report is consequently generated which details both the usability problem and suggestions to solve it.

3. Usability evaluation in-use. Even when early usability evaluation is performed on the video game software artifacts (models), the game may also need further usability in-use evaluation in a specific context with players. This usability-centered playtesting is well documented in the video game bibliography [8]. Since this paper focuses on early usability and model-driven development, usability in-use evaluation is not within the scope of this work.

After usability evaluations, game developers should perform refinements to solve the usability problems. Early usability issues detected in the game design phase can be directly refined in the game design stage. Usability in-use issues, however, may need refinements in all the phases of game development. In some cases, when the game meets all the evaluation requirements but the players are still not experiencing good usability, usability evaluation forces a re-check of the evaluation requirements, thus re-establishing the thresholds for the measures. In all cases, the game must be re-evaluated to verify whether the changes have solved the usability problems detected. This means that both game development and evaluation are iterative processes.

V. CASE STUDY

In order to show the feasibility of our approach, the Video Game Usability Model was applied to a specific example - a 2D fighting game for the XBOX 360, which is similar to the commercial Capcom's Street Fighter IV™ for the same platform. The example game was designed by following a specific model-driven video game development methodology. Section 5.A provides an overview of this specific video game development methodology. Section 5.B describes the activities concerned in the establishment of the usability evaluation requirements. Finally, Section 5.C shows how the operationalized measures were applied in order to perform an early evaluation of the selected artifacts.

A. Model-Driven Video Game Development

Model-driven video game development [14] is a game development methodology that focuses on defining platform-independent models which provide a precise high-level specification of the gameplay, control, and graphical user interface of the video game under development.

In this paper we focus only on the platform-independent models that offer the most suitable modeling primitives for usability evaluation. These platform-independent models are described as follows:

Screen Navigation Diagram. Video games display visual information on different game screens through which players can navigate. Fig. 1 shows the screen navigation metamodel. A screen navigation diagram can be specified by using *screen nodes* and *screen transitions*.

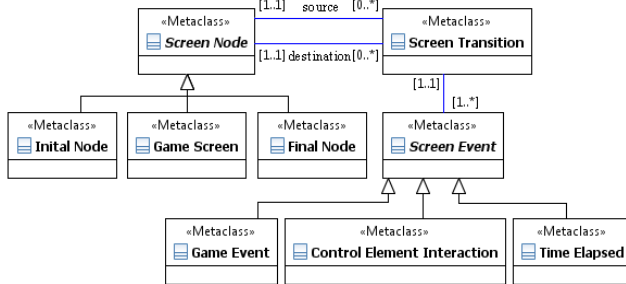


Fig. 1. Screen Navigation metamodel

A game screen represents a game state in the screen navigation. Two special screen nodes denote the *initial* and *final* states that define the screens on which a video game starts and ends. Screen transitions represent a change of state in the screen navigation, i.e., moving from one screen to another. Screen transitions are triggered by *screen events* such as control interactions, time, or rule executions.

Screen Layout Diagram. When the flow of screens is clearly defined in a screen navigation diagram, each game screen GUI should be further specified by using a screen layout diagram. Fig. 2 shows the screen layout metamodel.

A screen layout diagram can be specified by different GUI *display primitives* that can be positioned and sized on the screen. These primitives provide a visual representation of a game attribute which is previously defined in the gameplay perspective. There are four types of GUI display primitives: *numeric containers* and *textual containers* which represent information as plain numbers or text, *image containers* which represent information using 2D images or animations, and *progress containers* which represent the progress of information as a relative percentage of a colored bar or a succession of small icons.

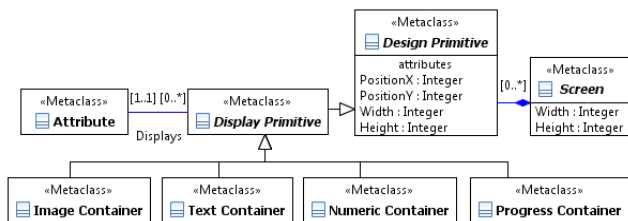


Fig. 2. Screen Layout metamodel

Control Mapping Diagram. A game control mapping defines how players interact with controller devices in order to communicate with the game. Fig. 3 shows the control mapping metamodel. A *controller* is a device that players use to communicate with the game. Controllers are made up of smaller *control elements* such as keys, buttons, joysticks and

triggers that players use to communicate atomic game interactions. *Control element interactions* such as pressing or releasing a button, moving a joystick, or pulling a trigger, activate the specific action rules of a player's character.

A control mapping diagram specifies which *control elements* and *interactions* are associated with gameplay *actions*.

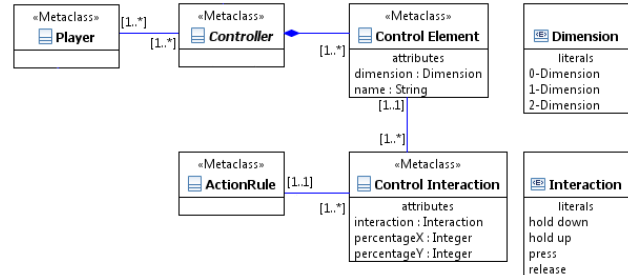


Fig. 3. Control Mapping metamodel

B. Establishment of the usability evaluation requirements

The evaluation profile of the example 2D fighting game used in the case study is as follows:

- Game development method: the application is designed by using the model-driven development method discussed in Section 5. The main software artifacts (models) involved in the early usability evaluation are the screen navigation diagram, the screen layout diagram and the control mapping diagram.
- Type of video game: the example game belongs to the 2D fighting genre.
- Target technological platform: the example game is developed for the XBOX 360 video game console.
- Target audience: the example game, like most 2D fighting games, is targeted at a hardcore audience of players who have a great deal of previous experience in games of the same genre, and who thus know and expect certain common genre conventions.

For the sake of simplicity, only two usability sub-characteristics of the Video Game Usability Model were evaluated in the selected models: *Appropriateness* and *Ease of Use*.

The selected attributes for the case study were *Visibility*, *Interface Simplicity*, *Control Simplicity* and *External Navigation Simplicity*, whose associated measures are shown in Table I.

The operationalizations of the aforementioned measures are presented in Table III. Note that all the measure thresholds defined in the operationalizations are defined in accordance with specific information from the evaluation profile of the example game used in the case study.

TABLE III. OPERATIONALIZED MEASURES FOR THE CASE STUDY

Measure	Percentage of Screen Usage (PSU)
Attribute	Appropriateness Recognisability / Visibility
Artifact	Screen Layout Diagram (PIM)
Operationalization	Each display primitive of the Screen Layout Diagram has attributes for its width and height. The screen metaclass also has attributes for its width and height. Both the primitive display size and the screen size can be defined as the product of their width and height
Formula	$PSU = (\sum \text{display primitives width} \times \text{height}) / (\text{screen width} \times \text{height})$
Thresholds	The XBOX 360 is typically played on a high-resolution TV, which benefits visibility. Hardcore players are also well trained in the specific genre conventions of 2D fighting games, thus minimizing the space needed to convey the game's visual information. Critical Usability Problem: [PSU > 0.5] Low Usability Problem: [0.1 < PSU ≤ 0.2] Medium Usability Problem: [0.2 < PSU ≤ 0.5] No Usability Problem: [PSU ≤ 0.1]
Measure	Total Number of GUI Elements (TNGUIE)
Attribute	Appropriateness Recognisability / Interface Simplicity
Artifact	Screen Layout Diagram (PIM)
Operationalization	Each display primitive of the Screen Layout Diagram is a GUI of the screen
Formula	TNGUIE = Sum of all display primitives of the Screen Layout Diagram
Thresholds	Hardcore players are experienced in the genre conventions of 2D fighting games and therefore expect their typical interface layout, with a number of GUI elements between 3 and 10. Critical Usability Problem: [TNGUIE > 10] Low Usability Problem: [3 < TNGUIE ≤ 5] Medium Usability Problem: [5 < TNGUIE ≤ 10] No Usability Problem: [0 ≤ TNGUIE ≤ 3]
Measure	Total Number of Control Mappings (TNCM)
Attribute	Appropriateness Recognisability / Control Simplicity
Artifact	Control Mapping Diagram (PIM)
Operationalization	Each control element of the Control Mapping Diagram is associated with a control interaction primitive and, with an action rule primitive
Formula	TNCM = Sum of all control elements associated with an action rule.
Thresholds	The XBOX 360 offers many buttons, triggers and joysticks. Hardcore players are experienced in the complex control interactions required by the 2D fighting genre. Critical Usability Problem: [TNCM > 12] Low Usability Problem: [8 < TNCM ≤ 10] Medium Usability Problem: [10 < TNCM ≤ 12] No Usability Problem: [0 ≤ TNCM ≤ 8]
Measure	Shortest Path To Gameplay (SPTG)
Attribute	Ease of Use / External Navigational Simplicity
Artifact	Screen Navigation Diagram (PIM)
Operationalization	Each screen primitive of the Screen Navigation Diagram can be associated with a game screen
Formula	SPTG = Minimum number of transitions between the initial screen node and the gameplay screen node
Thresholds	Hardcore players typically value direct gameplay over cumbersome menu interfaces. Medium Usability Problem: [SPTG > 3] No Usability Problem: [0 ≤ SPTG ≤ 3]
Measure	Shortest Path To Exit (SPTe)
Attribute	Ease of Use / External Navigational Simplicity
Artifact	Screen Navigation Diagram (PIM)
Operationalization	Each screen primitive of the Screen Navigation Diagram can be associated with a game screen.
Formula	SPTe = minimum number of transitions between the gameplay screen node and the final screen node.
Thresholds	The XBOX 360 offers a built-in interface to exit from the game at any moment. Hardcore players value immediateness of menu interfaces. Medium Usability Problem: [SPTe > 2] No Usability Problem: [0 ≤ SPTe ≤ 2]
Measure	Shortest Return Path To Gameplay (SRPTG)
Attribute	Ease of Use / External Navigational Simplicity
Artifact	Screen Navigation Diagram (PIM)
Operationalization	Each screen primitive of the Screen Navigation Diagram can be associated with a game screen.
Formula	SRPTG = minimum number of transitions between the game over screen node and the gameplay screen node.
Thresholds	Hardcore players value immediateness of menu interfaces. Medium Usability Problem: [SRPTG > 2] No Usability Problem: [0 ≤ SRPTG ≤ 2]

C. Early usability evaluation of software artifacts

With regard to the Screen Layout Diagram (see Fig. 4), we apply the two specific measures shown for this artifact in Table III in order to evaluate the *Visibility* and *Interface Simplicity* attributes of the video game.

- By applying formula of the **Percentage of Screen Usage** we obtain PSU = 0.09 (by dividing the sum of the size of all the display primitives by the screen size). This indicates that there is no usability problem related to the *Visibility* attribute since PSU is in the threshold [PSU ≤ 0.1]. By applying the formula of **Total Number of GUI Elements** we obtain TNGUIE = 13 (by counting all the display primitives in the diagram), which leads to a critical usability problem related to the *Interface Simplicity* attribute since the value obtained is [TNGUIE > 10]. Table IV shows the usability report associated to this usability problem (UP001).



Fig. 4. Street Fighter IV screenshot and its Screen Layout Diagram

TABLE IV. USABILITY REPORT FOR USABILITY PROBLEM UP001

ID	UP001
Description	There are too many GUI Elements on the same game screen.
Affected attribute	Appropriateness Recognisability / Interface Simplicity
Severity level	Critical [TNGUIE=13 > 10]
Artifact evaluated	Screen Layout Diagram
Problem source	Screen Layout Diagram
Recommendations	Collapse GUI elements that render the same information, such as the image and the text container that portray the fighter portrait and name

With regard to the Control Mapping Diagram, we can apply a measure from Table III in order to evaluate the *Control Simplicity* attribute of the video game:

- By applying the formula of **Total Number of Control Mappings** we obtain TNCM = 7 (by counting all the control elements: 1 x 2-Dimensional control element +

6 x 1-Dimensional control element). This signifies that there is no usability problem related to the *Control Simplicity* attribute since the value obtained is in the threshold [$0 \leq \text{TNCM} \leq 8$]. The game uses a small set of controls for the basic game actions.

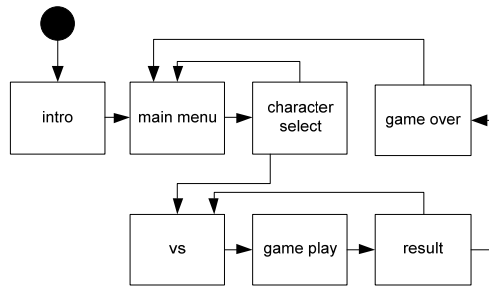


Fig. 5 Street Fighter IV Screen Flow Diagram

With regard to the Screen Flow Diagram (see Fig. 5), we can apply three specific measures from Table III in order to evaluate the *External Navigation Simplicity* attribute belonging to the video game's *Ease of Use* sub-characteristic:

- By applying the formula of **Shortest Path To Gameplay** we obtain $\text{SPTG} = 4$ (by counting the minimum number of screen transitions between the *intro* and *gameplay* screens), which leads to a medium usability problem related to the *External Navigation Simplicity* attribute, since the value obtained is in the threshold [$\text{SPTG} > 3$]. Table V presents the usability report associated with this usability problem (UP002).
- By applying the formula of **Shortest Path To Exit** we obtain $\text{SPTE} = 0$, since there is no final screen node. This value shows that there is no usability problem. Most XBOX 360 games use the built-in console interface rather than an in-game option to exit from the game. This measure ensures that the XBOX 360 interface shortcut to exit from the game enhances videogame usability.
- By applying the formula of **Shortest Return Path To Gameplay** we obtain $\text{SRPTG} = 4$ (by counting the screen transitions from *gameplay*, *result*, and *vs* screens), which leads to a medium usability problem related to the *External Navigation Simplicity*, since the value obtained is in the threshold [$\text{SRPTG} > 2$]. Table VI presents the usability report associated with this usability problem (UP003).

TABLE V. USABILITY REPORT FOR USABILITY PROBLEM UP002

ID	UP002
Description	There are too many screens that render redundant or non-interactive information
Affected attribute	Ease of Use / External Navigational Simplicity
Severity level	Medium [$\text{SPTG} = 4 > 3$]
Artifact evaluated	Screen Flow Diagram
Problem source	Screen Flow Diagram
Recommendations	Allow players to skip the introduction cut-scene and the fighters versus screen.(or collapse non-interactive information screens)

After applying the measures, we can conclude with regard to the *Appropriateness Recognisability* sub-characteristic that the video game has poor *Interface Simplicity* but very good *Visibility* and *Control Simplicity*, i.e., the game has a complex interface but effectively manages to keep gameplay visible and the control schema simple. With regard to the *Ease of Use* sub-characteristic, we can conclude that the video game has poor *External Navigational Simplicity*, i.e., the game has a complex flow of screens which makes it difficult to start and restart the game.

TABLE VI. USABILITY REPORT FOR USABILITY PROBLEM UP003

ID	UP003
Description	Need to navigate through several screens to restart the game
Affected attribute	Ease of Use / External Navigational Simplicity
Severity level	Medium [$\text{SRPTG} = 4 > 2$]
Artifact evaluated	Screen Flow Diagram
Problem source	Screen Flow Diagram
Recommendations	Add a shortcut (e.g., retry) from the game-over screen to the gameplay screen

VI. CONCLUSIONS AND FURTHER WORK

This paper presented a usability evaluation strategy that can be used in early stages of model-driven video game development. The strategy relies on a Video Game Usability Model that has been developed specifically for the video game domain. This model is aligned with the SQuaRE standard and allows the evaluation and improvement of the usability of video games developed according to a model-driven development process. Thus, our strategy does not only allow to perform usability evaluations when the video game is completed, but also in early stages of its development. Usability is therefore considered throughout the entire game development, thus enabling a more usable video game to be developed and thereby reducing the maintenance effort.

The inherent features of model-driven development provide a suitable context in which to perform usability evaluations since usability problems that may appear in the final application can be detected and corrected at the model level. Model-driven development also allows automating common usability evaluation tasks that have been traditionally performed by hand (e.g., generating usability reports). Although the proposed usability model has been operationalized to a specific video game development method, it can also be applied to other methods by specifying the relationships between the generic measures from the usability model and the modeling primitives of the different software artifacts of the selected game development method. Finally, it is worth mentioning that the proposed usability model can be used to discover deficiencies and/or limitations in the expressiveness of the model primitives to support certain usability attributes.

Future work include the application of the strategy to industrial case studies and the definition of aggregation mechanisms for combining the values obtained from individual measures into usability indicators. We also plan to

empirically validate the completeness and effectiveness of the proposed usability evaluation strategy (and the video game usability model) by means of controlled experiments in which the results of the evaluations obtained at the model level will be compared to the ones obtained when players interact with the generated video game application.

ACKNOWLEDGMENTS

This research work is funded by the MULTIPLE project (MICINN TIN2009-13838), the FPU program (AP2007-03731) from the Spanish Ministry of Science and Innovation, and the FPI program (199880998) from the UPV.

REFERENCES

- [1] Abrahão S., Iborra E., Vanderdonck J.: Usability Evaluation of User Interfaces Generated with a Model- Driven Architecture Tool. *Maturing Usability: Quality in Software, Interaction and Value*, Springer, pp. 3-32 (2007)
- [2] Abrahão S., Insfran E.: Early Usability Evaluation in Model-Driven Architecture Environments. In: 6th IEEE International Conference on Quality Software (QSIC'06), Beijing, China. IEEE Computer Society, pp. 287-294 (2006)
- [3] Ampatzoglou A., Stamelos I.: Software engineering research for computer games: A systematic review. In: *Information and Software Technology*, Volume 52, Issue 9, pp. 888-901 (2010)
- [4] Andre T.S, Hartson H.R, Williges R.C.: Determining the effectiveness of the usability problem inspector: a theory-based model and tool for finding usability problems. *Human Factors* 45(3): 455-82 (2003)
- [5] Calero C., Ruiz J., Piattini M.: *Classifying Web Metrics Using the Web Quality Model*. Emerald Group Publishing Limited. Vol. 29, Issue 3, pp. 227-248 (2005)
- [6] Federoff M.: *Heuristics and Usability Guidelines for the Creation and Evaluation of Fun in Video Games*. Indiana University Master of Science Thesis (2002)
- [7] Fernandez A., Insfran E., Abrahão S.: Integrating a Usability Model into a Model-Driven Web Development Process. 10th International Conference on Web Information Systems Engineering (WISE 2009), pp. 497-510, Springer-Verlag (2009)
- [8] Greenwood-Ericksen A., Preisz E., Stafford S.: Usability Breakthroughs: Four Techniques To Improve Your Game. In: *Gamasutra* (2010), http://www.gamasutra.com/view/feature/6130/usability_breakthroughs_four_.php.
- [9] Hall A., Chapman R.: Correctness by construction: Developing a commercial secure system. *IEEE Software*, 19(1), 18–25 (2002)
- [10] Hwang W., Salvendy G.: Number of people required for usability evaluation: the 10±2 rule. In *Communications of the ACM* 53(5), 130-133 (2010)
- [11] ISO/IEC: ISO/IEC 25010 Systems and software engineering -- Systems and software Quality Requirements and Evaluation (SQuaRE) -- System and software quality models (2011)
- [12] Järvinen A., Heliö S. Mäyrä F.: *Communication and Community in Digital Entertainment Services*. Prestudy Research Report, Hypermedia Laboratory, University of Tampere, Tampere (2002), <http://tampub.uta.fi/tup/951-44-5432-4.pdf>.
- [13] Microsoft: Best Practices for Indie Games 3.1, http://create.msdn.com/en-US/education/catalog/article/bestpractices_31.
- [14] Montero E., Carsi J.A.: A Platform-Independent Model for Videogame Gameplay Specification. In: *Digital Games Research Association Conference (DiGRA'09)*, London, UK (2009), <http://www.digra.org/dl/db/09287.28003.pdf>.
- [15] Nacke L.: From Playability to a Hierarchical Game Usability Model. In: *FuturePlay at Game Developers Conference Canada*, Vancouver, Canada (2009)
- [16] Nokia: Top Ten Usability Guidelines for Mobile Games. In: *Design and User Experience Library v2.0*, http://library.forum.nokia.com/topic/Design_and_User_Experience_Library/top10_usability.pdf
- [17] Pinelle D., Wong N., Stach T.: Heuristic Evaluation for Games: Usability Principles for Video Game Design. In: *Proceedings of the Special Interest Group in Computer Human Interaction (SIGCHI'08)*, ACM, pp. 1453–1462 (2008)

Tool Support for Quality Aware Product Configuration in Software Product Lines

Guoheng Zhang, Huilin Ye, and Yuqing Lin

School of Electrical Engineering and Computer Science
University of Newcastle
Callaghan 2308, NSW, Australia

Abstract - *Quality aware product configuration (QAPC) is the process of configuring a product from a feature model based on the customers' functional requirements as well as quality requirements. The key issue of achieving QAPC is to measure the interdependencies between features and quality attributes. Existing QAPC approaches have several limitations on this issue, such as lacking of an appropriate method for measuring the interdependencies and lacking of a complete tool support for product configuration. To overcome these limitations, a tool for QAPC is developed based on a systematic approach for interdependency measurement. The measured results will be represented as knowledge for product configuration in a quality attribute knowledge base (QAKB) that is incorporated into the tool to guide the QAPC process. A case study based on a tourist guide software product line is presented to demonstrate how the tool works.*

Keywords: product configuration; quality attributes; software product line; feature models; tool.

1 Introduction

Cost, product, and time-to market have been the main concerns in software engineering since 1960s [1]. Software product line approach emerges as an attractive software reuse approach to address these concerns. A software product line is defined as "a set of software-intensive systems sharing a common, managed set of features that satisfy specific needs of a particular market or mission, and that are developed from a common set of core assets in a prescribed way" [2]. In a software product line, software products are developed in a two-stage process: domain engineering and application engineering. In domain engineering, the commonalities and variabilities of software product line members are identified and implemented into a set of reusable software artifacts. In application engineering, the new software applications are derived from the software product line by composing a subset of the reusable artifacts.

As an important artifact developed in domain engineering, a feature model is used to capture and represent the common and variable characteristics of software product line members in terms of features and to specify the

constraints among features. In application engineering, a feature model is used to capture the configuration rules for a software product line and it serves as configuration tools to configure specific software applications from a software product line. The process of deriving a member product from a feature model by selecting the desired features based on the customers' requirements is named as feature based product configuration (FBPC). In most cases, application engineers select functional features based on the customers' functional needs in FBPC. However, only functional requirements are not sufficient for the satisfaction of the final products. Non-functional requirements, also called system quality attributes, are also major concerns of different stakeholders for the target product. The need of deriving a product that satisfies the quality requirements from a software product line motivates the quality aware product configuration, the process of configuring a product from feature models with a full consideration of the customers' functional requirements as well as the customers' quality requirements.

To achieve quality aware product configuration, we must understand the interdependencies between functional features and quality attributes in a feature model. The contributions to a quality attribute made by individual functional features must be estimated. Then these individual contributions need to be analyzed to quantify the aggregated impact on the quality attribute of a configured product. Current existing approaches either require real final products or involve heavy domain experts' judgments for measuring interdependencies. However, the real products are usually not available during the product configuration stage and the manual one by one judgment on quality levels of configured products is time-consuming. To overcome these limitations, we have proposed an approach to modelling quality attributes in feature models in our early works [3]. A pair-wise comparison method called analytic hierarchical process [4] is adapted to measure individual functional features' contribution to a quality attribute. The collective impact of these contributions on a quality attribute can be estimated based on the defined quantitative assessment. Once the interdependencies between functional features and quality attributes have been recognized, this knowledge can be used for assessing quality attribute levels for any configured product, i.e., no need to assess the products one by one. As a result, the domain

experts' efforts involved in the assessment have been significantly reduced. The quality knowledge of a software product line can be stored in a knowledge database named as Quality Attribute Knowledge Base (*QAKB*) to facilitate quality aware product configurations in the software product line.

Most product configuration approaches need tool support, as it is time-consuming for application engineers to manually configure products from feature models [5]. In the context of quality aware product configuration, the tools are even more important, as the product configuration process becomes more complex when involving the customers' quality requirements. Most existing *QAKB* approaches neglect tools [6-10] or only provide incomplete tool support [11-15]. To fill this gap, two tools have been developed; one tool named as *QAMTool* aims to help domain engineers to model quality attributes in a feature model and the other named as *QAPCTool* aims to assist application engineers in the process of quality aware product configuration.

The reminder of this paper is organized as follows: Section 2 will introduce the related works on quality aware product configuration and their supporting tools. Section 3 will introduce the *QAMTool* which is used for quality attributes modelling in feature models. Section 4 will introduce the *QAPCTool* which is used to conduct quality aware product configuration process. Finally, we conclude this paper and identify the future work in section 5.

2 Related works

In literature, several approaches on quality attributes modelling in feature models and quality aware product configuration have been proposed and some of them have provided tool support. For example, Lee et al. develops a quality attribute feature diagram and relates the *QA* features with functional features using some qualitative labels [6]. Yu et al. uses a goal model to represent stakeholders' goals and trace goals to features [7]. Sinnema et al. uses a dependency to represent a system property and specifies how the selection of variants at different variation points influence the value of the system property [12]. The limitation of these approaches is that they only support rough quality attributes assessments as they use qualitative values to represent the impact of features on quality attributes. To support more precise assessments, several approaches propose quantitative analysis methods by testing the generated products [8, 11, 13]. Although these approaches can support more precise assessments, they are inefficient as it is a costly and time-consuming task to generate real products in practice. To avoid generating products for interdependency measurement, Zhang et al. proposed an approach using *Bayesians Belief Network* to represent the quantitative impact of a feature configuration on a quality attribute [10]. Bagheri et al. proposes an approach for prioritizing features based on system concerns [15]. Although real products are not required in these

approaches, they need heavy domain experts' efforts involved in judgments.

Some of the above approaches have provided tool support. For example, Siegmund et al. develops *SPL Conquerer* to automate interdependency measurement, computation of the product sets, and approximation of a feature's non-functional property [11]. Sinnema et al. develops *Mocca* for managing variability and dependency which represents non-functional properties [12]. Sincero et al. extends *Linux Kernel Configuration Tool* to display the non-functional property of a feature selection [13]. White et al. develops a tool named as *Scatter* to output a specific product based on the resource constraints [14]. Bagheri et al. extends *Feature Model Plug-in (fmp)* to provide application engineers with the rankings of variant features under a variation point [15]. However, none of these tools provide a complete support for activities from domain engineering to application engineering.

Our early proposed approach [3] can overcome the limitations of the existing approaches. In the following sections, we aim to introduce our developed tools that implement the concepts of the early approach.

3 Tool support for modelling quality attributes in feature models

In this section, we will introduce the Quality Attributes Modelling Tool (*QAMTool*) that supports our early approach of modelling quality attributes in feature models. In our early approach [3], we first identify the quality attributes that are critical for a software product line by adapting non-functional requirements framework and extend feature models with a sub-feature tree called quality attribute (*QA*) feature diagram to organize the identified quality attributes. Second, we measure the interdependencies between functional features and the identified quality attributes based on a pair-wise comparison method called analytic hierarchical process (*AHP*). The relative impacts of individual features on a quality attribute and the inter-relationships among features with respect to affecting a quality attribute are estimated based on the domain experts' judgments using *AHP*. Then the overall quality of a configured product can be assessed by aggregating the relative impact of features included in the configured product. Third, we store the quality knowledge about the measured interdependencies and the inter-relationships among different quality attributes into a knowledge database called quality attribute knowledge base (*QAKB*).

A tourist guide software product line is used to illustrate how *QAMTool* supports our approach. As shown in Figure 1, the feature diagram shows the feature model of the tourist guide software product line [3] and the *QA* feature diagram shows the quality attributes of this software product line. The

process of identifying these quality attributes for the tourist guide software product line can be found in [3].

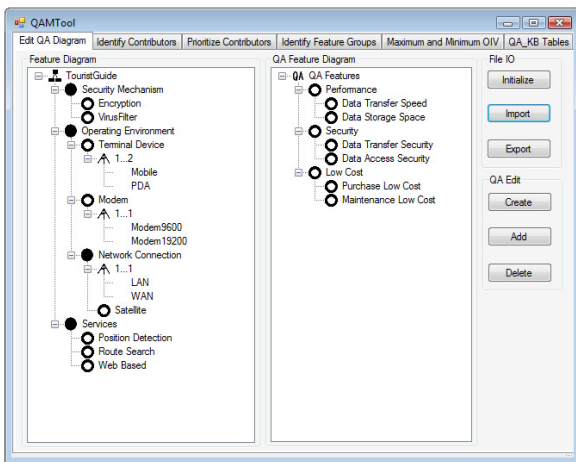


Figure 1 Feature Diagrams of Tourist Guide SPL in QAMTool

Once the quality attributes of a software product line are identified and represented as quality attribute features in a *QA* feature diagram, we need to measure the interdependencies between features and quality attributes. A set of sub-steps need to be followed to measure the interdependencies.

Step 1: Identify the contributors of a quality attribute (QA): A quality attribute (*QA*) is related to a set of features whose inclusions or exclusions will have either positive or negative impact on *QA*. The set of features affecting a quality attribute are named as *contributors* of the quality attribute. The first step of measuring interdependencies is to identify the contributors of a quality attribute *QA*. The identification process can be achieved based on *NFR framework* [16] and domain experts' knowledge and experience. As shown in Figure 2, *QAMTool* can help to model the positive contributors and negative contributors for a quality attribute. Figure 2 shows that we have identified the contributors of data transfer speed (*DTS*) as "Encryption", "Mobile", "Modem19200", "Modem9600", "LAN", "PDA" and "WAN". Among these contributors, "Encryption" has negative impact on *DTS* and others have positive impact on *DTS* when they are selected into product configuration.

Step 2: Prioritize the contributors of QA: The identified contributors have different impacts on *QA*. We adapt a popular pair-wise comparison method, *analytic hierarchical process (AHP)*, to prioritize the identified contributors based on their relative importance for satisfying the quality attribute. Domain experts compare each pair of features among the contributors of *QA* and assign each comparison with a value (-9.0 ~ +9.0) which represents domain experts' belief about how much a feature is more important or less important than another feature in terms of affecting *QA*. Then a comparison matrix which uses the contributors of *QA* as both the column members and the row members is made based on the pair-wise

comparisons. From the comparison matrix, we can calculate a priority vector which consists of the relative impact of each contributor of *QA*. We define *Relative Importance Value (RIV)* to represent the calculated relative impact of individual contributors and use $RIV(QA, F)$ to represent the *RIV* of feature *F* on quality attribute *QA*. As shown in Figure 3, *QAMTool* supports to prioritize the contributors of a quality attribute based on *AHP* method. In Figure 3, domain experts use *QAMTool* to generate a comparison matrix for the contributors of data transfer speed (*DTS*). *Modem 19200* in the row is moderately more important than *Mobile* in the column, so a value "+5" is assigned to the corresponding cell of the matrix. *Mobile* in the row is absolutely less important than *LAN* in the column, so a value of "-9" is assigned. The details of importance intensity can be found in [3]. Based on the priority vector calculated from the comparison matrix in Figure 3, we can know that $RIV(DTS, WAN) = 28.74$ and $RIV(DTS, LAN) = 33.91$, which means *LAN* is more important than *WAN* for satisfying data transfer speed.

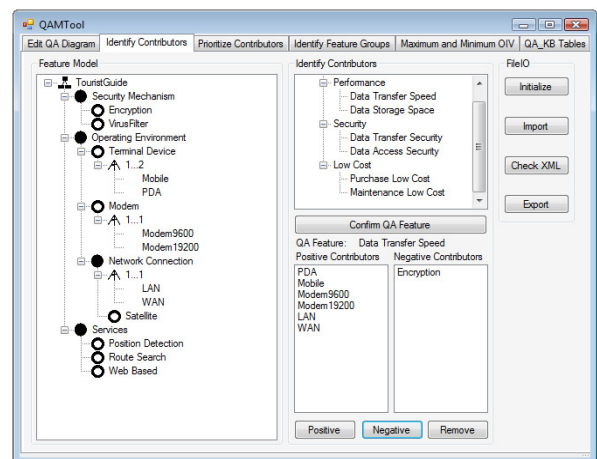


Figure 2 Function of Identifying Contributors for a Quality Attribute

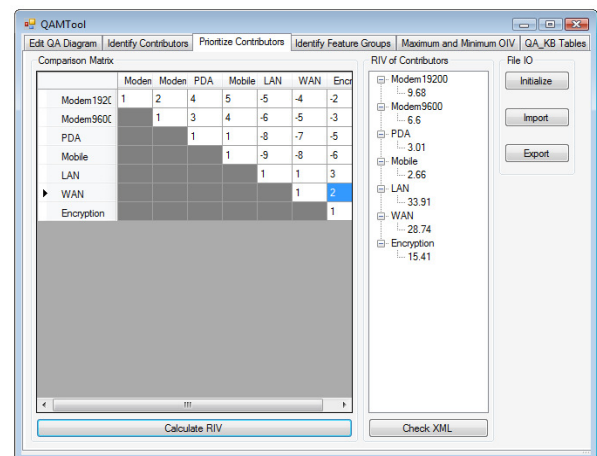


Figure 3 Function of Prioritizing Contributors of a Quality Attribute

Step 3: Identify the relationships among contributors of QA: Once we have calculated the impact of individual contributors of QA , we can calculate the overall impact on QA made by a set of contributors. We define *Overall Importance Value (OIV)* to be the overall impact of a combination of contributors and use $OIV(QA, fg)$ to represent the OIV of a set of contributors fg from the contributors of QA . Intuitively, the simplest way to calculate $OIV(QA, fg)$ is to add the RIV of all the contributors in fg . However, in many cases, some contributors of QA will affect QA interdependently, i.e. they are related with each other in terms of affecting QA . The overall impact of two related contributors may not be the sum of their relative impact. To recognize the relationships among some contributors in terms of affecting QA , we define four types of feature groups: $SumGp$, $AvgGp$, $MinGp$ and $MaxGp$. If the OIV on QA made by the selected contributors from a feature group can be considered as an average of the RIV of individual selected contributors, the feature group is called $AvgGp$. The definitions of other feature groups can be found in [3]. The current version of $QAMTool$ supports the above four types of feature groups. With the $QAMTool$, domain experts can model four feature groups among the contributors of data transfer speed: $\{Modem19200, Modem9600\}$ and $\{PDA, Mobile\}$ are two $AvgGp$, $\{Encryption\}$ is a $SumGp$; and $\{LAN, WAN\}$ is a $MinGp$.

Step 4: Calculate and normalize the overall impact of a configured product on QA: The QA level of a configured product is determined by the overall impact of the set of contributors included in the configured product cp , which can be represented as $OIV(QA, cp)$. The calculation of $OIV(QA, cp)$ is based on the RIV of contributors included in the configured product and feature groups these contributors belong to. As the calculated OIV of a configured product cannot represent its relative QA level in the application domain comparing with other SPL members, we use formula (1) to normalize OIV into normalized overall importance value ($NOIV$) in $[0...1]$ where "1" represents the highest QA level, "0" represents the lowest QA level and a number between "0" and "1" represents the relative QA level comparing with the highest one and the lowest one. To use formula (1), we must obtain the maximum OIV and the minimum OIV among all software product line members. Figure 4 shows the function of calculating the maximum OIV and minimum OIV in $QAMTool$. We adapt $FAMA$ [17] to find all valid products of a software product line and use our tool to calculate the OIV for each valid product. Then we can find the maximum OIV and the minimum OIV among all valid products. As shown in Figure 4, for data transfer speed (DTS) in the tourist guide SPL , the maximum OIV is 59.00 and the minimum OIV is 31.75.

$$NOIV(QA, VS) = \frac{OIV(VS) - MIN(OIV(VS_i))}{MAX(OIV(VS_i)) - MIN(OIV(VS_i))} \quad (1)$$

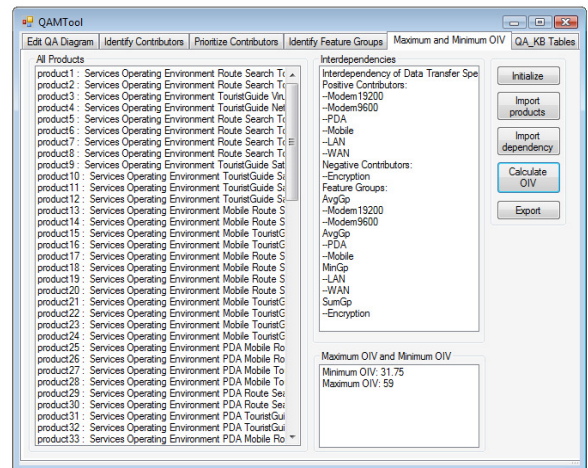


Figure 4 Function of Calculating Overall Importance Value

By the above steps, we can establish the interdependency between a quality attribute and its contributors. For a specific quality attribute QA , the interdependency should include all the contributors of QA and their RIV , the feature groups that the contributors belong to, and the maximum and minimum OIV . This interdependency can be used as a reusable artifact of a software product line to predict the quality attributes of a configured product. However, if the predicted quality attributes of a configured product cannot satisfy the customers' quality requirements, we cannot use this interdependency to find solutions of modifying the existing product to satisfy the customers' quality requirements. For example, we can include some features or exclude some features to achieve the desired QA level based on the interdependency between QA and its contributors. However, it is not clear that what impact this modification will have on other quality attributes. In this case, we find that the information missing in the interdependency are the relationships among related quality attributes. Without the explicit knowledge of the related quality attributes, it is difficult for application engineers to configure a product that satisfies the customers' quality requirements in an informed and rational way. Therefore, we develop another reusable artifact, a *quality attribute knowledge base (QAKB)*, to manage the relationships among related quality attributes.

A $QAKB$ is a kind of knowledge database which stores the information about the relationships among related quality attributes in a software product line. We design a $QAKB$ as multiple tables, each of which representing the relationships among a set of related quality attributes. An example of $QAKB$ table is shown in Table 1. The columns of the table represent a set of related quality attributes in a software product line while the rows of the table represent the valid selections with respect to this set of related quality attributes. Herein, a valid selection in a $QAKB$ table is a valid combination of features among the contributors of the set of related quality attributes in the $QAKB$ table. The

corresponding cell $NOIV(QA_i, VS_j)$ in the table represents the QA_i level of valid selection VS_j .

Table 1 QAKB Table

	QA_1	...	QA_i	...	QA_m
VS_1	$NOIV(QA_1, VS_1)$...	$NOIV(QA_i, VS_1)$...	$NOIV(QA_m, VS_1)$
...
VS_j	$NOIV(QA_1, VS_j)$...	$NOIV(QA_i, VS_j)$...	$NOIV(QA_m, VS_j)$
...
VS_n	$NOIV(QA_1, VS_n)$...	$NOIV(QA_i, VS_n)$...	$NOIV(QA_m, VS_n)$

To develop a $QAKB$ table as Table 1, we need to achieve two tasks: first, we need to derive all the valid selections with respect to the related quality attributes $\{QA_1, QA_2...QA_m\}$. We draw a new feature diagram which includes the contributors of $\{QA_1, QA_2...QA_m\}$. The selection constraints among the included features can be derived from the original feature diagram. From such a feature diagram, we can derive a set of valid selections with respect to $\{QA_1, QA_2...QA_m\}$. The second task is to calculate $NOIV(QA_i, VS_j)$ for each cell in the table. The calculation of $NOIV(QA_i, VS_j)$ can be achieved based on the interdependency between QA_i and its contributors. In the example of tourist guide software product line, we can generate a $QAKB$ table for two related quality attributes data transfer speed (DTS) and data transfer security ($DTSS$) as shown in Table 2.

Table 2 A QAKB Table of DTS and $DTSS$

	Valid Selections for DTS and $DTSS$	$NOIV$ for $DTSS$	$NOIV$ for DTS
VS_1	WAN, Web-Based, PDA	0.0	0.57
VS_2	WAN, Web-Based, PDA, Mobile	0.0	0.57
VS_3	WAN, Web-Based, PDA, VirusFilter	0.05	0.57
VS_4	WAN, Web-Based, PDA, Mobile, VirusFilter	0.05	0.57
VS_5	WAN, Web-Based, PDA, VirusFilter, Encryption	0.74	0.0
VS_6	WAN, Web-Based, PDA, Encryption	0.68	0.0
VS_7	WAN, Web-Based, Mobile	0.0	0.56
VS_8	WAN, Web-Based, Mobile, VirusFilter	0.05	0.56
VS_9	LAN, Modem 9600	0.27	0.89
VS_{10}	LAN, Modem 19200	0.27	1.0
VS_{11}	LAN, Modem 9600, Encryption	0.95	0.32
VS_{12}	LAN, Modem 19200, Encryption	0.95	0.43
VS_{13}	LAN, Modem 9600, Encryption, VirusFilter	1.0	0.32
VS_{14}	LAN, Modem 19200, Encryption, VirusFilter	1.0	0.43
VS_{15}	LAN, Modem 9600, VirusFilter	0.32	0.89
VS_{16}	LAN, Modem 19200, VirusFilter	0.32	1.0

Once the $QAKB$ of a software product line is generated, we can use this knowledge database to assist application

engineers to derive a product with desired software qualities in quality aware product configuration process.

4 Tool support for quality aware product configuration

In this section, we will introduce the Quality Aware Product Configuration Tool ($QAPCTool$) we developed to assist application engineers in the quality aware product configuration ($QAPC$) process. In $QAPC$, we first validate the customers' quality requirements to check whether these exists any conflicts in the requirements. Then we select or remove features based on the customers' functional requirements to generate a configured product. Finally, if the configured product fails to satisfy the customers' quality requirements, we provide solutions of modifying the existing configured product to achieve the desired quality attributes.

4.1 Validating quality requirements

In some cases the customers' quality requirements cannot be achieved by a software product line, which means none of the software product line members can satisfy the customers' quality requirements. For example, a customer may expect to configure a product with high security, high performance and low purchase cost. Obviously, it is impossible to configure such a product from a software product line as these three quality attributes conflict with each other. Therefore, the first step in quality aware product configuration is to validate the customers' quality requirements. The method of validating the customers' quality requirements is straightforward. If none of the valid selections in $QAKB$ tables can satisfy the customers' quality requirements, we can say that the customers' quality requirements are invalid.

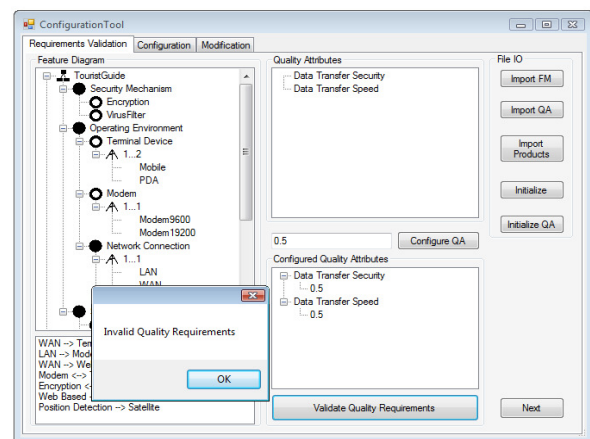


Figure 5 Function of Validating Customers' Quality Requirements

Figure 5 shows the function of validating the customers' quality requirements in $QAPCTool$. First, application engineers interpret the customers' quality requirements into a number in $[0...1]$ scale. In the example of Figure 5, the

customers' quality requirements are interpreted as 0.5 for data transfer speed (DTS) and 0.5 for data transfer security (DTSS). Then QAPC can check whether the customers' requirements on DTS and DTSS conflict with each other. As none of the valid selections in QAKB of Table 2 can satisfy the above quality requirements, the QAPCTool returns "Invalid Quality Requirements". The customers need to modify their desired levels on these two quality attributes. In this example, the customers modify their quality requirements as 0.4 for DTS and 0.4 for DTSS, which are valid by QAPCTool checking.

4.2 Making Decisions on Variation Points

After validating the customers' quality requirements, application engineers will make decisions on variation points in product configuration process. The product configuration is an iterative decision making process. At each step, application engineers select a variation point) and make decisions based on the customers' functional requirements. A configured product is derived from the feature model if decisions on all variation points have been made.

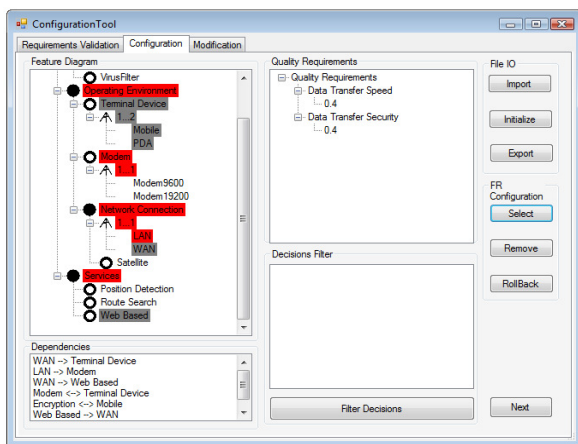


Figure 6 Function of Making Decisions on Variation Points

Figure 6 shows the function of making decisions on variation points in QAPCTool. The inclusion or removal of a feature is achieved by selecting the feature in the feature diagram and pressing the button "Select" or "Remove" respectively. The selection or removal of a feature will be propagated to other features automatically based on the constraint propagation algorithm we developed in [18]. Therefore, inconsistent decisions can be avoided in QAPCTool. If application engineers made a wrong decision, the QAPCTool has the "Rollback" function which can recover all the features affected by the wrong decision. In the example of Figure 6, application engineers select LAN based on the customers' functional requirements and the inclusion of LAN leads to the inclusion of Modem (colored in black) and the removal of WAN, Web-Based, Terminal Device, Mobile and PDA (colored in grey). When all features are selected or removed in the feature diagram, a configured product is derived.

4.3 Modifying Product Configuration

After making decisions on all variation points, a configured product can be obtained. If the configured product can satisfy the customers' quality requirements, it can be used as the final product for further software development; otherwise it must be modified to achieve the desired software qualities. In the latter case, the customers have no knowledge about how to modify the existing configured product to achieve their quality requirements. In this step, we provide application engineers with a set of solutions of modifying the configured product to achieve the desired quality requirements. Then application engineers can choose their desired solution with a full consideration of the changed functionalities and the changed quality attributes. The idea of finding the modification solutions for satisfying the quality requirements is to find all valid selections that satisfy the customers' quality requirements in QAKB tables and compare these valid selections with the existing configured product.

Figure 7 shows the function of modifying the existing configured product to achieve the desired software qualities in QAPCTool. If the predicted quality attributes of the configured product cannot achieve the customers' quality requirements, all the modification solutions will be returned in the list box "Modification Solutions" in QAPCTool. Each solution will provide the set of features that need to be included, the set of features that need to be removed, and quality attribute levels of the modified product. In the example of Figure 7, two modification solutions are provided, as the predicted quality attributes of the configured product cannot satisfy the quality requirements. A modification solution shown in Figure 7 is removing Encryption and VirusFilter from the current product and the quality attribute levels of the modified product is 0.43 for DTS and 1.0 for DTSS. Finally, application engineers select a desired solution and exports the final product into an xml file.

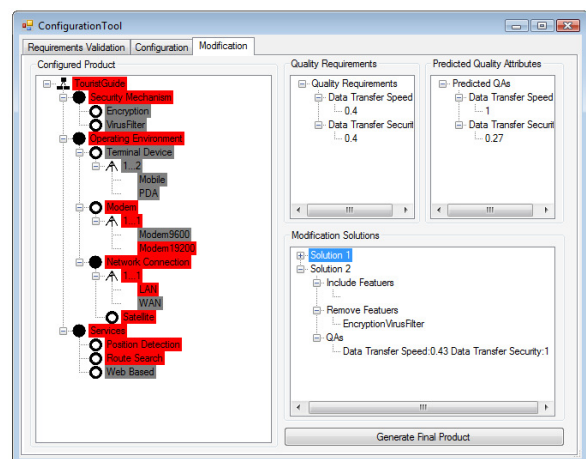


Figure 7 Function of Modifying Product Configuration

5 Conclusion and future work

In this paper, we develop tools that support the activities of our approach on quality attributes modelling in feature models [3] and quality aware product configuration from feature models. With our tools, we can derive a product from a feature model with a full consideration of the customers' functional requirements as well as the customers' quality requirements in a more informed and rational way. Comparing with other related tool support, our tool provides a more complete support for software product line engineering. In the future, we would like to study how to model quality attributes into software product line architectures.

6 References

- [1] Sinnema, M. and S. Deelstra, *Classifying Variability Modeling Techniques*. Information and Software Technology, 2006. 49(7): p. 717-739.
- [2] Clements, P. and L. Northrop, *Software Product Lines: Practices and Patterns*. The SEI Series in Software Engineering, 2002, Boston: Addison-Wesley. 573.
- [3] Zhang, G., H. Ye, and Y. Lin. *Quality Attributes Assessment for Feature-Based Product Configuration in Software Product Line*. in Asian Pacific Software Engineering Conference. 2010. Sydney, Australia.
- [4] Hallowell, D., L., *Analytical Hierarchical Process (AHP)-Getting Oriented*. ISixSigma.com Retrieved 2007-08-21, 2007.
- [5] Botterweck, G., et al. *Visual Tool Support for Configuring and Understanding Software Product Lines*. in 12th International Software Product Line Conference. 2009. Limerick
- [6] Lee, K. and K.C. Kang. *Usage Context as Key Driver for Feature Selection*. in 14th International Conference on Software Product Line 2010. Jeju Island, South Korea, .
- [7] Yu, Y. and A. Lapouchnian. *Configuring Features with Stakeholder Goals*. in ACM Symposium on Applied Computing. 2008. New York.
- [8] Etxeberria, L. and G. Sagardui. *Variability Driven Quality Evaluation in Software Product Lines*. in 12th International Software Product Line Conference. 2008. Limerick.
- [9] Jarzabek, S., B. Yang, and S. Yoeun, *Addressing Quality Attributes in Domain Analysis for Product Lines*. Software, IEE Proceedings, 2006. 153(2): p. 61-73.
- [10] Zhang, H., S. Jarzabek, and B. Yang. *Quality Prediction and Assessment for Product Lines*. in Proceedings of 15th International Conference on Advanced Information Systems Engineering. 2003. Klagenfurt, Austria.
- [11] Siegmund, N., et al. *Scalable Prediction of Non-functional Properties in Software Product Lines*. in Software Product Line Conference. 2011. Munich, Germany.
- [12] Sinnema, M., O.d. Graaf, and J. Bosch. *Tool Support for COVAMOF*. in International Workshop on Software Variability Management for Product Derivation -Towards Tool Support. 2004.
- [13] Sincero, J., W. Schroder-Preikschat, and O. Spinczyk. *Approaching Non-Functional Properties of Software Product Lines: Learning from Products*. in 17th Asian Pacific Software Engineering Conference. 2010. Sydney, Australia.
- [14] White, J. and D.C. Schmidt. *Automating Product-Line Variant Selection for Mobile Devices*. in 11th International Software Product Line Conference. 2007. Kyoto, Japan.
- [15] Bagheri, E., et al. *Stratified Analytic Hierarchy Process: Prioritization and Selection of Software Features*. in 14th International Software Product Line Conference. 2010. Jeju Island, South Korea.
- [16] Chung, L., et al., *Non-Functional Requirements in Software Engineering*. International Series in Software Engineering. Vol. 5. 2000: Kluwer Academic. 476.
- [17] Benavides, D., et al. *FAMA: Tooling a Framework for the Automated Analysis of Feature Models*. in the First International Workshop on Variability Modelling of Software Intensive Systems 2007. Limerick, Ireland.
- [18] Zhang, G., H. Ye, and Y. Lin. *Feature Model Validation: A Constraint Propagation-Based Approach*. in SERP'11-10th International Conference on Software Engineering Research and Practice. 2011. USA.

A New Architecture for Logging and Auditing in Distributed Systems

Elnaz B. Noeparast, Reza Ravani

Department of Computer Engineering, Islamic Azad University, Central Tehran Branch, Tehran, Iran

Abstract - Due to widespread communications and unsafe accesses in distributed systems, controlling and auditing of events are considered as one of the major challenges to achieve security goals. Several methods have been introduced with some drawbacks such as the integrated information accuracy concern, server overhead and impossibility of client management in emergency situations. This paper presents a new architecture for logging and auditing systems. Tampering and losing data prevention, reducing server overhead during data collection and integration and notifying the clients' status to server to apply suitable security policy are the main goals of the proposed architecture. In our system architecture, system events are classified to four categories, negligible, marginal, critical and catastrophic. Only the information of critical and catastrophic events will be sent to the server during emergencies. Also, this paper presents a method for data encryption which has a validity period and it will be updated periodically from the server.

Keywords: Distributed Systems, Event-Oriented Architecture, Auditing, Data Integration

1 Introduction

With extensive operations between different systems in distributed systems, we need a mechanism that provides essential and necessary information for keeping track of important events. This is informed us about the status of each system during its operations and its operation's accuracy. This mechanism is system logs auditing which is considered as a necessary and valuable part of each system. Since recording events in distributed systems are performed in various and unsecure systems, on one hand, it is difficult to integrate information for analysis and on the other hand, it is possible to loss or manipulate information.

The previous researches have provided methods such as using public key cryptography [1,2], tamper-resistant hardware [3,4] and information symmetric encryption [3,5,6,7]. The main problems of these methods are their high implementation cost, need for being online during logging operations and the probability of the detection of encryption algorithm keys. As mentioned above, another problem is the information integrating from all systems for auditing and tracking system errors or detecting hacker attacks and illegal entries into the system. So far, the methods that have been proposed to resolve these problems are divided into two

groups. In one group, each system's logs are stored locally and alerts are generated on client-side, then the local information is transmitted to a central server through methods like network-based sensors [8]. In second group, in addition to storing data locally and generating warning on the client-side, some parts of audit operations are performed on client-side and then required result will be sent to server for analysis [9].

According to the mentioned problems, we need a method and system architecture that provides data integration from all systems (which may work offline), and also ensures the integrated information accuracy.

This paper organizes as follows: in the next section, first the Event- Driven Architecture is explained and then features of the proposed auditing system are presented. After that, a high-level architecture will be provided based on these features. This section also covers the description of the structure and the function of this architecture. In section 3, we will evaluate and compare our new architecture with previous methods. Finally we finish the paper with concluded remarks and future works.

2 System Architecture

Event-Driven Architecture (EDA) is an architectural style which includes two or more software components related together based on the events with minimal dependencies. Here, minimal dependency means that the relationship between sender and receiver is unidirectional and the sender doesn't transmit any data to receiver by receiving a request [10], rather when an event occurs, a notification is generated and sent to receivers which have subscribed for it before [11]. Accordingly, three components *publisher*, *subscriber* and *communication infrastructure* play the fundamental roles in this architecture [12]. Publisher is a software component that produces information and publishes them on communication infrastructure. Subscriber is the consumer (receiver) of some information that is produced and published by various publishers. This information is *events* and the method of providing them is called *notification*. The communication infrastructure is a *service bus* which manages events. Subscribers register in this bus for events that they want to have notified about their occurrence, without having any information about their publishers. The subscribed information is stored in the bus

and is not sent to publishers. By creating an event, the publisher notifies the bus about its happening through the publishing operation, and the bus informs the event subscribers through notification [13].

The architecture is presented in this paper is an EDA. By an event occurrence, server services are invoked which have registered for that event. This reduces the dependencies between server and client application and enhances the security, because communication infrastructure sends a message to the system which has registered for this event. Logging and auditing system proposed in this article has additional features and advantages as follows:

1- Protecting the security of local data:

To increase auditing data security in a client system and preventing the destructive manipulation, it is necessary to store this information in encrypted form and after sending it periodically to server, decryption will be done on the server-side. Instead of receiving the encryption key which has been stored in the client system in specific time-intervals, it is necessary that servers receive a signed encryption algorithm via a web-service. But since there is the probability to detect the encryption algorithm, data is still at risk. The encryption algorithm should be different for each client and could be changed and updated in regular time-intervals. To achieve this goal, this article applies Aspect-Oriented Programming (AOP) [14] on the client-side to utilize an encryption algorithm. Therefore, storing logs structure which contains an encryption algorithm is received from the server as a file in specific time-intervals. Then, the program stores it in a specific local path and uses it to store logs (audit data). Each time the new algorithm is sent to a client, its decryption algorithm and the relevant timing information are stored in the server database. So, whenever the server receives information from clients, it uses the decryption algorithm to decrypt data according to the recording time of data on the client-side.

2- Storing messages and preventing the lost

In distributed systems, sometimes it may not be possible to have a communication between server and client for some reasons such as the communication lines traffic, busy servers, the offline clients and etc. Thus, a mechanism should have used in such situations that can store messages and prevent losing messages. Since the role of service bus is enabling essential cooperation between various service components in a range of different platforms and it can also store messages inside the related queues [15], so it is a suitable option for this purpose. The proposed architecture uses this infrastructure for communication between the client and the server.

3- Reducing overhead due to audit data transmission from client to server

If transmitting audit data to the server is performed in short time-intervals, it increases overhead and server communication buses traffic. On the other hand, if this interval is long, it causes that the server does not aware of issues that happens on the client side. Thus, it is necessary to classify the events and audit data types, so each data is sent to the server at a specific time according to its classification. Therefore, audit data (different types of system events) are divided into the following four categories in this paper.

- *Negligible* events: An event that effects on non-operational functions and slightly enhanced the cost and time.
- *Marginal* events: This is an event that reduces the system functionality and its technical performance.
- *Critical* events: An event that not only causes the delay of system performance and increasing operating costs, but also it causes that some parts of the non-essential functions do not work properly.
- *Catastrophic* events: The events which prevent the normal operation of the essential parts of the system are catastrophic. Some of these events may even lead to disconnect the client completely.

If the audit data type is critical or catastrophic, it is sent to the server as soon as it occurs. The difference between these two types of data is the transfer mechanism. In catastrophic events, audit data will be sent through HTTP protocol in text-based format to reduce transmission time, but sending critical audit data will be done again through the HTTPs protocol but in encrypted format. In the latter case, if the message is received by a third party, it could not be aware of the semi-safe situation of the client. Sending periodic audit data based on the time intervals is specified in the configuration file and it includes marginal, critical and catastrophic data. Negligible audit data is sent to server only when critical or catastrophic data is available in the local database. In this case, these audit data will be transmitted together with critical or catastrophic data. This information may help for more accurate analysis of the critical or catastrophic incidents on the server-side.

4- Increasing the client flexibility due to server changes

The configuration file contains information about the address of services which are offered by the server, the specific scheduling for client to send data periodically to server and service names that are subscribed for various types of events. This file can be received from the server whenever a client connects to the server. This increases the client flexibility to updates itself if the scheduling policies and service addresses changes on the server side. Also, the type and priority of the audit data could be changed on server-side based on analysis on the received audit data from clients. So, it is necessary for clients to receive updated data type configuration and recognition settings in regular time-intervals from the server.

Till now, we have presented our system architecture features and characteristics. Now, we will more focus on the client-side and server-side detailed architectures and workflow.

2.1 Client-Side Architecture

Auditing system in client-side follows component-oriented design. For this purpose, this system includes 8 different components which are explained as follows. The block diagram and relationship between different components has been shown in Figure 1.

- **Organizer component:** This component is responsible for five major following tasks:
 - Receives configuration file from the server: Each time the client system is connected to the network, Organizer component sends a request to the server via Service Provider component and based on SSL protocol. After receiving the configuration file, it saves this file in a specific local path for future purpose.
 - Receives encryption algorithm from the server: This component receives Logger component assembly file from the server in regular time-intervals periodically and saves it to the program installation path. This assembly file contains the encryption algorithm to encrypt the log data.
 - Receives the audit data type recognition settings: Organizer component sends a request to server in regular time-intervals via Service Provider component for requesting this file. Then it will received an XML file which will be saved in a specific local path.
 - Publishes critical and catastrophic data: as soon as Organizer component detects a critical or catastrophic type of audit data (which indeed has been acquired by Auditing component and forwarded to Organizer), it publishes this data via Service Provider component. The detection configuration settings is based on XML pattern recognition file.
 - Publishes audit data periodically: According to the scheduling which is listed in the configuration file, Organizer component publishes audit data to the server via Service Provider component.
 - **Service Provider component:** This component communicates with server through HTTP and HTTPs service protocols. It consists of two types of service calls. Invoking services which subscribe for special events and calling services that are invoked directly. Calling subscription services is for sending critical and catastrophic event data when it happens and sending audit data periodically. Invoking methods directly is used for receiving Logger assembly settings,
- configuration and pattern recognition files from the server via HTTPs protocol.
- **Service Provider Interface:** This component will handle the communication with services offered by server through the addresses and "service contracts" that have been registered before.
 - **Auditing Interface:** This interface provides methods for recording the information related to an action or changing an object. Since Audit component is considered as a part of a software system in this paper, an explicit call to Audit component is required for performing relevant operations.
 - **Logger component:** This component is responsible for encryption and storing data in local database. This component is an assembly file which is generated by the server based on each request and has a validation time-interval.
 - **Event Aggregator component:** In case of critical or catastrophic event, this component invokes related subscribed service from Service Provider component. This component also sends audit data periodically whenever the system timing will be equal to schedule timing stored in configuration file [16].
 - **Auditor component:** This component is responsible to analyze and refine the information which is provided by the system software. This information is converted to data audit after analysis.
 - **Aspect Loader component:** This component operates based on the AOP rules and standards. It loads the Logger component assembly from a specific path of the installed program and then stores an encrypted audit data into the local database.

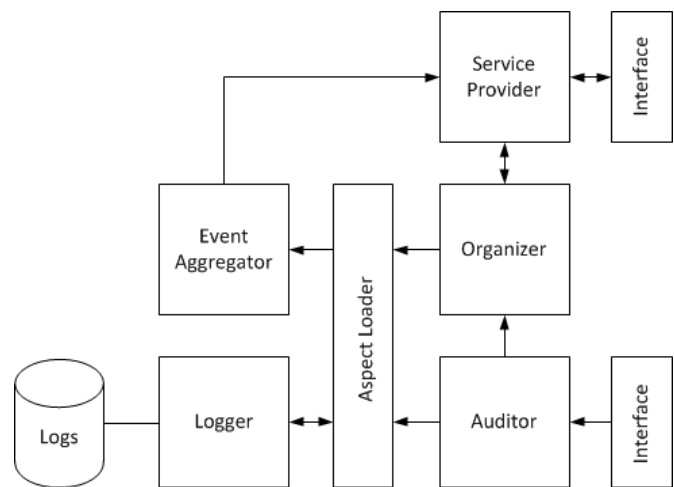


Figure 1 The client-side auditing system architecture

The workflow diagram of client-side auditing system is shown in Figure 2. By initializing Audit module, the software system sends required information to the Audit component by a method call whenever an action is performed by user or an object is changes. After analyzing and refining information in Audit component, the audit data is sent to the Organizer component through a method call. Then the Organizer component publishes an audit data according to the pattern recognition file if the data type was critical or catastrophic. The Event Aggregator component extracts the service address which is subscribed for this event from the configuration file. Then it sends a notification to the server via Service Provider component. Parallel to this operation, the Auditor component sends an audit data to the Aspect Loader component by a method call. After that, this component initializes Logger component by using AOP methods and stores an encrypted audit data in the local database. This operation repeats if the Auditor component's method is called again.

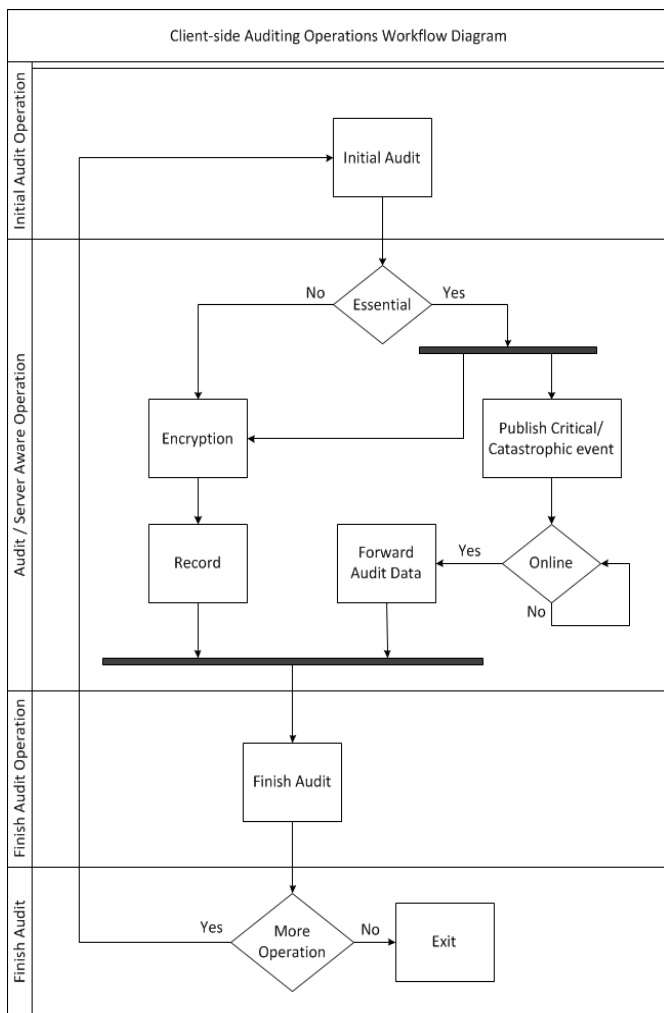


Figure 2 Workflow diagram of the client-side auditing system

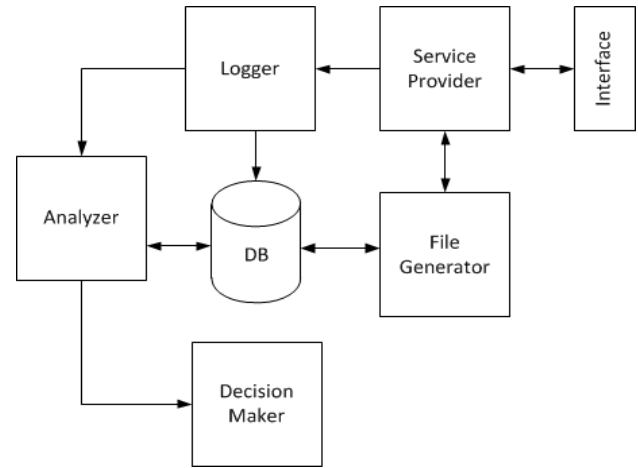


Figure 3 The server-side auditing system architecture

2.2 Server Side Architecture

The server's Auditing system is responsible for the information collection and integration from clients and the management/coordination of clients audit process. It includes 6 different components which are explained as follows. The block diagram and relationship between different components has been shown in Figure 3.

- **Decision Maker component:** This component makes decision about clients which a critical or catastrophic events has been occurred during their operation. Considering different condition, this decision could be notifying system administrator, disconnecting client temporary or etc. based on the configuration files have been specified by system administrator in the server's database as security policies.
- **Analyzer component:** This component analyzes the reasons of a specific event based on the collected auditing data and original configuration data in the database. Then it sends the results of this analysis to the Decision Maker component through a method call.
- **Service Provider Interface:** This interface shows the service addresses and contracts for invoking services offer by server by clients Auditing component.
- **Service Provider component:** This component provides following services:
 - A secure service for receiving audit data periodically
 - A secure service for receiving critical type of audit data
 - A text-based service for receiving catastrophic type of audit data
 - A service for providing client requested files

- **File Generator component:** This component generates a configuration file, an event type recognition Pattern file (XML file) and an assembly file of client's Logger component which consists of the encryption algorithm. These file will be generated according to the time and the client specifications like client's identifier code, username and password.
- **Logger component:** By receiving auditing data, this component retrieves its decryption algorithm from the database based on the data creation time and client specifications. Then, it decrypts the data and stores it in the database. For critical or catastrophic audit data types, its creation time is equal to the receiving time. But if an audit data has been sent periodically, its creation time is the last time when the client has received a Logger assembly file (this data is encrypted by the last assembly).

Considering above details specification of server-side auditing system, the auditing system workflow is explained on the following way (Figure 4). When a service from the server is invoked by a client and it is about a receiving file request, then the desired file is generated by File Generator component and will be sent to the client. But if this is a request for data collection and integration invoked by server itself, then this data is decrypted by the Logger component and stored in database. After that, it is sent to the Analyzer component by a method call. Analyzer component receives this data and analyzes it based on the information which has been stored in database. If the reasons of the problem are unauthorized entry, data manipulation or hacker attacks, then a request is sent to the Decision Maker component. This component makes a decision according to the stored security policies in database which the server administrator has defined them in advanced.

3 Architecture Analysis and Evaluation

As mentioned in Introduction section, there are different methods for the audit data protection. One of the simplest techniques is to use a bug-free tamper-resistant hardware (to prevent audit data manipulation) and maintaining a secure communication channel between clients and the server (to upload runtime data) [17]. Forwarding secure signature is another proposed scheme for storing secure information. In forwarding secure signature, a sender digitally signs each data item as soon as it is stored. Then the sender improves its secret keys, removes previous ones and uses new keys for signing next data items [18]. A group of these schemes relies on the use of the symmetric cryptography and the others utilizes the Public Key Cryptography (PKC). Each of these methods has its own benefits and drawbacks that are comparable with our proposed architecture in this paper.

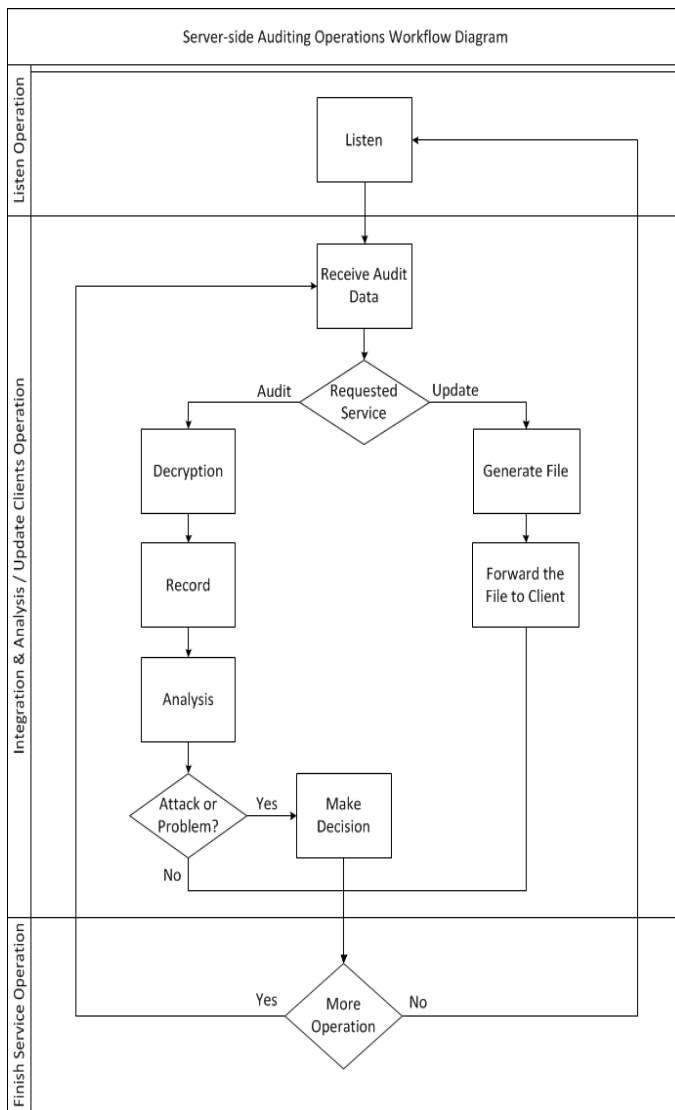


Figure 4 Workflow diagram of the server-side auditing system

- **Efficiency:** A group of schemes which relies on the symmetric cryptography uses Message Authentication Codes, semi-random number generation or one-way hash chain for data encryption, all of these methods result to computational complexity [17]. The proposed algorithm for encryption could generate better (in worse case equal) performance results, depending on the encryption policy.

- **Storage and Communication Overhead:** Due to accumulating signatures from data items in forwarding secure signature method and also the need to maintain and transmit an authentication tag for each log, signing data items continuously causes to significant communication and storage overhead [17]. But in the proposed architecture, encryption algorithm is generated according to each client in the Logger file, which is replaced the previous one on the client-side. So it is not required to store any information about this algorithm, because its generated time would be store in the server database. With this method, the client storage overhead

is decreased. Also in this architecture, only essential data is sent to the server as soon as it happens, as a result, the communication overhead is reduced.

- *Ability of Public Audit:* Methods use symmetric encryption could not offer public audit. Because they need a complete symmetric keys distribution or an online TTP support. The complete symmetric keys distribution brings a significant storage overhead for system entities. Supporting online TTP causes architectural difficulties, increases the communication overhead and makes vulnerabilities against the system attacks [17]. But in our architecture auditing operations will be done on the client-side and the results of this process will be sent to the server for analyzing. By this method, operation overhead on the server-side is reduced. Essential filtered information is sent to the server, so the server doesn't process unnecessary or insignificant information of clients.
- *Applicability:* Maintain a secure communication channel between the clients and server for modern computer systems is impractical. Because in distributed systems, it is not possible to reserve an end-to-end real-time communication channel between the client and server. Also, it could not be assumed that a tamper-resistance hardware (which must be bug-free) will guarantee to be compatible with all platforms [19]. The proposed architecture can be used in distributed systems and by all distributed applications, because of its low structural/data coupling between client and server. Also, this architecture could be implemented on all platforms as long as they support programming languages which supports AOP, Event and Service concepts. In this architecture, the server information is easily updatable on the client-side, so it is possible to change or extend the information to the multiple servers.

4 Conclusions and Future Works

In this paper, an event-oriented architecture in a distributed system for log, collecting/integrating and analyzing of audit data has been proposed to manage and coordinates separate nodes with different tasks. In this architecture, Aspect-Oriented Programming (AOP) will be utilized on the client-side to encrypt audit data. This encryption algorithm has a limited validation time and after expiration new configuration for another algorithm will be generated and transmitted by server to client to update its current algorithm. During audit data collection and integration, server decrypts each audit data with a specified algorithm considering to its encrypted time. So, the possibility of unauthorized access to encryption algorithm and consequently audit data will be decreased.

We have also presented a classification and transmission method of audit data to server to reduce the

server overhead due to receiving different types of audit data from different clients while it could be informed when an important event happens in a client. As a result of comparing our architecture and previous methods, it can be easily understood and verified that the proposed architecture has a more general applicability than previous ones. Also, the proposed method provides ability of public audit in distributed systems and causes less communication and memory overhead for these systems. The infrastructure service bus details has not been mentioned in this paper, because this bus can be any of-the-shelf (COTS) bus depends on the platform.

A high-level architecture for a logging and auditing in distributed system has been discussed in this paper. In the near future full implementation of this architecture considering system-level design limitations and concerns will be presented. Using autonomic computing for the Decision Maker component in server or anticipating the weakness points of distributed systems by applying the integrated audit data analysis results, are some of the challenges that need to be studied more.

5 References

- [1] Rafael Accorsi. "BBox: A Distributed Secure Log Architecture"; Proceedings of the 7th European conference on Public key infrastructures, services and applications (EuroPKI'10), Vol. 6711, 109—124, 2011.
- [2] Wensheng Xu, David Chadwick, Sassa Otenko. "A PKI-based Secure Audit Web Service"; Proceedings of the IASTED International Conference on Communication, Network, and Information Security (CNIS 2005), Nov 2005.
- [3] Di Ma, Gene Tsudik. "Forward-Secure Sequential Aggregate Authentication"; IEEE Symposium on Security and Privacy, 86—91, 2007.
- [4] Daniel Halperin, Thomas S. Heydt-Benjamin, Kevin Fu, Tadayoshi Kohno, William H. Maisel. "Security and Privacy for Implantable Medical Devices"; IEEE Pervasive Computing, Vol. 7 Issue. 1, 30—39, Jan 2008.
- [5] Mihir Bellare, Bennet Yee. "Forward-security in private-key cryptography"; Proceedings of the 2003 RSA conference on The cryptographers' track (CT-RSA'03), 1—18, 2003.
- [6] Bruce Schneier, John Kelsey, Bruce Schneier, John Kelsey. "Cryptographic support for secure logs on untrusted machines"; In Proceedings of 7th USENIX Security Symposium, 53—62, 1998.
- [7] Bruce Schneier, John Kelsey. "Secure audit logs to support computer forensics"; ACM Transactions on

Information and System Security (TISSEC), New York, NY, USA, Vol. 2 Issue. 2, 159—176, May 1999.

Proceedings of 2012 Financial Cryptography and Data Security (FC 2012), 2012.

[8] Ambareen Siraj, Rayford B. Vaughn, Susan M. Bridges. “Intrusion Sensor Data Fusion in an Intelligent Intrusion Detection System Architecture”; Proceedings of the 37th Hawaii International Conference on System Sciences (HICSS'04), Jan 2004.

[9] Mike Davis, Ed Coyne, Craig Winter. “Security Audit architecture For Audit As A Service”, Office of CIO, Health Information Architecture Office, Feb 2006, http://hssp-security.wikispaces.com/file/view/HIA_20060310_Security+Audit+Architecture+V1.doc, Downloaded on [28 04 2012].

[10] K. Mani Chandy. “Event-Driven Applications: Costs, Benefits and Design Approaches”; Presented at the Gartner Application Integration and Web Services Summit, Jun 2006.

[11] K. Mani Chandy, W. Roy Schulte. “Event Processing: Designing It Systems for Agile Companies”. McGraw-Hill. 2010.

[12] David Trowbridge, Ulrich Roxburgh, Gregor Hohpe, Dragos Manolescu, E.G. Nadhan. “Integration Patterns”. Patterns & Practices . Avail. at www.microsoft.com. 2004.

[13] Patrick Th. Eugster, Pascal A. Felber, Rachid Guerraoui, and Anne-Marie Kermarrec. “The many faces of publish/subscribe”; ACM Computing Surveys, Vol. 35 Issue. 2, 114—131, Jun 2003.

[14] Dharma Shukla, Simon Fell, and Chris Sells. "Aspect-Oriented Programming Enables Better Code Encapsulation and Reuse"; MSDN Magazine, Mar 2002.

[15] David A. Chappell. “Enterprise Service Bus”. O'Reilly Media. 2004.

[16] Martin Fowler: Information on <http://martinfowler.com/eaaDev/EventAggregator.html>

[17] Attila Altay Yavuz, Peng Ning. “BAF: An Efficient Publicly Verifiable Secure Audit Logging Scheme for Distributed Systems”; Proceedings of the 2009 Annual Computer Security Applications Conference, 219-228, 2009.

[18] Attila Altay Yavuz, Peng Ning. “Hash-Based Sequential Aggregate and Forward Secure Signature for Unattended Wireless Sensor Networks”; Proceedings of the Sixth Annual International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services (MobiQuitous 2009), Jul 2009.

[19] Attila Altay Yavuz, Peng Ning, Michael K. Reiter. “Efficient, Compromise Resilient and Append-only Cryptographic Schemes for Secure Audit Logging”; in

Requirements Metrics for Requirements Statements Stored in Database

Chao Y. Din¹, David C. Rine²

¹NuWave Solutions LLC, McLean, Virginia, USA

²Computer Science Department, George Mason University, Fairfax, Virginia, USA

Abstract - *In a software development project, a requirements document, either a hard copy or stored in a database, summarizes the results of requirements analysis and becomes the basis for subsequent software development. In many cases, the quality of the requirements statements dictates the success of software development. The need for determining the quality of requirements statements is acute when the target applications are large, complicated, and mission critical. The purpose of this research is to (1) summarize our previous research on the quality indicators that indicate the quality of requirements statements in a requirements document and (2) report our current research to expand the previous quality indicators to evaluate the quality of requirements statements stored in a database. A suite of complexity metrics of requirements statements is proposed as quality indicators and is developed based upon research of noun phrase (NP) chunks.*

Keywords: Metrics, Cohesion, Coupling, Requirements, Software Quality

1 Introduction

This paper asserts that a set of requirements statements either documented in a requirements document or stored in a database, is the single artifact produced through the requirements engineering process. Its quality inevitably becomes the main focus of requirements management. Research studies repeatedly confirm that requirements quality has the greatest impact on the overall quality of software applications and hence has been associated with the highest repairing cost [12,23,27].

The purpose of this research is two folds: (1) summarize our previous research on a set of metrics to indicate the quality of requirements statements in a requirements document and (2) propose a modified set of metrics to indicate the quality of requirements statements stored in a database. The quality factors are presented by a set of goodness properties. The indicators will be able to identify requirements statements with low goodness property values.

The previous research [4] used statistical and partial parsing approaches to obtain a subset of noun phrases, named Noun Phrase (NP) chunks. Abney indicated that chunks are the basic language parsing unit, and they correspond to “the basic concepts” for human brains to comprehend a text document [1]. NP chunks are hence adopted as the basic processing units in this research.

The previous research [4] developed three core complexity metrics: count of NP chunks (NPC-Count), cohesion of

requirements sections (NPC-Cohesion), and coupling of requirements sections (NPC-Coupling).

A two-phased empirical case study was performed [4] to evaluate the proposed complexity metrics. Phase I of the case study compared the NPC-Cohesion and NPC-Coupling metrics with the cohesion and coupling metrics proposed by Ricker [24]. Ricker's research demonstrated the correlation between the complexity metrics and understandability, or comprehension, of the requirements. By demonstrating the consistency between the two sets of metrics, the previous research [4] proved to be correlated to understandability, one of the goodness properties, of the requirements statements. Furthermore, the case study showed that the NP chunk based complexity metrics possess the following two additional capabilities: (1) they differentiate nouns from other syntactic categories (or word classes) – an important capability to differentiate object methods and properties from object classes, and (2) they adopt the spatial distance of NP chunks as the measuring units – an important capability in a cognition complexity model [2].

The phase II case study then demonstrated how the three core metrics can be used to identify low quality requirements statements.

Based upon the two phased case study, it was assured that the proposed complexity metrics indicate the content goodness properties of requirements.

This paper further discusses updates to the previously developed requirements metrics for requirements statements stored in a database. Requirements statements printed on a requirements document is presented in a linear order, while requirements statements stored in a database can be viewed in various orders depending on the interaction between the users and the viewer applications.

2 Research Problem and Its Importance

How to identify low quality requirements statements in a requirements document or in a database is an intricate research question. This research answers the question in a constrained environment where the current best practices of identifying requirements and eliminating requirements defects are adopted. The constraints are as follows.

- 1) A systematic requirements method such as Viewpoint Oriented Requirements Definition (VORD) has been followed to produce the requirements statements [28].
- 2) The requirements statements are grammatically correct and spelling errors have been checked.
- 3) Traditional requirements guidelines to avoid ambiguous terms (large, many, user friendliness) and weak phrases (as applicable, as required, as a minimum) have been followed [25].
- 4) A domain thesaurus and/or company term definitions have been supplied.

- 5) A requirements inspection method has been adopted to eliminate requirements defects.

Certain requirements defects are hard to identify and remove. Capers Jones concluded in his study that (1) formal inspection is the most efficient way to remove defect and (2) formal inspection together with defect prevention can achieve the best overall quality [12]. This research will expand the current practices in identifying low quality requirements, and hence preventing requirements defects. However, this research will not and has never intended to replace the current practices in identifying requirements defects.

The proposed suite of complexity metrics can be used to identify high complexity and hence low quality requirements. Once low quality requirements are identified, analysis of the low quality requirements can be conducted so that they can be classified into categories of potential risks. Appropriate management actions can then be considered. Wilson, Rosenberg & Hyatt [29] identified several categories of system risks due to low quality requirements.

Low quality requirements are not only the source of system product risks but also the source of system development resource risks, which includes cost overrun and schedule delay. Using the proposed suite of complexity metrics as quality indicators, an impact assessment and threats classification of the identified low quality requirements can be performed. Such an early warning is vital to rescue a possibly failing project.

The development of high quality systems depends on management's awareness of such low quality requirements, their ability to expediently assess the impacts of those low quality requirements, and the capability to develop a plan to rectify the problem. This identification of low quality requirements and the subsequent risk analysis of those requirements provide the foundation for the development of high quality systems.

The proposed complexity metrics to indicate the quality of requirements statements can be developed using computer programs. This will provide a way to quickly identify potential requirements defects, and hence substantially improve the time to identify low quality requirements and expedite the rescue of a troubled project.

3 Background

3.1 Quality and Content Goodness Properties

Schneider proposed 11 goodness properties as a better coverage of quality factors [26]: Understandable, Unambiguous, Organized, Testable, Correct, Traceable, Complete, Consistent, Design independence, Feasible, and Relative necessity.

In this research, the main concerns are the four goodness properties: Understandable, Unambiguous, Organized, and Testable. These four goodness properties are named as *Content Goodness Properties* and are the only goodness properties on which the remainder of the research will focus.

3.2 Complexity, Complexity Metrics and Measurement

Complexity is a major system characteristic that controls or influences quality. It has been widely accepted as an indirect indicator of quality and hence the content goodness properties [8,10,14,16]. The remainder of the research focuses the discussion on complexity.

Purao and Vaishnavi [21] provide a survey of complexity metrics and identified five out of 375 metrics that are related to requirements. Unfortunately, none of those five metrics are used to measure the natural language descriptions of the requirements. Ricker's research [24], not listed on the survey, developed a set of requirements metrics: cohesion, context, and coupling. One of the contributions [24] made is the demonstration of a positive correlation between cohesion, context, and coupling metrics and understandability of requirements statements. In [24], the context metric is assessed by the relationships between the sentences of a section and their section title. This research does not consider the context metric.

In another survey published about the same time, seven research papers were identified that were relevant to requirements metrics for documents written in natural languages [19]. Two of the research papers discussed the same research study. Hence six research studies were identified. However, the main focus of most of the six research studies were quality attributes of requirements, and most of the quality indicators presented in those research studies depended heavily on human intervention to collect the measurement values. For example, understandabilities were defined as the number of requirements that could be understood divided by the total number of requirements. Here the number of requirements that could be understood could only be manually determined through an expensive data collection effort. Two of the six research studies did develop automated tools to measure requirements [7,25]. Unfortunately, these automated tools tried to identify ambiguous terms and weak phrases, which is a prerequisite of this search.

A more recent research on requirements metrics presented by Iqbal and Khan [11] also defined the understandability metric as the number of requirements that could be understood divided by the total number of requirements, which again requires heavy human intervention to obtain the measurement value.

3.3 Readability Index

When measuring the quality of documents written in natural languages, the readability indexes or metrics may be considered. In general the written communication skills are measured in terms of readability and hence the use of readability indexes. Readability indexes are designed to access the suitability of a piece of writing for readers at particular grade levels or ages.

Factors considered in the readability indexes are number of words, number of syllables in words, number of words in sentences, ..., etc. Scores of the readability indexes are compared with scales based on judged linguistic difficulty or reading grade level.

Unfortunately, readability indexes are not comparable with our research for the following reasons:

- 1) The readability indexes are designed for the whole documents, instead of sections of documents.
- 2) The readability scores are not reliable indicators when the document under evaluation has less than 200 words [17]. However, many of the requirements statements have less than 50 words.
- 3) The definition of cohesion used with readability indexes is different from the definition of cohesion used in Computer Science, and there are no coupling metrics for readability indexes [9,18].

4 NP Chunk Based Complexity Metrics

Because humans tend to read and speak texts one chunk at a time, Abney proposed using what is called *chunks* as the basic language parsing unit. There are several categories of chunks similar to the traditional categories of phrases. For example, there are Noun Phrase (NP) chunks, Verb Phrase (VP) chunks, Prepositional Phrase (PP) chunks, ... etc [1]. Our research focuses on NP chunks and ignores other types of chunks.

4.1 Three Core Metrics

It is believed that a small subset of existing metrics can enable parsimonious evaluation, prediction and control of software complexity [13]. Our research hence proposes three types of complexity metrics, NPC-Count, NPC-Cohesion, and NPC-Coupling, for measuring the complexity of requirements statements in a requirements document and in a database.

Size counts are the oldest method of measuring complexity. For software design and coding, the most popular size count is Line of Code (LOC). The wide acceptance of LOC as a complexity metric is due to its simplicity, ease of application, inertia of tradition, absence of alternative size metrics, and its intuitive appeal [5, 15]. Based upon the above reasons, two distinct metrics (NPC-Sentence and NPC-Req) are developed to count the NP chunks of a text, and these two metrics are collectively named as NPC-Count.

Darcy and Kemerer believe that cohesion and coupling are effective metrics and they can represent the essential complexity measures for the general software design tasks [3]. Hence, NPC-Cohesion and NPC-Coupling are chosen in our research to represent the complexity of requirements. To assist the identification of low quality requirements, a composite metric (NPC-Composite) that combine cohesion and coupling measures is also proposed and studied in the research.

In addition, Regnell etc [22] suggested that "*the complexity of a set of requirement is heavily related to the nature of interdependencies among requirements.*" The cohesion and couple metrics presented in this research is one way to measure the interdependencies among requirements.

4.2 Sentence/Requirements Statement Level Complexity

The sentence level complexity metric, or NPC-Sentence, can be calculated as follows. For each NP chunk, the occurrence count in a sentence is divided by the total occurrence counts in all sentences. Then all the frequency distributions of the NP chunks in the sentence are added together to form the final complexity value.

The requirements statement level complexity metric, or NPC-Req, is the aggregation of NPC-Sentence of the component sentences.

The above calculation for the sentence and requirements level complexity is applicable to requirements statements in requirements documents and requirements statements stored in a database.

4.3 Intra-Section Level Complexity

The NPC-Cohesion metric for requirements statement in a requirements document is a normalized cluster size that can be calculated using the sum of all cluster sizes in a requirement section divided by the size of the requirements section. Here a cluster is

defined as the collection of adjacent sentences in a requirements section that shares the same NP chunks. For example, if sentence 1 contains NP chunk A, sentence 2 contains NP chunk A and B, and sentence 3 contains NP chunk B, then the three sentences form a single cluster.

On the other hand, the requirements statements in a database do not possess the spatial distance property and the type of clusters we defined. A simple NPC-Cohesion metric is defined as the number of NP chunks appears in a requirements section divided by the product of the total number of NP chunks and the total number of sentences in the requirements section. For example, assume there are five different NP chunks in a requirements section that contains six sentences. If the five NP chunks appears 15 times in the section, the NPC-Cohesion is $15/(5 \times 6) = 1/2$.

4.4 Inter-Section Level Complexity

The NPC-Coupling metric for a requirements document is the sum of the spatial distances between its internal and external NP chunks. If an NP chunk belongs to a cluster, then the centroid of the cluster is used to calculate its distance to the external NP chunks.

The NPC-Coupling metric for requirements statements stored in a database can be calculated as follows. For each sentence in the current requirements section, count the number of sentences in other requirements sections that have common NP chunks. Adding all these counts together and then divided it by total number of sentences outside the current requirements section. The result is a normalized NPC-Coupling metric value.

5 Empirical Case Study

An empirical case study [4] was conducted and the results confirmed the NP chunk based metrics are more effective than the term based metrics. More importantly, the proposed metrics can serve the quality indicators for requirements statements in requirements documents.

First, the previous research on NP chunk based metrics for requirements statements in a requirements document is presented. Requirements documents exhibit a linear order on requirements statements. Hence the concept of spatial distance and cluster distance can be applied.

This section of the paper then describes the current research that revises the NP chunk based metrics so that they can be applied to requirements statements stored in a database, which does not exhibit a linear order on requirements statements.

5.1 Case Study Methodology

The case study methodology [30] is an empirical research strategy commonly used in psychology, sociology, political science, social work, business, community planning, and economics. The case study methodology adopted here consists of five components, which form a logic plan for the research design of case studies.

- 1) A study's questions
- 2) Study propositions, or hypotheses
- 3) Unit(s) of analysis
- 4) The logic linking of the data to the propositions
- 5) The criteria for interpreting the finding

There are three types of case studies: exploratory, descriptive, and explanatory. In a nutshell, an exploratory case study is either used to define the questions and hypotheses of a subsequent case

study or to determine the feasibility of a subsequent research, i.e., an explanatory case study. A descriptive case study presents a complete description of a phenomenon. An explanatory case study explains the cause-effect relationships indicated in the research question [30].

5.2 Two Phased Case Study for Requirements Statements in a Requirements Document

The goal of our previous research [4] was to answer the constraint question about identifying low quality requirements statements in a requirements document with the specified constraints. This question was divided into two sub-questions.

- Q1. Can NP chunk based complexity metrics be more effective than the term based complexity metrics in terms of measuring requirements content goodness properties?
- Q2. How and why NP chunk based complexity metrics measure the content goodness properties of requirements statements?

The two phases of the case study, exploratory in phase I and explanatory in phase II, were designed to answer the two sub-questions, respectively. The first sub-question and hence the phase I case study was an evaluation of NP chunk based complexity metrics. If NP chunk based complexity metrics could not produce consistent results as the term based complexity metrics proposed by [24], there was no reason to perform further study on the research question. Ideally, the NP chunk based complexity metrics should be more effective than the term based complexity metrics; otherwise, the research provided little contribution to the research question.

The second sub-question and the corresponding phase II case study assumed the phase I case study had positive outcomes. The phase II study then explained how and why the NP chunk based complexity metrics worked. Evidence and findings to support the proposed metrics were presented one by one in this case study.

5.3 Exploratory Case Study – Phase I for Requirements Statements in a Requirements Document

The five components of the phase I case study are described as follows.

Study Question: The study question was “Can NP chunk based complexity metrics be more effective than the term based complexity metrics in terms of measuring requirements content goodness properties?”

Study Propositions, or Hypotheses: The purpose of the phase I case study was to determine whether the NP chunk based complexity metrics could measure content goodness properties of requirements documents. The term based complexity metrics published by Ricker [24] revealed positive correlation to understandability, one of the content goodness properties of requirements documents. Ernst and Mylopoulos [6] also adopted terms, or keywords, to measure quality requirements. Since their measures are not related to understandability, a comparison with their research cannot be performed.

Another term based system is presented by [20], where two sets of requirements for the same project were compared against each other. One set of the requirements came from customers, and the other set of requirements came from product development team. Again, it is not appropriate to compare our research with [20].

The derived specific study hypotheses/propositions were as follows.

- P1. Consistency: The NP chunk based complexity metrics are consistent with the term based complexity metrics.

Ricker proposed three term based complexity metrics: context, cohesion, and coupling, for requirements statements. However, the published metric values, or measures in [24] focused mainly on the cohesion and coupling metrics. The only metrics that could be compared against were cohesion and coupling metrics. Hence, the above proposition was divided into the two sub-propositions: one for cohesion and the other for coupling.

- P1.1. Cohesion: NPC-Cohesion, the NP chunk based cohesion metric, is consistent with the term based cohesion metric.
- P1.2. Coupling: NPC-Coupling, the NP chunk based coupling metric is consistent with the term based coupling metric.

For simplicity reason, the degree of consistency for the above two propositions were categorized into three ordinal values: strongly consistent, somewhat consistent, and cannot-determine. The degree of consistency must be strong in order to claim the two sets of metrics were consistent to each other.

- P2. Sensitivity/Accuracy: The NP chunk based complexity metrics are either more sensitive or more accurate than the term based complexity metrics.

The degree of sensitivity or accuracy was categorized into three ordinal values: strongly sensitive/accurate, somewhat sensitive/accurate, and cannot-determine. The degree of sensitivity/accuracy must be strong to claim the proposed metrics were more sensitive or accurate than Ricker’s metrics.

- P3. Additional Information: The NP chunk based complexity metrics can provide additional information on the requirements content goodness properties than the term based complexity metrics.

The linking of derived data to the above proposition was categorized into two ordinal values: “yes” (it provides additional information) and “no” (it does not provide additional information).

Unit(s) of Analysis: The unit of analysis for the phase I case study was a requirements document of a Federal Aviation Agency (FAA) project available in [24].

The Logic Linking of the Data to the Propositions Criteria for Interpreting the Findings: The logic linking of the data to the propositions was the first step of data analysis in the case study design, which was divided into two sub-steps: cohesion and coupling. The second step of data analysis was to interpret the findings using the evaluation criteria stated above.

Cohesion Metrics: Based upon the proposed NPC-Cohesion metric defined previously, the NPC-Cohesion measures and the cohesion measures published in [24] were consistent with each other except in one section – section 11 of the FAA requirements document.

The mismatch between the two cohesion metrics could be explained as follows. Section 11 of the FAA requirements document consisted of two sentences. By examining the two sentences, it was found that there were no common NP chunks between the two sentences. This was why the NPC-Cohesion metric gave a low cohesion measure for the above requirements section. On the other hand, Ricker used terms to measure the cohesion of the section, and the word “outputs” appeared in the first sentence as a noun, while the word “output” appeared in the second sentence as a verb.

Ricker's algorithm did not consider syntactic categories and hence linked the two sentences.

It is believed that a word in different forms, i.e., verbs and nouns, in different sentences should not always be considered as cohesive, since the two words in the two forms can refer to two totally different objects. When closely examining the two sentences, it was found that the word "output" in the two sentences indeed refers to two different things or two different concepts. Hence, the proposed cohesion metrics was more effective.

As indicated above, the evaluation criterion for the cohesion proposition (P1.1) was whether the two sets of metrics were strongly consistent with each other. Since there was only one mismatch and the mismatch can be explained, the degree of consistency was strong. For the additional information proposition (P3), the evaluation criterion for linking the data to the proposition was whether the NP chunks based complexity metrics could provide additional information. Since the NP-Cohesion metric could differentiate word classes, the NP-Cohesion metric did provide additional information. It was hence concluded that NPC-Cohesion supported proposition P1.1 and P3.

Coupling Metrics: The coupling measures based on the NPC-Coupling metric were consistent with the coupling measures in [24] except in one section – section 4 of the requirements document.

The discrepancy between the two coupling metrics could be explained as follows. Section 3 was titled as "routing processing", and Section 4 was titled as "additional routing processing." Since the fourth section was a supplement to the third section, its coupling in Ricker's method was very high.

On the other hand, the coupling value for section 4 was low for the NPC-Coupling metric because the spatial distance between the two sections was low. In other words, the effect of spatial distance was counted in the NP-Coupling metric, while Ricker's method did not consider the spatial distance.

Since there was only one mismatch and the mismatch could be explained, the degree of consistency was strong. For the additional information proposition (P3), the evaluation criterion for linking the data to the "additional information" proposition was whether the NP chunks based complexity metrics could provide additional information. Since the NPC-Coupling metric could measure spatial distance, the NPC-Coupling metric did provide additional information. It was hence concluded that NPC-Coupling supported proposition P1.2 and P3.

Sensitivity/Accuracy: The NPC-Cohesion metrics are relative measures. They are normalized and fall in the range of 0 to 1. Comparing such relative measures derived from different requirements documents may not be logical. Hence, it could not determine whether P2 proposition was supported or not.

Although the NPC-Coupling metrics for requirements statements in a requirements document were based upon spatial distance between NP chunks, they were not normalized. Comparing NPC-Coupling metrics with Ricker's metric which used different units of measurement did not seem to be logical either. All in all, the evaluation for the derived data from the case study and the P2 proposition resulted in the "cannot-determine" ordinal value.

Summary: Based upon the above analysis, it were concluded that the derived data from the case study met the evaluation criteria for the consistency proposition (P1) and additional information proposition (P3). On the other hand, no evidence supported the

opposite argument. Hence, the phase I study question was asserted. It is clear that the NP chunk based complexity metrics were more effective than the term based complexity metrics.

5.4 Explanatory Case Study – Phase II for Requirements Statements in a Requirements Document

Study Question: The phase II study question was "How and why NP chunk based complexity metrics measure the content goodness properties of requirements statements?"

Study Propositions, or Hypotheses: The study question above was decomposed into three propositions.

- P4. NP Chunk Counts: The NP-Count, as a simple form of complexity metric, can measure the content goodness properties of the requirements statements.
- P5. Cohesion: NP chunk based cohesion complexity metrics such as the NPC-Cohesion metric can measure the content goodness properties of the requirements statements.
- P6. Coupling: NP chunk based coupling complexity metrics such as the NPC-Coupling metric can measure the content goodness properties of the requirements statements.

The evaluation criteria for the linking of derived data from the case study to the above three propositions was whether the linking could explain the cause-effect relationship. For simplicity reason, the cause-effect relationship was categorized into three ordinal values: strong, medium, and weak/no cause-effect relationship.

Unit(s) of Analysis: In the phase II research design, the unit of analysis was also requirements documents. Two sources of requirements documents were used for the case study: (1) four versions of the Interactive Matching and Geocoding System II (IMAGS II) requirements documents for U. S. Bureau of Census and (2) the FAA requirements document used in the phase I study.

The Logic Linking of the Data to the Propositions Criteria for Interpreting the Findings: In this section the three major categories of metrics, sentence/requirements complexity metrics, cohesion metrics, and coupling metrics, were discussed separately.

NPC-Sentence (Sentence Level Complexity Metrics): The NPC-Sentence metric was basically a way to count the NP chunks, and it could be used to identify complex requirements.

Section 3.4 of the IMAGS II requirements document was used to illustrate how the NPC-Sentence metric works. The complexity measures were first obtained from Section 3.4 of both the version 2 and version 3 of the requirements documents. The complexity measures were then compared between the two versions of the requirements. The NPC-Sentence measures of Section 3.4 of the version 2 requirements document showed that sentence 10, 11, and 12 have high degree of complexity. Subsequent iteration of requirements review indeed identified those three sentences as "difficult to understand". A new set of sentences were then developed in the version 3 of the requirements document. The comparison of the two versions of the requirements section showed that the complexity measures of the three sentences were improved in the new version of the requirements document.

NPC-Req (Requirements Level Complexity Metrics): Another section of the IMAGS II requirements document was used to illustrate the capability of the NP chunk complexity metrics at the requirements level.

The NP-Req metric values were compared between two versions of Section 3.2 of the IMAGS II requirements document: version 2 and 3. During the third iteration of the requirements gathering phase, four modifications were made, and the version 2 NPC-Sentence measures did not show clearly which sentences should be improved or re-written.

On the other hand, the version 2 NPC-Req metrics show that two requirements were the most complex requirements in the section. This coincided two of the modifications shown on the version 3 requirements document.

Based upon the above analysis, it was clear that NP chunk counts could measure the complexity of requirements statements and hence showed the strong cause-effect relationship to the content goodness properties of requirements statements. In other words, the proposition P4 was supported.

Cohesion and Coupling Metrics: The cohesion measures for the version 1 to version 4 of IMAGS II requirements documents could be illustrated by the differences between the two adjacent versions of IMAGS II requirements documents. The results were three sets of measures, and they revealed that the iteration from version 1 to version 2 and from version 2 to version 3 had substantial changes. On the other hand, the iteration from version 3 to version 4 was bounded in a relatively narrow range.

Similar to the cohesion metrics, the coupling metrics also showed that the iteration from version 3 to version 4 of the requirements stayed in a relatively narrow range.

In addition to NPC-Cohesion and NPC-Coupling, NPC-Composite was used in the study. For the IMAGS II project, NPC-Composite showed that Section 30 is the worst requirements section. After examining the requirements document, it was found that Section 30 was for reports. Reports requirements typically referenced all other sections and were independent of each other. The next low quality requirements sections were Section 3, 6, and 33. Section 3 was the requirements for the overall operations, which included multiple requirements for suspend and shutdown some operations but leaved others operational. Section 3 indeed contained complicated requirements. Section 6 discussed the address import, and Section 33 provided performance related requirements. These two sections did not seem to be complicated.

For the FAA project, the most complicated requirements section indicated by NPC-Composite was Section 13, which had the highest number of sentences and the cohesion value for Section 13 was zero. This section is indeed complicated. The next set of low quality requirements sections were Section 6, 19, and 22. Although the NP chunk count of Section 6 was not the highest, the coupling value was the highest. The problem with Section 19 and 22 was evident by their zero cohesion value.

The Phase I case study provided evidence that the proposed NPC-Cohesion and NPC-Coupling metrics were consistent with Ricker's metrics which were correlated to understandability, a content goodness property, of requirements statements. This section again reported evidence that the NPC-Composite metric had a strong cause-effect relationship to the content goodness properties of requirements statements. In other words, NPC-Cohesion supported the proposition P5, and NPC-Coupling supported the proposition P6.

Summary: Based upon the evidence discussed above, the hypothesis that the proposed complexity metrics could identify low

quality requirements statements in a requirements document was asserted.

5.5 Two Phased Case Study for Requirements Statements in a Database

We are currently conducting a two phased case study, similar to the above study, on the NP chunk metrics for requirements statements in a database. For NPC-Count, one of the three core metrics, the way to count NP chunks remains the same and is not affected by the locations of each sentence. Hence, the conclusions we have derived for NPC-Count in the previous study are still valid.

On the other hand, the previous NPC-Cohesion and NPC-Coupling utilize the concept of spatial distance and clustering, which are not applicable in the database environment. At least they are not applicable in the vector space constructed by NP chunks.

Although we calculate NPC-Cohesion and NPC-Coupling metrics differently in the two environments, documents and database, NPC-Cohesion/NPC-Coupling for documents can be used when printing requirements statements from a database. A requirements statements printout, either from a document or a database, shows requirements statements in a linear order. To maximize NPC-Cohesion and minimize NPC-Coupling for the printout should still utilize the NPC-Cohesion/NPC-Coupling for documents, not NPC-Cohesion/NPC-Coupling for databases.

6 Summary

This research made two contributions: (1) the invention of a suite of complexity metrics to measure the content goodness properties of requirements statements and (2) the empirical case study to evaluate the invented suite of complexity metrics.

The invented complexity metrics are researched and developed to identify low quality requirements statements. In the previous empirical two phased case study, it was demonstrated that the proposed metrics could measure the content goodness properties of requirements statements.

The research demonstrates the feasibility of using NP chunks as the elements of measurement for complexity metrics. In addition the invented suite of complexity metrics provides requirements engineers and managers with a tool to measure the quality of the requirements statements. These metrics can be used to identify low quality requirements. They can also be used to identify requirements and requirements sections that may require more rigorous testing. Potential flaws and risks can be reduced and dealt with earlier in the software development cycle.

At a minimum, these metrics should lay the groundwork for automated measures of requirements in documents or in databases.

7 References

- [1] S. Abney, "Parsing By Chunks". Robert Berwick and Steven Abney and Carol Tenny (eds), Principle-Based Parsing, Kluwer Academic Publishers, 1991.
- [2] S. Cant, D. R. Jeffery, and B. Henderson-Sellers, "A conceptual model of cognitive complexity of elements of the programming process", Information and Software Technology, 37(7), 351-362. 1995.
- [3] D.P. Darcy and C.F. Kemerer, Software Complexity: Toward a Unified Theory of Coupling and Cohesion, MIS Research

- Center, Carlson School of Management, University of Minnesota, February 8, 2002.
- [4] C.Y. Din, Requirements Content Goodness and Complexity Measurement Based On NP Chunks, Doctoral Dissertation, George Mason University, Fairfax, VA, 2007. Reprinted by VDM Verlag Dr. Muller, 2008. A short summary of the research was published on Journal of Systemics, Cybernetics and Informatics (JSCI), 7(3), 2009, pp.12-18.
- [5] H.E. Dunsmore, "Software Metrics: An Overview of an Evolving Methodology", Information Processing and Management, 20(1-2), 183-192, 1984.
- [6] N.A. Ernst and J. Mylopoulos, "On the perception of software quality requirements during the project lifecycle", In International Conference on Requirements Engineering: Foundation for Software Quality, pages 143-157, Essen, Germany, June 2010.
- [7] F. Fabbrini, M. Fusani, S. Gnesi, and G. Lami, "An Automatic Quality Evaluation for Natural Language Requirements", In 7th International Workshop on RE: Foundation for Software Quality REFSQ'01, 2001.
- [8] N.E. Fenton and M. Neil, "Software metrics: roadmap" Proceedings of the International Conference on Software Engineering (ICSE), 357-370, 2000.
- [9] A.C. Graesser, D.S. McNamara, M.M. Louwerse, and Z. Cai "Coh-Matrix: Analysis of Text on Cohesion and Language", Behavioral Research Methods, Instruments, and Computers, 36(2), 193-202, 2004.
- [10] B. Henderson-Sellers, Object-Oriented Metrics – Measures of Complexity, Prentice Hall PTR, New Jersey, 1996.
- [11] S. Iqbal and M.N.A. Khan, "Yet another Set of Requirement Metrics for Software Projects", In International Journal of Software Engineering and Its Applications, Vol. 6, No. 1, January, 2012.
- [12] C. Jones, Software Quality in 2008 : A Survey of the State of the Art. Technical Report, Software Quality Institute, 2008.
- [13] C.F. Kemerer, "Progress, Obstacles, and Opportunities in Software Engineering Economics", Communications of ACM, 41, 63-66, 1998.
- [14] T. Klemola, "A Cognitive Model for Complexity Metrics", 4th International ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering, June 13, 2000.
- [15] S. Lee, Proxy Viewpoints Model-Based Requirements Discovery, Doctoral Dissertation, George Mason University, 2003.
- [16] A.V. Levitin, "How to Measure Size, and How Not to", Proc. Tenth COMPSAC 1986, Chicago, Oct 8-10, 1986, IEEE Computer Society Press, Washington DC, 314-318, 1986.
- [17] D.S. McNamara, "Reading both high and low coherence texts: Effects of text sequence and prior knowledge", Canadian Journal of Experimental Psychology, 55, 51-62, 2001.
- [18] D.S. McNamara, E. Kintsch, N.B. Songer, and W. Kintsch, "Are good texts always better? Text coherence, background knowledge, and levels of understanding in learning from text", Cognition and Instruction, 14, 1-43, 1996.
- [19] M.M. Mora and C. Denger, "Requirement Metrics: An initial literature survey on measurement approaches for requirement specifications", IESE-Report No. 096.03/ Version 1.0, October 1, 2003.
- [20] J. Natt och Dag, B. Regnell, V. Gervasi, and S. Brinkkemper, "A linguistic-engineering approach to large-scale requirements management", Software, IEEE, 22 (1), pg. 32-39, January 2005.
- [21] S. Purao and V. Vaishnavi, "Product Metrics for Object-Oriented Systems", ACM Computing Surveys, 35(2), 191-221, 2003.
- [22] D. Regnell, R.B. Svensson, and K. Wnuk, "Can We Beat the Complexity of Very Large-Scale Requirements Engineering?", In International Conference on Requirements Engineering: Foundation for Software Quality, Vol. 5025, Lecture Notes in Computer Science, pages 123-128, Montpellier, France, 2008
- [23] D.J. Reifer, "Profiles of Level 5 CMMI Organizations", Crosstalk: The Journal of Defense Software Engineering, January, 2007.
- [24] M. Ricker, Requirements Specification Understandability Evaluation with Cohesion, Context, and Coupling, Doctoral Dissertation, George Mason University, Fairfax, VA, 1995.
- [25] L.H. Rosenberg, T.F. Hammer, and L.L. Huffman, "Requirements, Testing, and Metrics", In 15th Annual Pacific Northwest Software Quality Conference, 1998.
- [26] R.E. Schneider, Process for building a more effective set of requirement goodness properties, Doctoral Dissertation, George Mason University, Fairfax, VA, 2002.
- [27] C. Schwaber, "The Root of the Problem: Poor Requirements", IT View Research Document, Forrester Research, September, 2006.
- [28] I. Sommerville, Software Engineering: update, 8th Edition, International Computer Science, Addison Wesley, 2006.
- [29] W.M. Wilson, L.H. Rosenberg, and L.E. Hyatt, "Automated Quality Analysis of Natural Language Requirement Specifications", Fourteenth Annual Pacific Northwest Software Quality Conference, October, 1996.
- [30] R.K. Yin, Case study research, 3rd edition. Thousand Oaks, CA: Sage Publications, 2003.

Proposal and Development of Markers-type Mouse System with Considering Practical and Entertainment

Kazuya Murata, Takayuki Fujimoto
Graduate School of Engineering, Toyo University
Kujirai2100, Kawagoe-City, Saitama, Japan
gz1100108@toyo.jp, me@fujiotokyo.com

Abstract - *We proposed the new mouse interface that does not need operation restrictions of the existing mouse, and developed the prototype system. Our mouse system was developed using Augmented Reality Technology. This system attaches a mouse function to the marker attached to the object, makes a computer recognize it, and operates. All the things that attached the marker with a mouse function by this system can be used as a mouse interface. As a result, we realized the mouse interface system that does not have restrictions in mouse operation.*

Keywords: Marker, Augmented Reality technology, Mouse interface, Mouse function

1 Introduction

Currently, the computer has spread to a home as a tool useful for anyone. In the background, there is development of the user interface for computer operation. Moreover, a mouse interface is famous also in a user interface. As a large factor, there is existence of the mouse interface developed by Douglas Engelbart in 1961.

After a mouse interface is invented, various mouse interfaces in consideration of convenience, a design, etc. have been developed. A mainstream mouse interface has three functions, "Left-click", "Right-click" and "Cursor movement". Currently, a mouse interface with these three functions is in use. Moreover, the mouse of an elliptical form called "two Button type" with a minimum function for controlling a computer is general. The design and structure of the "two Button type" are being fixed as a "form" of today's mouse interface.

However, the environment and the purpose of a computer are also diversified in recent years. Therefore, as for the interface for computer operation, diversification is demanded. Still now, various mouse interfaces are proposed and developed.

In this paper, we propose the new mouse system with a different concept by using the Augmented Reality Technology, and develop a prototype system.

In the prototype system developed by this study, the "marker" to which the function of the mouse was attached is used. By using this marker, even if there is no existing mouse, the function of a mouse is realizable. Moreover, if it is a thing that can attach a marker, it is utilizable as a mouse. As a result, a user's favorite thing can also be utilized as a mouse.

2 Purpose of Study

A mouse in recent years is divided into two classifications. First, it is a mouse operated using rotation of a trackball. Second, it is a mouse that makes the bottom of a mouse possess LED, is made to reflect LED in a plane floor etc., and is operated.

Although the mouse that LED possesses on the bottom in recent years is in use, the operation method is mentioned as the reason. When cursor operation is considered, with the mouse of a trackball, a ball is rotated in person. Therefore, precise operation cannot be performed if the hand of cut and moderate rotation is not taken into consideration.

When it is the mouse that LED possesses on the bottom to it, the same motion as a direction to move on a flat place is performed. By that, there is an advantage that can perform cursor movement easily satisfactorily. However, when a flat place does not exist, there is a danger that operation is impossible. Moreover, while the use environment of a computer is diversified, the form of the interface used for the operation is also asked for diversity. But, a sufficient proposal or development is not necessarily made.

So, we aim at the interface that does not need restriction of the existing mouse interface. In order to realize this aim, we used the marker of Augmented Reality Technology. We propose the new mouse interface that uses the marker of extended actual feeling technology and makes computer operation possible. In addition, based on the proposed interface, and we develop a prototype system.

3 Background of Study

The mainstream of the present computer operation is a mouse interface. However, the present mouse interface requires a flat place. Moreover, in the track pad type of a notebook computer, operation area is narrow, and it is not fit for efficient operation. Furthermore, use becomes difficult when there is no place for placing a trackball type mouse interface. Currently, there are restrictions that a flat place is required for the mouse interface proposed and offered. Therefore, although restrictions of a mouse interface are not needed, the new mouse interface with a basic function can predict diversification. Then, we thought that there was big interest in a new mouse interface.

4 Summary of System

In this system, not using the existing mouse interface, the new mouse interface that conquers the fault of a mouse interface is realized. As the method, we propose the system that sticks a "marker" on a familiar thing and as which it operates it as a mouse interface. From this, it becomes possible to use mouse function even if there is no existing mouse interface. Moreover, the new mouse interface that does not need the flat place that is a fault of the existing mouse interface is realized.

The outline of this system is described below. In this system, the function of a mouse interface is given to a marker, and even if there is no existing mouse interface, the function of a mouse interface is realized. Augmented Reality Technology is used for giving the function of a mouse interface to a marker. Augmented Reality Technology recognizes the marker with the Web camera attached to a computer. Therefore, if it is a computer with a camera function, it can use also by a general computer. Augmented Reality Technology can display three-dimensional graphics, if a marker can be recognized with a camera. The marker of Augmented Reality Technology has two processing. It is "when recognize" and "when cannot recognize". This system the function of a mouse interface is realized using this processing.

In addition, when the functions of a common mouse interface is considered, there are three functions, "Left-click", "Right-click" and "Cursor movement". The functions realized by this system are these three functions.

4.1 Marker Attached Mouse Functions

This system realizes a mouse function with the marker of Augmented Reality Technology. It is operated by processing "when recognize" and "when cannot recognize". In addition, when considering the minimum function of a mouse, there are six functions "Left-click", "Right-click", "Movement on cursor", "Movement under cursor", "Movement left cursor" and "Movement right cursor". Therefore, the six markers for them are needed.

Figure 1 show the markers to use in this paper. An upper two markers are marker for a click. The marker of an arrow is a marker for cursor movement. The markers in Figure 1 are one example. A design can be changed into a free design if recognition conditions are fulfilled by Augmented Reality Technology.

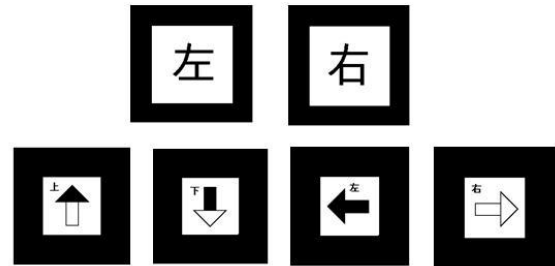


Figure 1. Example of the marker used for the prototype system.

5 How to Operate the Marker

This study developed a prototype system using the proposed method. In this chapter, the usage of the prototype system is described.

5.1 Mouse Click Operation

A case of click operation, "when recognize" has nothing happen. And, "When cannot recognize" click operation is performed.

As an example, the procedure of the click operation is shown below.

- (1) The click marker can be recognized from the camera. (Click button leave)
- (2) The click marker cannot be recognized by the finger etc. (Click button push)
- (3) The click marker can be recognized from camera again. (Click button leave)
- (4) The click function operates.

The example of click operation is shown in Figure 2.



Figure 2. Example of click operation

5.2 Mouse Cursor Movement Operation

Four markers of the lower berth of Figure 1 were used for cursor movement.

A case of cursor movement operation, “When cannot recognize” has nothing happen. And, “When cannot recognize” cursor movement is performed. The operation procedure of the cursor movement is shown below.

- (1) The cursor movement marker cannot be recognized from the camera. (Cursor move stop)
- (2) The cursor movement marker can be recognized. (Cursor move start)
- (3) The cursor movement marker cannot be recognized from the camera again. (Cursor move stop)

At this time, if a marker can be recognized, it can attach anywhere. For example, it is a PET bottle, a writing case, and a purse etc. If a marker can attach, anything can realize the function of a mouse. Figure 3 shows one example of “Extempore mouse interface” used to experiment. These were able to use a minimum function of the mouse.

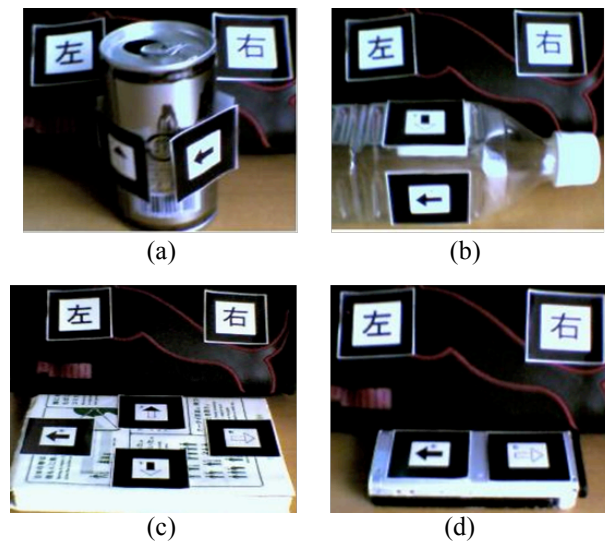


Figure 3. Example of “Extempore mouse interface” (a)empty can, (b)Plastic bottle, (c)book, (d)mobile phone

6 Example of System Operation Procedures

In this system, by attaching the marker in which the function of the mouse is incorporated, anything can be used as a mouse. As the example of operation of this prototype system, “Unnecessary box” was used. Of course, even if this is not “Unnecessary box”, if a marker can be attached, it can use anything.

The example of operation is described below. As an example, operation of starting “Internet Explorer” on a

desktop is performed. Usually, in order to start a browser, it carries out at the following three steps.

- (1) A cursor is moved to on an icon.
- (2) Double-click the left click of mouse.
- (3) The browser start.

Therefore, this system also needs to perform same operation. A cursor movement marker is attached on an “Unnecessary box” and a mouse click marker is arranged. In this stage, “Unnecessary box” serves as an interface with a mouse function.

First, the “Unnecessary box” that stuck the marker so that it might be visible from a camera is arranged. Next, a cursor movement marker is made to recognize and a cursor is moved. The actually used box is shown in Figure 4, and a system startup state is shown in Figure 5. In addition, the three-dimensional graphics currently displayed on Figure 5 is graphics of a fundamental quadrangle. These graphics are made intelligible in an experiment. Of course, the graphics in consideration of entertainment, etc. can be changed freely.



Figure 4. Example which attached the marker of the mouse function on the “Unnecessary box”

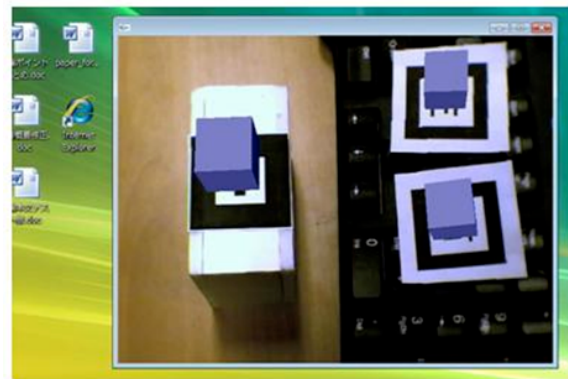


Figure 5. Example of starting of the whole system

Next, if movement of a cursor is completed, it prevents from recognizing a mouse click marker with a finger etc. Then, the click Operated and "Internet Explorer" starts.

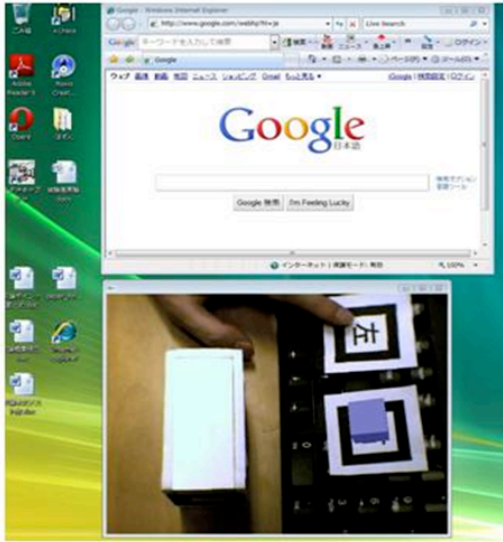


Figure 6. Starting form the icon by click operation

7 Conclusion and Consideration

In this paper, we proposed the mouse interface according to the new concept. In addition, we proposed a new mouse interface using Augmented Reality Technology, and development a prototype system.

This prototype sytem, mouse functions can be realized without the need for operational constraints of the existing mouse interface. Moreover, this system is possible to display three-dimensional graphics by using Augmented Reality technology. That is, application in consideration of entertainment is possible. From this, the value of entertainment that was not in an old mouse interface can be added.

This system dose not needs the operation restrictions of existing mouse interface. But, there is a problem that cannot be used without the camera for recognizing a marker. However, the recent notebook computer has a small built-in web camera. From this, we think that it is not a difficult problem. Therefore, this system for general computer user dose not requires a special environment and constraints. In addition, the existing mouse interface and operation constraints are not needed. From the above, we think that it can contribute for general-purpose use, choice of a mouse interface and extension of diversity.

Especially, the marker type mouse system in particular proposed by this study is not the new mouse interface mounted by a simply different concept. Rather, we think that the deployment possessing entertainment of a new mouse system and the possibility of application of welfare apparatus etc. could also be suggested.

8 Problem Point

8.1 Problem of Cursor Movement operation

This In this system, in realizing the function of mouse interface by a marker, six minimum functions were mounted "Left Click", "Right Click", "Up Move", "Down Move", "Left Move", and "Right Move". As a result, the cursor movement has been fixed vertically and horizontally. That is the problem that nonlinear movement cannot be performed has occurred.

8.2 Problem of Click operation

In The problem that should improve also about click operation remained. The marker designed the system that performs click operation, "cannot be recognized". As a result, if recognition of a marker becomes impossible in some accidents, an error may occur. In addition, when the state where a marker cannot be recognized continues, there is a problem that malfunction of click operation generates. We would like to consider it as a future examination subject.

9 Relevant study and Similar Study

In this study, there is ARDesktop as related study. ARDesktop is the three-dimensional graphics library that applied ARtoolkit and was manufactured. If the graphics of a marker are touched in the graphics currently displayed, operation corresponding to it will be performed. Moreover, it is possible to operate it so that using two or more markers may hold graphics.

There is a mouse interface of OZUPAD as similar study. Incorporated Company Blue Mouse Technology developed OZUPAD. OZUPAD is a multifunction mouse interface that can operate not only as the function of a mouse interface bus as a joystick and the presentation function also has. Moreover, it is wireless, and it can be operate even if there is no flat place like this study.

In this study, the proposal and the prototype were performed based on the concept of "realizing the function of a mouse interface using a marker not using the existing mouse interface." However such a mouse system exists in neither the existing study nor similar study, so it is thought that novelty is high.

10 References

[1] T. Tanijiri, "Augmented Reality Technology Programing Technique to Achieve Augmented Reality," CUTT System, Ltd. 2008

[2] T. Tsuchiya and K. Takahashi, "Development of AR system Using Hand Gesture," The science and engineering review of Doshisha University 50(3), [107]-113, 10-2009, Doshisha University.

[3] Y. Teramae, Y. Ban and K. Uehara, "The Development of MR-based Desktop Collaboration System," IEICE technical report, PRMU, pattern recognition and media understanding 104(572), 1-6, 13-01-2005.

[4] T. Wada, M. Takahashi, K. Kagawa and J. ohta, "Method to Realize Mouse functions Using Laser Pointer," journal of the institute of Image Information and Television Engineers 63(5), 657-664, 01-05-2009

[5] T. Kurata, T. Okuma, M. Kouroggi and K. Sakaue, "The Hand-mouse : A Human Interface Suitable for Augmented Reality Environments Enabled by Visual Wearables," IEICE technical report, PRMU 100(565), 69-76, 11-01-2001

[6] GIGAZINE:
http://gigazine.net/news/20081204_mouse_over_the_years/

ERADICATING COMPLEXITY IN SOFTWARE INTERFACE FOR INCREASED PRODUCTIVITY

Increasing Effectiveness of Enterprise Systems

Felix Bollou, Edmund Balogun, and Inah Usang
School of Information Technology and Communication (SITC),
American University of Nigeria, Yola Adamawa State, Nigeria

Abstract -Enterprise Resource Planning Systems (ERPs) are usually complex in terms of conception, design and in terms of usability for the average users. This situation was tolerated for a long time because the majority of users where either computer technical personnel or the few privileged educated elite with good computer literacy. The complexity of ERPs interface is legendary and no matter how long ERP systems have been used, the users have difficulties to intuitively run the transactions. In order to incorporate the new requirements, efforts by programmers to create more intuitive and interactive systems have been somewhat successful yet there still remain the tendency to introduce complexity and too much sophistication in interface design. This paper gives an insights into some paradigms associated with interface designs and also provides contextual case studies.

Keywords: ERP, Interface design, Human-Computer Interaction, Complexity in Interface Design

1 CHAPTER ONE

1.1 Introduction

In the world of computing today, the importance of a grand interface design cannot be overemphasized. Before the advent of personal computers in the late 1970s, the main people who often used and interacted with computers were software developers, computer scientists and information technology professionals. Computing has evolved overtime and the existence of personal computers has created an avenue for humans to interact with computers. Personal computers have played a major role in the development of mankind, as they have been applied to daily lives and activities. Computer applications have been developed for

virtually all sectors, ranging from medicine, banking to education. The most important factor for the proper use of a computer program is the interface; however in recent times, some computer programs have become highly intricate and uneasy to manipulate because of the complexity of their interfaces.

The use of computers in today's world has become a widespread phenomenon. Computers have basically been developed to aid man in certain endeavors. As computers are increasingly becoming service oriented, the need to design and implement a user-friendly interface is on the rise. Systems ranging from Automated Teller Machines (ATMs), Cell Phones, and Personal Computers (PCs) to Digital blood pressure apparatus are all engineered to be service oriented. The way humans interact with a particular system is important and as such the major goal of system developers and designers should be to provide a tremendous user interface based on the functionality and the intuitiveness of the system. Nowadays most companies strive to employ Ubiquitous Information Systems (UIS) to aid their operations. UIS offers a medium for organizations to provide users and workgroups with services of ubiquitous computing technologies in real-time. "UIS come with more complex requirements than the more strongly constrained Information Systems for office settings" (Maass, 2012), and as such introduces some complexity in its structure. Similarly some systems introduce complexity by trying to meet the functionality of the system and often disregard the concerns of the user.

Human-Computer Interaction deals with certain paradigms of how humans relate and respond to computers. Consequently, this paper would offer an insight into such paradigms and examine from a contextual view, the importance of interface designs especially in service oriented systems. Considering a scenario where a user stands in front of an ATM machine for several minutes, trying to manipulate the ATM and is unable to perform a

smooth operation. This dawdling process can be attributed to so many factors, one of which is intricate interface design. In the course of this research, similar situations would be pointed out and properly examined to determine how complexities can affect productivity, especially in organizational environments. Consequently, it is deceptive to see that an SAP expert will require additional training before becoming proficient on Oracle business suite. Today, computing has become ubiquitous; even people that apparently don't have any business with computers are exposed to information to be processed on daily basis. Therefore, the way to present information must be adapted to the new, diverse and broader audience. To gain an understanding of how users' can be affected by an interface, some users' feedback were studied on the Banner higher education system at the American University of Nigeria. Some findings were that users were reluctant to the user interface because it is too technical and lacks ease of use. Further, we suggest that ERP systems introduce some standards to ease technician's move from one to the other.

1.2 The problem of Interface Complexity.

The problem of complex interface sometimes affects users performances and leads to low productivity especially in organizations that use extensive Information Systems and self-service systems for their business transactions. Management Information Systems as a field that deals with the interaction of computers, procedures, and people in an organization should clearly play a central role in deciding some of the design principles and techniques to be adopted in programs used by organizations. Most Organizations spend huge sums of money in correcting errors in their processes. These errors can be attributed to many factors: These factors include time delays in performing a task, inconsistency, incomplete information and inaccurate data. Most of the time, these problems are caused by un-easy user Interfaces.

Human-Computer Interaction addresses issues that deal with the design methodologies of user-interface and guidelines for proper interface designs. HCI in Information Systems is an imperative stream that tends to merge both fields to solve common problems associated in both areas. Further research have sprung up under this topic, but this present work focuses on identifying and proposing solutions to certain identified complexities in systems encountered by users, particularly users in organizations. Several techniques, guidelines, and design

solutions are highlighted in this research that can be adopted by developers to further enhance interface design.

2 CHAPTER TWO

2.1 Literature review

Attempts have been made by several researchers to define Human-Computer Interaction (HCI). Human-Computer Interaction is an area of research that began in the 1980s as a branch in computer science (Carroll, 2009). HCI is viewed as a discipline concerned with the design, evaluation and implementation of interactive computing systems for human use (Thomas , et al., 1996). HCI is a multidisciplinary study that deals with both art and science; it illustrates the interdependence of a software system and its interface. (Nanni, 2004). According to Tufte (1989) HCI can be seen as "two powerful information processors, i.e. the human and the computer attempting to communicate with each other via a narrow bandwidth, highly constrained interface" (Tufti, 1989). It is imperative that we have an understanding of what has been said about Human-computer Interaction and how it relates to interface design. Usability in interface design is of importance because it is the degree to which an "interface takes into account the human psychology and physiology of its users" (Wachowiak, Wachowiak--Smolikova, & Friya, 2010). Therefore we can say that HCI deals with the methodologies of how computers and humans interact and how interdependent they are. HCI, in consanguinity to interface design is vital, and it has been estimated that about 48% of works on system development goes to the design and implementation of the user interface (Myers & Rosson , 1992).

When designing user interface for systems, the communication between users and computers must be taken into consideration because the user's attributes do not often match computer features. (Fetaji, Suzana, Bekim, & Mirlinda, 2007). The significance of interface designs and issues in Human-Computer Interaction has not risen to a high level in information systems; its importance has long gone unrecognized (Peslak, 2005). Zhang, et al. highlights some opinions on the significance of HCI considerations in business applications and how essential it is in the development of a system. They further suggest that some information systems failures can be credited to "faulty design choices resulting from the lack of emphasis on the human/social aspects of system use" (Zhang, Jane, Dov, & Marilyn, 2004).

Complexities are sometimes introduced in interface designs, and as such reduce users' ability to utilize full functionality of the system. Suggestions have been made for developers and students in design areas to develop competency through guided learning in understanding and devising user-friendly systems and solutions (Peslak, 2005). Consequently (Quaye, 1990), further conducted an investigation of HCI and how the quality of an interface affects users both on personal and enterprise levels. Comprehensive works from various researchers have been done on HCI and have showcased different topics and raised many concerns as well as contributions. According to (Danino, 2001) certain questions arise when conferring about HCI and how it relates to interface design. Questions like; why do some people become so good in navigating new systems effortlessly while others scuffle to learn? Why are some websites easier to navigate than others? Why do some users encounter difficulties in operating electrical devices?

The relevance of this work would be to infer ways in which these problems can be alleviated. This project also offers insights into some minor interface problems often neglected by developers which sometimes cause time wastage, reduces performance and usability both on small and large scale basis.

2.2 What is Human-Computer Interaction?

Early computers were mostly used by scientists for research and by business organizations for several analytical and production functions. Today, the number of computer user's increases due to the availability of PCs. The increase in computer users triggered the need for good interface designs to enhance the interaction between humans and computers, which sprung a research stream known as Human-Computer Interaction (HCI). The study of how people make use of various mediums of computer technology tools to perform their daily activities is what we define as Human-Computer Interaction. Consider user-A using an ATM machine to perform a transaction or user-B using a cell phone to make a phone call. The interaction between the user-A and the ATM's interface or user-B and the cell phone is a basic idea of the concept of Human-Computer Interaction, though it can be viewed with more complex examples. The study of HCI integrates different disciplines like computer science, engineering, graphic design, psychology, philosophy and ergonometic.

The interaction between humans and computer systems is made possible through a medium called the User Interface (UI). The user interface acts like a middleman

between computer system and humans. The user interface gives the users a first impression of a system therefore if the interface is not properly designed, users may not make optimal use of the system. To enhance interactivity, the study of HCI is recommended for software developers for the production of user-friendly interfaces. Most software developers focus on the functionality of the system disregarding the fact that the users are also stakeholders of the software. Functionality of a system is the tasks or action that the system can perform. One major key to designing a good interface is by understanding the need of the potential user. To enhance user interactivity with computer systems, there should be a balance between the functionality and the usability of the system. Usability on the other hand is the degree in which the various functions of a system can be used to achieve the goal of the user (Fakhreddine, Milad, Jamil, & Mo, 2008).

HCI can also be linked with the integration of Computer Science and Cognitive Science (study of thought, learning and mental organization). To better understand how humans relate with computer systems, it is best to understand how mental activities prompt certain behavior. For this reason, cognitive science plays a vital role in the study of HCI. The study of cognitive science helps designers develop more intuitive systems. It also helps interface designers understand their user which makes it easy for interface designers to know what exactly the users want.

2.3 A Brief History of HCI

As stated earlier, before the 1970s, Information Technology Professionals mostly carried out computing and all this changed with the introduction of Personal computers (PC) in the early 1980s. PCs enhanced individual computing greatly by making software such as spreadsheet, gaming applications and computing platforms (Operating Systems) accessible to ordinary people. The existence of Personal Computers basically led to the extinction of machines like typewriters, adding machines and dedicated word processors because PCs had large processing capability and ease of use. Although the PCs made it possible for users to do more, a major setback was the complex and perplexing interface (Carroll, 2009). As PC users increased, there was a need for software developers to understand user requirement and produce intuitive and interactive systems.

Cognitive Science is an area related to HCI that has also been under research since the late 1970s. It is the

study of thought, learning and mental organizations. Scientist from different disciplines like Artificial Intelligence, Cognitive Psychology, Philosophy of the mind, Linguistics and Cognitive Anthropology converged to form the field of Cognitive Science. Some cognitive scientist made use of computer systems for their research leading to their understanding of how computers could be used by people to solve problems (Mary & Carroll M., 2002). The merging of Cognitive Science and computer science brought about HCI, which was the first branch of Cognitive Engineering. Most users judge systems based on the interface and not the internal components of the system

Also in the late 1970s, the Xerox Park Research Project was in progress though the scientists had no clear idea what HCI was exactly. The major goal of this research was to make computer software and systems more interactive for users to enhance productivity of organizations. This research led to the redesign of the computer mouse technology, which was invented earlier by Douglas Engelbart in 1953 (Vochin, 2009). The use of desktop icons was another important area of the research, a starting point of the more advanced user interface we make use of today (IconLogic, 2006).

The study of HCI grew as it influenced most branches of Information Technology. Software engineers in the 1970s developed complex and confusing systems because they emphasized more on developing systems and did not really consider user interactivity. Therefore the need for more interactive and intuitive systems cannot be overemphasized because it plays a vital role for subsequent generations of computing.

Another major impact of HCI is the GOMS (goals, operators, methods and selection rules) project, which was setup in 1983 by Stuart Card, Thomas P. Moran and Allen Newell. This project provided facts on the usability of a system through the study of the human mental activities. The GOMS model operates by collecting user specifications and description of task that the user can perform with the system and makes predictions based on the data collected. The GOMS model can make predictions of the time it would take a user to perform a task or learn to use a system. This often helps to save resources because data would not need to be empirically collected during design (Mary & Carroll M., 2002).

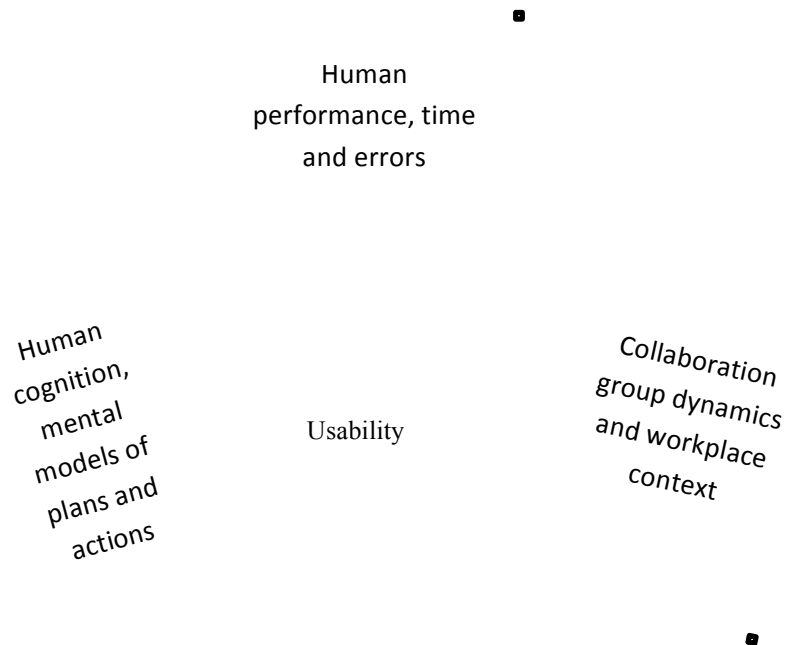


Figure 2.1 illustrates the three perspectives of usability engineering.

Source: Usability Engineering: Scenario Based Development of Human-Computer Interaction¹

3 CHAPTER THREE

3.1 Survey on Banner User Interface (Case Study)

The American University of Nigeria uses Banner for managing its student data and other processes. The Banner software is an administrative software program that was developed specially for managing data in higher institutions. A survey was conducted on four main administrative departmental users of banner at AUN. The departments are

- Finance
- Registrar
- Academic
- Housing/admission

The departments listed above are greatly affected by the performance of banner because they use it on a daily basis to compile information of both students and staffs.

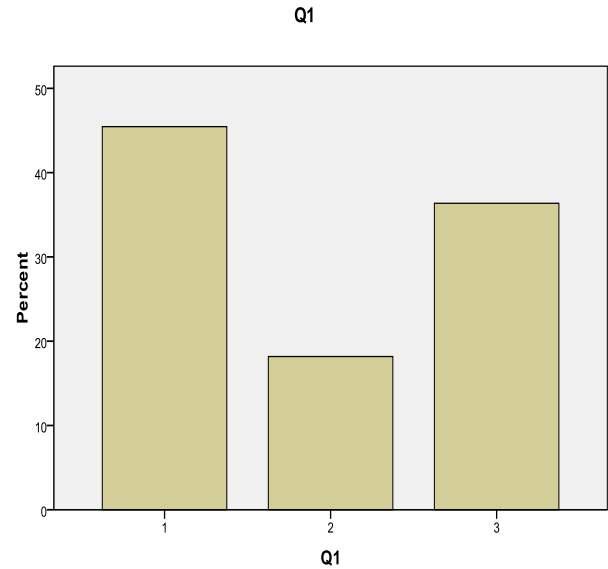
The survey was limited to the interface of banner and did not analyse or examine the effectiveness or functionality of

(Mark & Scott)¹

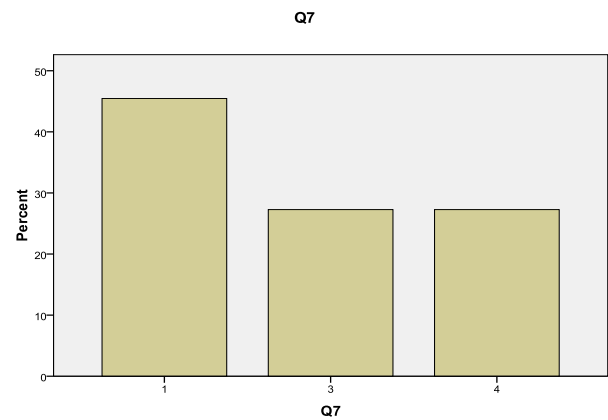
banner operations. Questionnaires were developed and shared to staffs in the four departments listed above. The questions asked are listed below:

KEY	QUESTIONS
Q 1	Does an Interface affect your Performance?
Q2	Do you think Banner is more difficult than the traditional spreadsheets like excel?
Q3	How long have you been using banner?
Q4	Are You Satisfied with the Banner Interface?
Q5	Has Banner improved your performance?
Q6	Are you conversant with banner operations?
Q7	Do you perform your work better with banner?
Q8	Do you think Banner system is highly interactive?
Q9	Are you 100% comfortable using banner?
Q10	Is the interface user friendly?
Q11	Do you think banner is some worth complex and uneasy to use?
Q12	Do you perform your work faster with banner?
Q13	Do you think banner wastes your time and does not really provide a learning environment?
Q14	What are some of the challenges you face when using banner, please specify.
Q15	If you have a better and more interactive interface would you use it instead of banner?

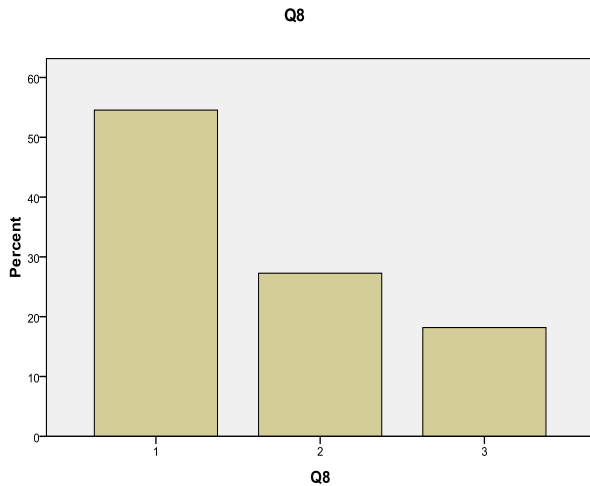
After collecting all the questionnaires, SPSS (Statistical Package for the Social Sciences) was used to analyse the results. The questions were represented with codes on SPSS, to enable a smooth computation of the data. The graphical representation of the results for each question is shown below.



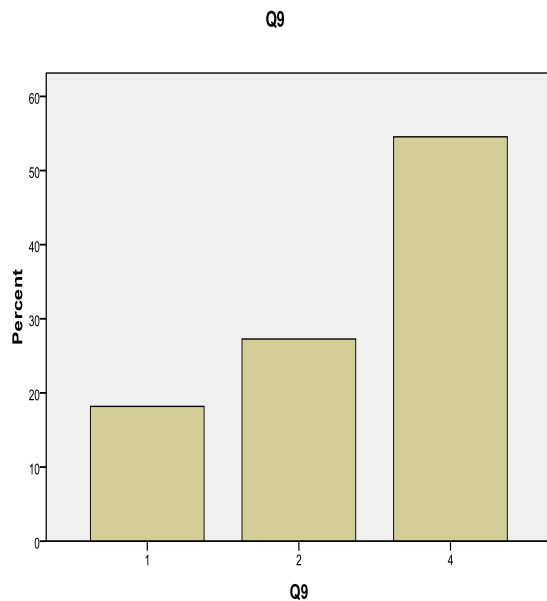
Question 1 showed that about 45% of the users admitted that an interface affected their overall performance, 35% were uncertain if an interface affected their performance and 20% of the respondents said an interface did not affect their performance.



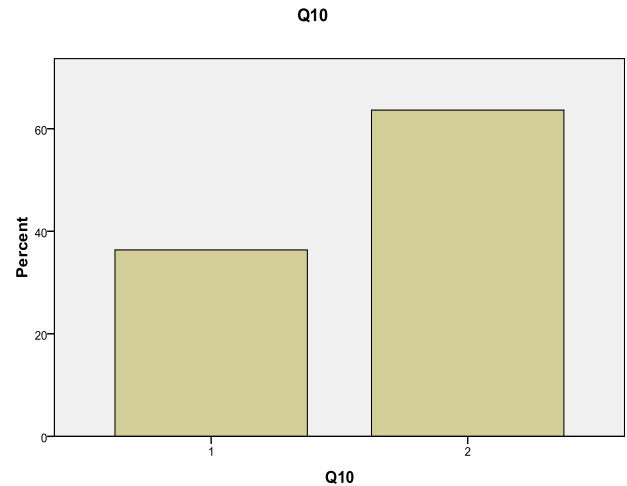
Quite similar to question 6, **Question 7** asked users if they performed their work faster with Banner and about 50% of the respondents were affirmative that they performed their job faster with Banner, while 25% each said they performed faster with banner sometimes and to some extent.



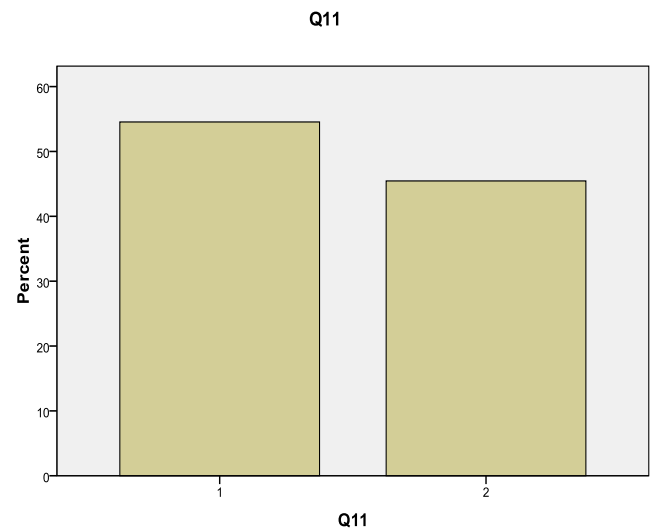
As noted earlier, users in the four departments have not been exposed to any software for performing tasks like banner; hence, in **Question 8**, 55% of users from all departments felt Banner was highly interactive, while 25% said it was not interactive, the remaining 20% said it was interactive to some extent.



On **Question 9** respondents were asked if they were 100% comfortable using Banner to perform daily activities and did not have any issues with it, 18% of respondents were not completely satisfied with it, 28% were not satisfied at all and the remaining 55% were satisfied.



Question 10 was the most relevant for this research; respondents were asked if they thought the banner user interface was user friendly, and 65% said it was not user friendly, while 35% said it was.



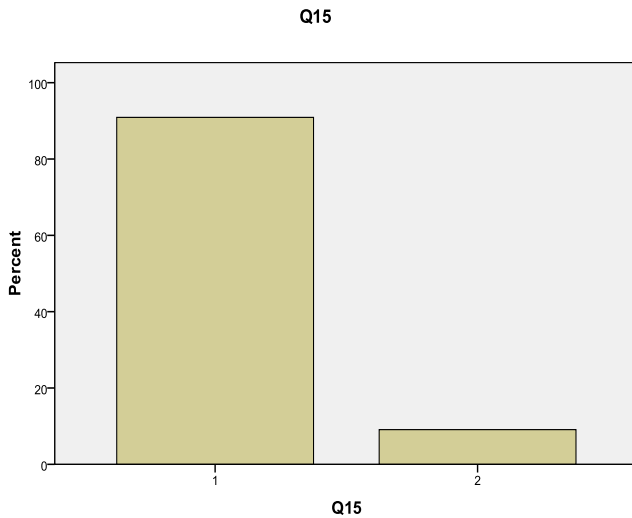
Question 11 was also very relevant for the research because 55% of the users in all departments said Banner was complex and uneasy to use.

Question 12 basically asked users if they worked faster with banner and it was similar to Question 7 and recorded the same results.

Question 13 was similar to the results of **Question 11**, because 55% of the respondents felt that banner wasted their time basically because of incomplete information and time wastage. The remaining 45% said it did not really waste their time.

On **Question 14**, users were asked to write down some of the challenges they faced with Banner, and the following conclusion was drawn from their responses.

1. Respondents felt Banner was technical
2. They said it was some worth complex
3. They said the interface was not user friendly



Question 15 was the last and Users were asked if they had a more user- friendly software, if they would stop using banner and move. 90% of respondents answered affirmative to this question.

3.2 Limitations of Survey

The survey on the Banner system was basically to determine users' opinion on the software to determine how the interface affected their performance and productivity and what they thought about the software in general. The survey did not in any way analyse the functionality of Banner, nor did it evaluate its performance and operations. It only strived to get an insight on the perspective of its users.

3.3 Comparison of SAP and OPEN ERP

To further illustrate examples of bad interface, we compared the interface of OPEN ERP and SAP, two important Enterprise Resource planning (ERP) applications widely used in Companies around the world. Closely examining both applications, SAP interface was not as interactive as OPEN ERP. SAP stands for "Systems, Applications, and Products in Data Processing". It is one of the world's leading enterprise application software. SAP basically integrates all functional areas of a business and

makes the business run faster. SAP has been very effective in maintaining businesses around the world. Despite its success and effectiveness, SAP faces some design challenges as compared to other Enterprise Resource Planning applications like OPEN ERP.

Users of both applications can immediately tell the difference between them. While OPEN ERP is easier to navigate and much more intuitive, SAP is some worth complex and requires experience. For instance if a user wants to create a sales order on SAP, the user has to go through series of steps before he/she can create the order. On the other hand, to create a sales order using OPEN ERP, the process is much more interactive and easy to create. The figure below shows SAP interface. As shown in the figure, to create a sales order the user has to understand each button and click five steps to find the create button before a sales order can be created.

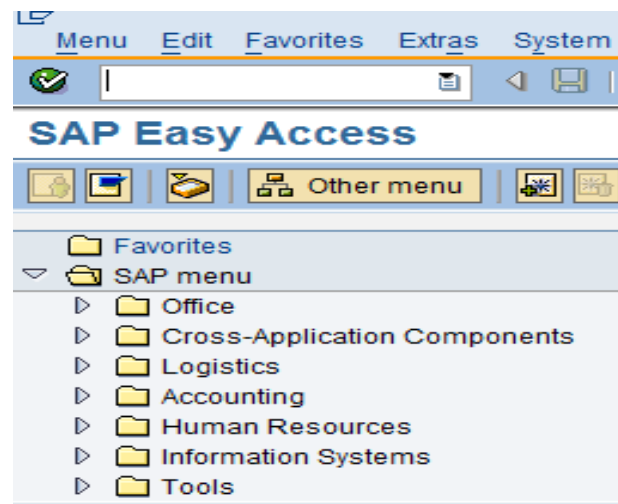


Figure 3.1 SAP Interface

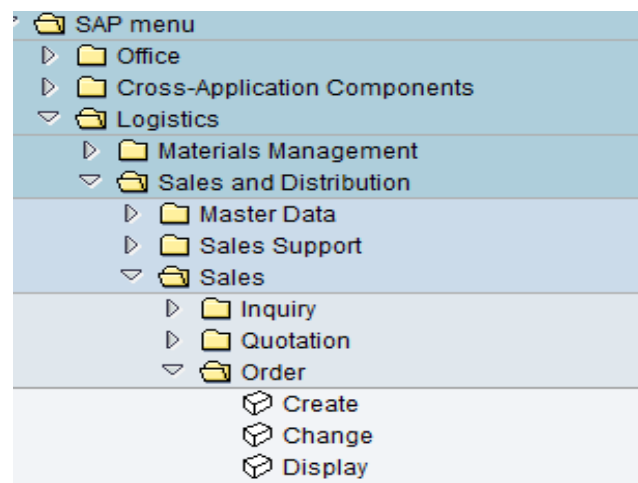


Figure 3.2 SAP create order screen

Alternatively, using OPEN ERP to perform a similar operation to create the sales order, the user is seamlessly able to navigate the application without going through the dawdling process associated with SAP. Once the user logs in, icons are displayed that represent each function the user can perform. To perform a sales order, the user only needs to click on the sales icon. The figures below show OPEN ERP screen and



Figure 3.3 OPEN ERP Home Screen Interface

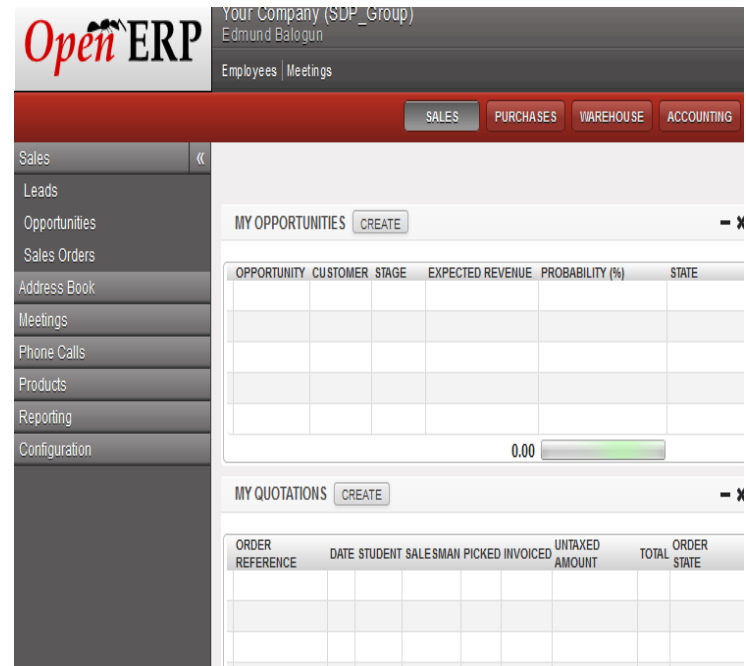


Figure 3.4 OPEN ERP create order screen

4 CHAPTER FOUR

4.1 What is Complexity?

Scholars from different disciplines have defined complexity. Before providing a definition of complexity that relates to the present research, it is important that we examine the definitions of other researchers on complex systems. “A complex system is a collection of elements, parts or agent that interacts and are interwoven defined by the structure of the system, the interaction between parts and the dynamics and patterns of the system that emerge from these interactions” (C.A. Manduca and D.W. Mogk, 2006). A complex system can also be “a system with numerous components and interconnections, interaction or interdependence that are difficult to describe, understand, predict, manage, design or change” (Magee Christopher and de weck Oliver, 2004)

Having looked at these definitions of complexity, we therefore define complexity of a system as an interaction between components or parts of a system that is difficult to comprehend which results in inefficiency for most users.

4.2 Sources of Complexity in Interface Design

Designing an interface requires skill and patience of the designer; we identified three major sources of complexity. Implementation complexity, Interface complexity and

Codebase complexity are sources of complexity discussed below (Raymond, 2003);

Implementation Complexity: This is the challenge a designer or programmer faces while trying to understand a program and debug it. If a developer does not completely understand the program, the programmer tends to design systems that are harder to debug, maintain and most especially, uneasy to use (Raymond, 2003).

Interface Complexity: This complexity is mostly experienced by regular users of a system. A bad UI causes users to make errors and spend much time performing a simple task. When users have to remember so many commands, concepts and gesture usability is reduced causing a decline in productivity in organizations and customer satisfaction.

Codebase Size: This is the effect of the sources mentioned above. It is basically the number of lines of code present in a given system. Developers often try to reduce the lines of code for systems and in some cases omit some important features of the system from the interface.

Having examined the three major causes of complexity, it is important to also outline types of complexity:

Essential Complexity: In this case, it is almost impossible to carry out simplicity in the design process because the features are complex and need to be added to the design for full functionality. A typical example is a jetliner. They are often too many equipment, channels, interface and subsystems that are controlled by one person; this complexity is in most cases unavoidable.

Accidental Complexity: the designer usually causes this complexity. It occurs when the programmer doesn't find an easy way to implement certain features for an interface. This complexity can be reduced by good designing or redesign.

Optional Complexity: This occurs when certain features are desired in the project objectives. The only way to get rid of this complexity is by changing the project objective.

The diagram below best describes the relationship between sources and kinds of complexity:

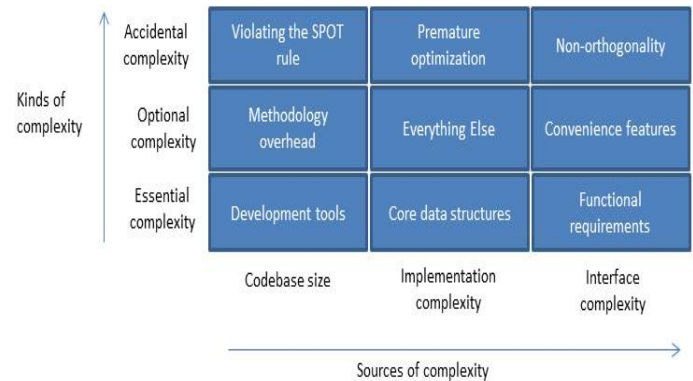


Figure 4.2, Sources and kinds of complexity²

From the diagram above, the relationship between kinds and sources of complexity is highlighted. The terms premature optimization, Non-Orthogonality, and SPOT etc. are further explained below

Non-Orthogonality: Occurs when interface operations do not perform exactly one task. This makes the interface more complicated, which causes interface complexity as shown in the diagram above.

Premature Optimization: This makes the code harder to understand when performing a task. Difficulty in understanding code (premature optimization) leads to an increase in implementation complexity.

SPOT (Single Point Truth): This rule lays emphasis on repetition of code because this is the major cause of inconsistency in UI design. When there is too much repetition, modifying the code when there is an error might be a problem. Also, code repetition increases the codebase size complexity as shown in the diagram.

Convenience Features: These are features that make usage of the system easier and they are not necessarily needed for proper functioning of the program. Sometimes adding too much “convenience features” can increase codebase size complexity thus making the interface more complicated. Better tools for development can be used to improve codebase size complexity. Using better algorithms can curb implementation complexity; additionally better interaction design can positively reduce complexity. Designers need to understand their users’ psychology (Raymond, 2003).

² (Raymond, 2003)

4.3 Characteristics of a Good User Interface Design

Having examined pervious literature on interface design, some of the characteristics of an Interactive Interface design are discussed subsequently.

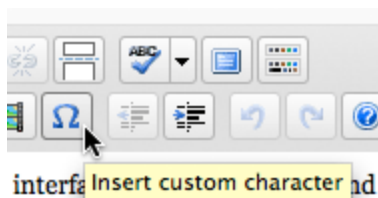
Responsiveness: This is essentially the speed of the interface or the software behind it. Users may get fed-up and abort the system if it takes too long to load or process information. Responsiveness may also refer to feedback from the interface. Feedback gives the user satisfaction because users always want to know what is going on. A spinning wheel or a progress bar could mean the application is loading; this gives the user a sense of comfort. A sample of a spinning wheel gotten from an article written by Brandon Walkin is shown below:

Loading dmitry@usabilitypost.com...



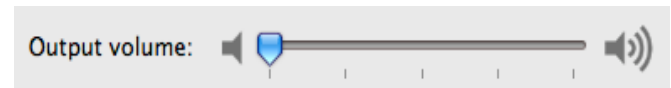
Familiarity: Here the designer tries to make the interface look like something the user has seen before. The key is making the interface something the user can naturally relate to integrating intuitive features to your design. For example, all close or cancel buttons are red. Making the close or cancel button on your design green may give the user a different idea of that feature.

Clarity: User Interface designers should always ensure that the tone, visual noise, design and hierarchy of elements are clear and easy to understand. A clear interface sends a direct message to users', making it easy for them to navigate through the interface. If an interface is clear enough, a user may not need to use a manual to understand the functionalities of the site and the user would be able to carry out task effectively. A Wordpress application provides a tooltip over each icon making it easy for users to understand the functions of each element. A diagram is shown below:



Concision: It is good to make an interface clear but over doing this can make the interface complex. Each time a

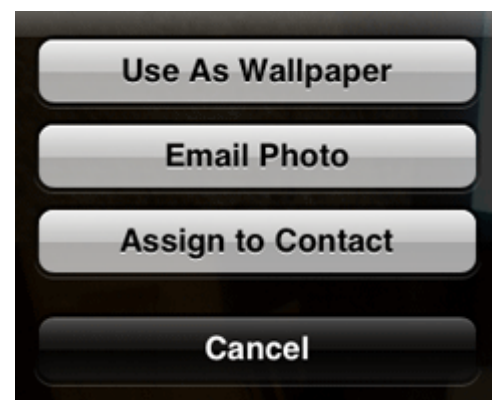
description is added to an icon or any other element, the interface adds mass. It's best to use short and concise words or sentences as the case may be. If bulky sentences are used to describe an element, it takes the user a lot of time to read and understand therefore making it stressful to navigate. The diagram below shows how small icons are used to represent the scale from low to high of OS X volume control:



Forgiveness: A designer must understand that users are bound to make mistakes. The interface should be designed to save users when they make a mistake. For example, if a user mistakenly deletes a file, the interface should provide an option to undo last action or should give the user the option of retrieving the document. The diagram below shows how Gmail helps you undo last action:

The conversation has been moved to the Trash. [Learn more](#) [Undo](#)

Efficient: UI designers should develop interface that is fast and easy to navigate with less effort. It is best to identify what your interface wants to achieve and design it to achieve that without any unnecessary additions. Make available only functions that need to be on the interface. The diagram below shows how apple identified the major task users perform with pictures on their Phone and provides an efficient interface to accomplish this:



Consistent: Designing a UI to be consistent makes it easy for users to understand or identify the elements of the interface in subsequent versions. This makes the user easily

identify usage patterns therefore reducing the time it takes to perform a given task. Consistency enables users to perform task effectively and efficiently therefore aiding user satisfaction. The diagram below shows the consistency of Microsoft Office:

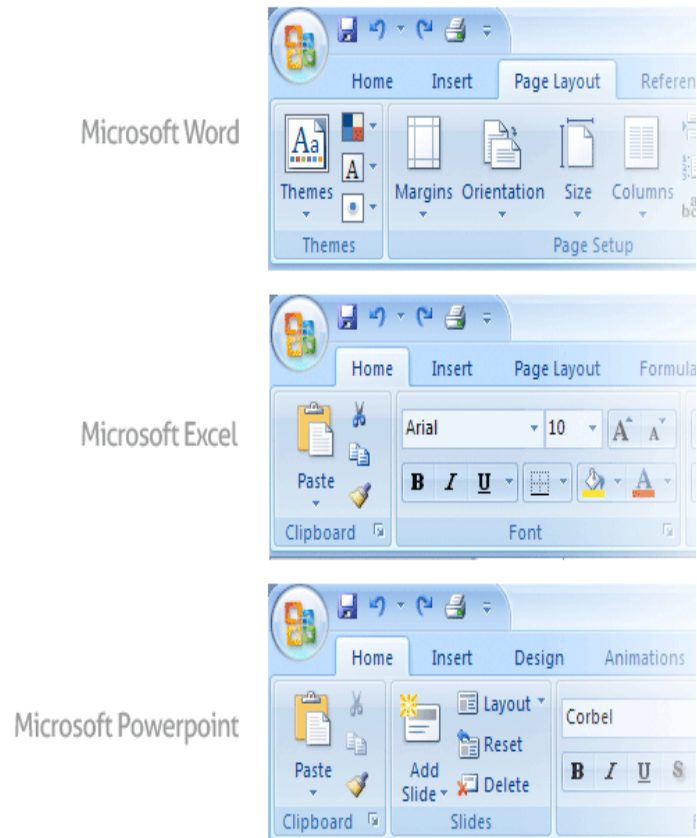


Figure 4.2, Microsoft Office (word, excel and PowerPoint)³

5 CHAPTER FIVE

5.1 Discussion

It is true that an interface can greatly affect productivity. A good interface would positively affect productivity and a bad interface will cause major problems in the utilization of an Information System. Many users like in the case of the Banner system in our study resented the system, they preferred to use their spreadsheets and self made access database rather than bother to understand the too technical processes associated with the interface and functionalities of the Banner system. As a consequence, there might have been instances of inconsistent data being entered into the

system; also all processes done using banner are much slower than even the previous manual system. Apparently all the blame cannot be put on the Banner system alone; another contributor of the difficulties of usage experience at the American University of Nigeria is the poor training. Users were not well trained enough to fully understand the system in order to use it effectively.

Many attempts to solving IS interface complexity have been made to address these problems. Recurrent notions of some solutions are: Interface **consistency** over versions on the software in order to create **familiarity**, **Concision** of messages, **interactive help** system, **user involvement** in the interface conception and **ease of use**. Some of the solutions have improved software usability and increased user satisfaction. The question is why do we still have complex and non-user friendly interfaces at this time and are there new ways this problem can find definite solution.

The reason people stand more than five minutes in front of an ATM machine or a self check-in post at the airport is because these systems lack intuitiveness. The system expects the human being to act like a robot, executing instruction in a predetermined and fixed order. But that is not the way the human being thinks and acts. In today's application-driven computing era, much is expected from software designers. Software solutions are supposed to solve human problems rather than imposing them a mechanical way of doing things. A few examples are being able to exit a process at any point. A user that does not want to go back to the previous page or pages would like to get where he wants to go just in one click. Further, the user would like to choose what to do rather than being asked to do things or being forced to answer questions. Finally the user wouldn't want to have any encounter with technical terms at all. Messages should be clear and in the everyday language. This is because nobody chooses to process information today, all of us are virtually forced to deal with IS interfaces one way or the other in this era of pervasive information system.

Conclusion

Human Computer Interaction and software interface design were intensely studied in the nineties. In those days Information System was still technology driven therefore some constraints that came from the limitation of the system were understandable and forgiven. Today the paradigm has completely shifted from the technology-driven IS to a more demanding application-driven IS.

³ (Walkin, 2009)

Because of the pervasiveness of IS, millions of users expect IS to behave a certain way well known to them. Imagine an IS where the delete function would be implemented under a floppy disk icon. This would be the best way to delete millions of files in the world. Almost all of us will expect that icon to perform the 'save' function and will validate the following message without even reading it. How many people still bother reading the user manual of the brand new handset or even the new laptop they just bought. The expectation is that it should operate exactly the same way the previous device did. Exception to that will cause a problem to the human being's cognitive mind. To make business flow well and consultant proficient at all time and all systems, ERP developers should begin to introduce some sort of standardization, consistency and ease of use in the design of their interface. This would help businesses productivity in that sense that when an expert is hired with knowledge in SAP he can be expected to be competent at the same level on Oracle.

References

- C.A. Manduca and D.W. Mogk. (2006). *Earth and Mind: How Geologists Think and Learn about the Earth*. GSA Books Boulder.
- Carroll, J. M. (2009). *Human Computer Interaction (HCI)*. Retrieved March 12, 2012, from Interaction-design.org: http://www.interaction-design.org/encyclopedia/human_computer_interaction_hci.html
- Carroll, J. M. (2009). *Human Computer Interaction*. Retrieved February 12, 2012, from Interaction-Design.org: http://www.interaction-design.org/encyclopedia/human_computer_interaction_hci.html
- Danino, N. (2001, November 14). *Human-Computer Interaction and Your Site*. Retrieved February 21, 2012, from Sitepoint.com: <http://www.sitepoint.com/computer-interaction-site/>
- Fakhreddine, K., Milad, A., Jamil, A. S., & Mo, N. A. (2008, March). *Human-Computer Interaction: Overview on State of the Art*. Retrieved from International Journal on Smart Sensing and Intelligent Systems: <http://www.s2is.org/Issues/v1/n1/papers/paper9.pdf>
- Fetaji, M., Suzana, L., Bekim, F., & Mirlinda, E. (2007). Investigating human computer interaction issues in designing efficient virtual learning environments. *Balkan Conference in Informatics*, (pp. 313-324). Sofia, Bulgaria.
- IconLogic. (2006). *Human Computer Interaction*. Retrieved from Icon Logic Learning Series: <http://www.iconlogic.com/pdf/HCI.pdf>
- Maass, W. (2012). *Ubiquitous Information Systems (UIS)*. Retrieved April 18, 2012, from <http://iss.uni-saarland.de/en/research/ubiquitous-information-systems/>
- Magee Christopher and de weck Oliver. (2004). *Complex SYstem Classification*. International Council On Systems Engineering (INCOSE).
- Mark, S. A., & Scott, D. M. (n.d.). *Privacy Issues and Human Computer Interaction*. O'Reilly & Associates, 6.
- Mary, B. R., & Carroll M., J. (2002). *Usability Engineering: Scenario Based Development of Human Computer Interaction*. San Francisco: Morgan Kaufmann Publishers, Academic Press.
- Myers, B., & Rosson, M. (1992). Survey on User Interface programming. *Proceedings SIGCHI'92: Human Factors in Computing Systems*, (pp. 195-202). Monterrey.
- Nanni, P. (2004, November 5). *Human-Computer Interaction: Principles of Interface Design*. Retrieved February 12, 2012, from VHML: http://www.vhml.org/theses/nannip/HCI_final.htm
- Peslak, A. (2005). A Framework and Implementation of User Interface and Human-Computer Interaction Instruction. *Journal of Information Technology Education*, 190-205.
- Quaye, A. K. (1990). *An Investigation of Human-Computer Interface Design Quality And Its Effects On User Satisfaction*. University of South Carolina.
- Raymond, E. S. (2003). *The Art of Unix Programming*.
- Thomas, H. T., Baecker, R., Card, S., Carey, T., Gasen, J., Mantei, M., et al. (1996). *Human-Computer Interaction*. New York: The Association for Computing Machinery.
- Tufti, E. R. (1989). *Visual Design of the User Interface*. New York: IBM Corporation.
- Vochin, A. (2009, July 17). *History of the computer mouse*. Retrieved from Softpedia:

<http://gadgets.softpedia.com/news/History-of-the-Computer-Mouse-3938-01.html>

Wachowiak, M. P., Wachowiak--Smolikova, R., & Friya, G. D. (2010). *Practical Considerations in Human-Computer Interaction for e-Learning Systems for People with Cognitive and Learning Disabilities*. North Bay, Canada: International Journal of Information Studies.

Walkin, B. (2009, August 10). *Managing UI complexity*. Retrieved January 29, 2012, from Brandon Walkin Blog: <http://www.brandonwalkin.com/blog/2009/08/10/managing-ui-complexity/>

Zhang, P., Jane, C., Dov, T., & Marilyn, T. (2004). Integrating Human-Computer Interaction Development into SDLC: A Methodology. *Proceedings of the Americas Conference on Information Systems, New York, August*, (p. 1). New York.

Software Development Methodology Revolution Based on Complexity Science - An Introduction to NSE Software Development Method

Chi-Hung Kao¹, Jay Xiong²

¹The Jumpulse Center of Research and Incubation of Northwestern Polytechnic University

²NSEsoftware., LLC., USA

Abstract - This article introduces NSE (Nonlinear Software Engineering) methodology based on complexity science. Complying with the essential principles of complexity science, especially the Nonlinearity and the Generative Holism principles results that the whole of a complicated system may exist before building up its components. The characteristics and behaviors of the whole system emerge from the interactions of its components, so that NSE software development methodology suggests almost all software development tasks and activities are performed holistically and globally. Complying with W. Edwards Deming's product quality principle that "Cease dependence on inspection to achieve quality. Eliminate the need for inspection on a mass basis by building quality into the product in the first place.", NSE software development methodology is driven by defect prevention and traceability from the first to the final step in order to ensure the quality of a software product. NSE software development methodology supports top-down plus bottom-up software engineering, makes software design become pre-coding, and coding become further design.

Keywords: software traceability, requirement traceability, validation, verification, testing, quality assurance, maintenance

1 Introduction - Almost All Existing Software Development Methodologies Are Outdated

Almost all existing software development methodologies are outdated because

(1) they are based on reductionism and superposition principle that the whole of a nonlinear system is the sum of its parts, so that almost all software development tasks and activities are performed linearly, partially, and locally.

(2) they are complied with the Constructive Holism principle that software components are developed first, then, as stated in CMMI, "Assemble the product from the product components, ensure the product, as integrated, functions properly and deliver the product." [1]

2. Outline of the Revolutionary Solution Offered by NSE

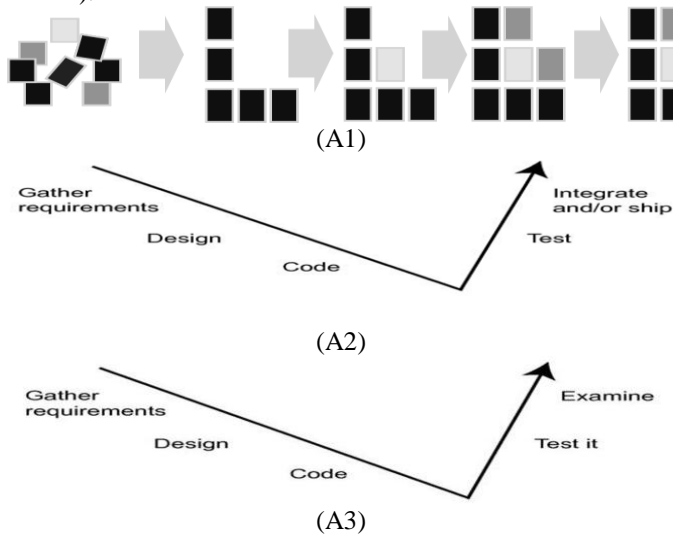
The revolutionary solution offered by NSE in software development methodology will be described in detail in this article later. Here is the outline of the solution:

- (1) It is based on complexity science by complying with the essential principles of complexity science, particularly the Nonlinearity principle and the Holism principle that the whole of a complex system is greater than the sum of its components – the characteristics and behaviors of the whole emerge from the interaction of its components, so that with NSE almost all tasks and activities in software development are performed holistically and globally. For instance, requirement changes are welcome to enhance customers' market competitiveness, and implemented holistically and globally with side-effect prevention through various traceability to avoid "Butterfly Effects".
- (2) It complies with the Generative Holism principle of complexity science that the whole of a complex system exists (as an embryo) earlier than its components, then grows up with them. As pointed by Frederick P. Brooks Jr. that "Incremental development – grow, not build software ... that the system should first be made to run, even though it does nothing useful except call the proper set of dummy subprograms. Then, bit by bit it is fleshed out, with the subprograms in turn being developed into actions or calls to empty stubs in the level below." [2] "An Incremental-Build Model Is Better – Progressive Refinement ... we should build the basic polling loop of a real-time system, with subroutine calls (stubs) for all the functions, but only null subroutines. Compile it; test it. ... After every function works at a primitive level, we refine or rewrite first one module and then another, incrementally growing the system. Sometimes, to be sure, we have to change the original driving loop, and or even its module interface. Since we have a working system at all times.
- (3) We can begin user testing very early, and we can adopt a build-to-budget strategy that protects absolutely against schedule or budget overrun (at the cost of possible functional shortfall)." [3] From the point of view of quality assurance, the NSE software development

methodology is driven by defect-prevention, defect propagation prevention, and traceability that a software quality is ensured from the first step down to the final one supported by various automated and self-maintainable traceability and software visualization. With NSE software development methodology software testing is performed dynamically in the entire software development lifecycle (including the requirement development phase, the product design phase, the coding phase, the testing phase, and the maintenance phase) using the innovated Transparent-box testing method (see Fig. 1 (B2)) [4] which combining functional testing and structural testing together seamlessly – to each test case, it not only checks if the output (if any, can be none – having a real output is no longer a condition to use this software testing method dynamically) is the same as what is expected, but also checks whether the real program execution path covers the expected one specified in J-Flow (a new type control flow diagram innovated by Jay Xiong), and then establishes the automated and self-maintainable traceability among the related documents, the test cases, and the source code to help users finding and removing the inconsistent defects. It means that the NSE software development methodology complies with W. Edwards Deming’s product quality principle that “Cease dependence on inspection to achieve quality. Eliminate the need for inspection on a mass basis by building quality into the product in the first place.” [5].

- (4) The defect prevention and defect propagation prevention also performed through software visualization in the entire software development process.

Fig. 1 shows a comparison of the software design strategy and the quality assurance strategy between the existing software development methodologies (part A) and the NSE software development methodology (part B).



Part B, the NSE software development methodology:

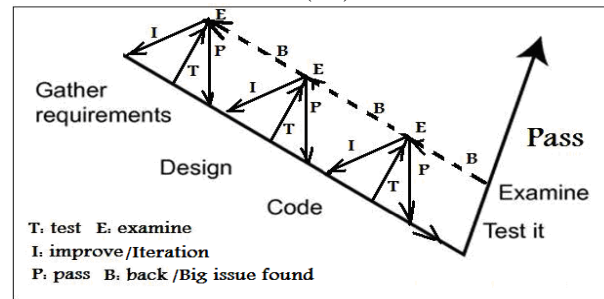
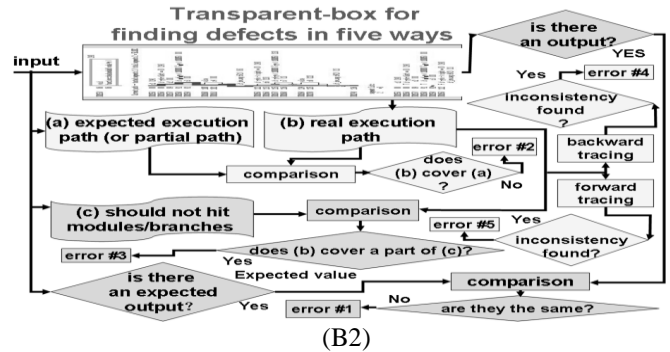
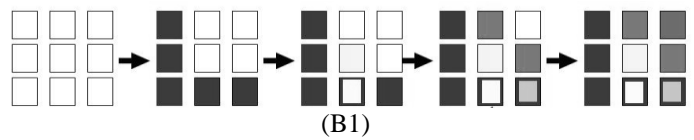


Fig. 1 A comparison of the software design strategy and the quality assurance strategy between the existing software development methodologies (part A) and the NSE software development methodology (part B) - (A1): The software product development strategy based on Constructive Holism principle that the components of a software product are developed first, then the whole system is built with the components; (A2): The quality assurance strategy for the incremental software development method – mainly depends on testing after coding using the black-box functional testing approach and structural testing approach [6]; (A3): The quality assurance strategy for the iterative software development method – also mainly depends on testing after coding using the black-box functional testing approach and structural testing approach [6]. (B1): The software product development strategy based on the Generative Holism principle that the whole of a software product exists first, then grows up with its components. (B2): The quality assurance strategy for the NSE software development methodology - mainly depends on defect prevention through dynamic testing using the Transparent-box testing method [4] combining functional testing and structural testing together seamlessly – to each test case it not only checks whether the output (if any, can be none when applied in requirement development phase and the design phase – having an output is not a condition to dynamically use this method) is the same as what is expected, but also checks whether the real execution path covers the expected path indicated in control flow diagram, and then establishes path automated and self-maintainable traceability among all related documents and the test cases and the source code through Time Tags automatically inserted into both the test case description part and the product test coverage

measurement database for mapping them together, and some keywords written in the test case description part to indicated the types of the related documents, the file locations, and the bookmarks for opening the traced documents from the corresponding locations, so that it can be used to find functional defects, structural defects, and inconsistency defects in the entire software development lifecycle. (B3): The defect prevention is performed mainly through dynamic testing using the Transparent-box testing method in all phases of a software development lifecycle.

The major features of NSE software development methodology:

- (1) It is visual – with NSE, the entire software development process and the obtained results are visible, supported by the NSE Software Visualization Paradigm.
- (2) The preliminary applications show that compared with the old-established software development methodologies based on reductionism and the superposition principle, it is possible for NSE software development methodology (working with the NSE software development process model) to help software development organizations increase their productivity, lower their cost, improve their product quality tenfold several times, and raise their project success rate.
- (3) It brings revolutionary changes to the CBSD (Components-Based Software Development) approaching by shifting the software component development foundation base from reductionism and the superposition principle to complexity science in order to greatly ensure the quality of the components themselves, and further make the components adaptive and maintainable as well. According to the principle of complexity science that the behavior and characteristics of a complex system is determined by both the whole of the system and its components, with NSE a software component used for CBSD should at least satisfy the following listed conditions:
 - * being 100% tested using the MC/DC (Modified Condition/Decision Coverage) test coverage metric, no matter whether it is provided as a class (a class can not be directly executed, so that the test coverage data should be collected through its instances) or a regular function;
 - * being verified that there is no memory leak or memory usage violation found;
 - * being verified that it will not become a performance bottleneck to the application system;

- * being verified that it will not bring bad effects to the file and the I/O systems for the applications;
- * being verified that it satisfies the quality standard in the corresponding applications;
- * being verified that it is provided with the related documents, the test cases, and (if possible) the source code traceable from and to the documents.

3 The Driving Forces for the Innovation of the NSE Software Development Methodology NSE Software Development Methodology is driven by defect-prevention and various automated and self-maintainable traceabilities

NSE Software Development Methodology is driven by defect-prevention and various automated and self-maintainable traceabilities:

(A) Defect prevention

Repeatable Defect Prevention through:

- (a) causal analysis,
- (b) preventive actions,
- (c) increase awareness of quality issues,
- (d) data collection, and
- (e) improvement of the Defect Prevention Plan.

New Defect Prevention (more useful) through bi-directional traceability to prevent

- (a) inconsistent or changed requirement definitions that may contain conflicts
- (b) inconsistent designs or design changes
- (c) inconsistent coding (such as inconsistencies between function definitions and calling statements)
- (d) inconsistent source code modification, etc.

(B) Traceabilities, including

- (a) automated and self-maintainable traceability among documents, test cases and source code, including the documents obtained from project planning, requirement development, product design, coding, testing, and maintenance. This type of traceability is essential to software validation, verification, debugging, and the identification of unimplemented requirements, useless source code modules,

requirements that are related to a module to be changed (for consistent modification), test cases that can be used for regression testing (whereby the efficiency of regression testing can be improved tenfold!), etc. This kind of traceability is established through dynamic testing using the Transparent-box method. Some Time Tags automatically inserted in the test cases description and the test coverage database would build the traceability between test cases and the source code.

- (b) the traceability between test cases and the source code has been extended to include all related documents using some keywords written in the description part of a test case to indicate the document formats, the file paths, and the corresponding bookmarks for showing the corresponding locations of the documents.
- (c) automated and bidirectional traceability within the source code, among source files, classes, functions, and detailed statements is established by diagramming the entire program. For instance, creating the traceability automatically between header file and "#include" statement, program tree and function body, function definition and function call statement, class instance and class definition, goto statement and label, etc. these types of traceabilities are essential for an efficient source code inspection and walkthrough, testing, bug checking, consistent source code modification, etc.
- (d) capability to trace a runtime error to the execution path and the related functions, This type of traceability is useful for debugging with testing.
- (e) automated traceability in a systematic analysis of software changes such as version comparison results at the system level, source file level, class and function level, and statement level would be displayed graphically. For instance it includes identifying which modules are deleted (shown in brown), added (shown in green), changed (shown in red), and unchanged (shown in blue). For a changed module, we can further trace the detailed source code to find which statements are deleted, added, and modified. This type of traceability is very useful for version comparison and debugging, particularly in the maintenance phase when some bugs have been removed but new bugs are found.
- (f) automated traceability among documents such as those for requirement management as specified in CMMI, including documents for requirement specifications, changes, comprehension, etc., in order to realize this type of automatic traceability, we use a set of predesigned templates in HTML/XML format. These templates will link together by themselves.
- (g) automatic traceability through all possible execution

paths for each module from a call graph, this kind of traceability is useful in identifying which other modules may be affected due to a change.

4 The related NSE software engineering process model

The NSE software development methodology works seamlessly with the NSE software engineering process model shown in Fig. 2.

NSE software engineering process model consists of three parts – the preprocess, the main process, and the support facility for automated and self-maintainable traceability among the related documents such as the requirement specification, the test cases, and the source code.

The main purpose of the preprocess is to assign priority to the requirements according to the importance, perform prototyping for the important and unfamiliar requirements to reduce risk, execute the function decomposition for functional requirements and system preliminary design through dummy programming to form the whole of a software system. For instance, an embryo using dummy modules contains an empty body including only some calling statements for other low-layered ones without detailed programming logic – see the Completeness Percentage axis of the graphical description of the NSE software development methodology shown in Fig. 3. The “Bone” system (about 5% of the product effort, the first milestone) is obtained in the preprocess.

The implementation of requirements is performed with the main process incrementally through two-way iteration supported by automated and self-maintainable traceability. It is recommended to complete the implementation of about 20% of the most important requirements to form an essential version of the product – see the Completeness Percentage axis of the graphical description of the NSE software development methodology shown in Fig. 3; corresponding to the “Essential” version (second milestone) of the product completeness.

After that, the whole system grows up with more incremental implementations of the requirements, until the final product is completed. With the NSE software development methodology, all versions including the “Bone” system are executable (even if there is no real output provided), and delivered to the customer for review and the feedback may be used for improvement.

As shown in Fig. 2, the NSE software engineering process model is a nonlinear one which assumes that the upper phases might introduce defects or some mistakes, so to check the inconsistency with the upper phases to improve the product is required – as a critical issue is found, there may be a need going back to the preprocess to design a better solution method for the corresponding requirement(s), and perform the prototyping again.

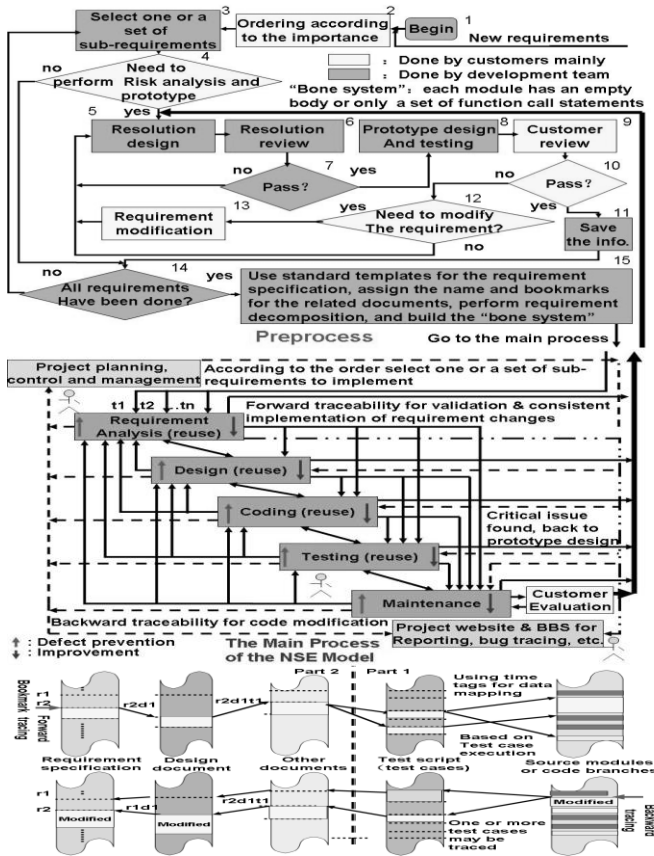


Fig. 2 The NSE Process Model which includes the preprocess part, the main process part, and the automated and self-maintainable facility to support bi-directional traceability using Time Tags automatically inserted into both the test case description part and the corresponding test coverage database for mapping test cases and the tested source code, and some keywords to indicate the related document types such as @WORD@, @HTML@, @PDF@, @BAT@, @EXCEL@ written in the test case description part followed by the file paths and the bookmarks to be used to open the traced documents from the specified positions.

5. Graphical Presentation of the NSE Software Development Methodology

The graphic description of the NSE software development methodology is shown in Fig. 3.

As shown in Fig. 3, there are three axes representing the Work Flow, the Time, and the Completeness Percentage separately.

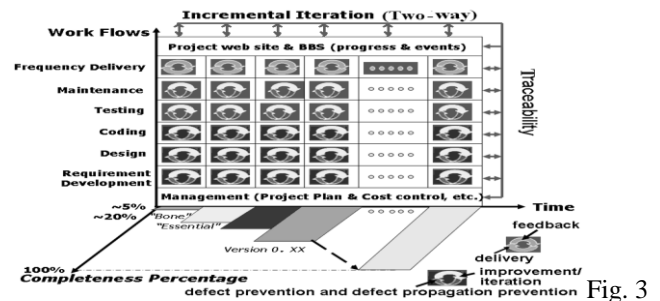


Fig. 3 NSE software development methodology

In the Work Flow axis, it not only includes the phases of requirement development, design, coding, testing, and maintenance, but also includes the project management, the product delivery, and the support for the product web site and BBS for communication – it combines the product development and maintenance together, and furthermore integrates software development and project management together. No matter in what phase, defect prevention and defect propagation prevention is performed to ensure the quality of the product being developed. It does not always follow a linear order – as shown on the right side, upstream movement is supported through traceability for two-way iteration, if necessary.

The Time axis represents the progress. The Completeness Percentage axis shows how many percents of the product are completed – there are three milestones: the first one is the “Bone” system completed through dummy programming; the second one is about 20% of the most important requirements have been implemented; the third one is the final product. The “Bone” version is completed through the preprocess – after prototyping and risk analysis, system decomposition of the functional requirements will be performed, then the decomposition result will be used for the preliminary design to establish the “Bone” system through dummy programming (each dummy module has an empty body or a list of function call statements without detailed program logic). The “Essential” version of the product is completed in the main process incrementally for most important requirements (about 20% of the initial requirements).

Often in the final product, the number of the requirements will be doubled or even more – NSE supports requirement changes in both the software development process, and the software maintenance process with side-effect prevention through various traceabilities.

Additional instructions on sections and subsections

Avoid using too many capital letters. All section headings including the subsection headings should be flushed left.

6 Application

As described above, the NSE software development methodology is driven by defect prevention, defect propagation prevention, and traceability mainly through dynamic testing using the Transparent-box testing method, and software visualization in the entire software development process.

NSE software development methodology supports top-down plus bottom-up approach – design becomes pre-coding, and coding becomes further design as described graphically in Fig. 4.

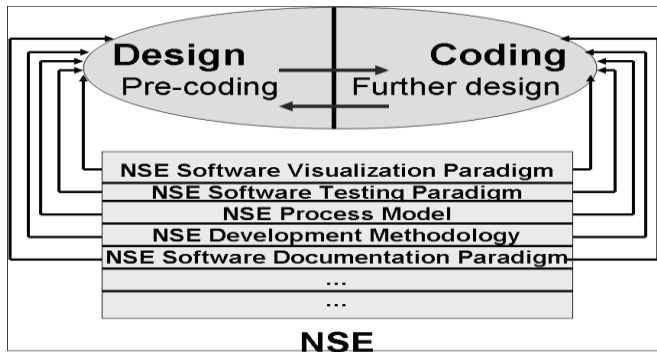


Fig. 4 With NSE software development methodology design becomes pre-coding, and coding becomes further design

With NSE, the preliminary design of the whole of a software system is performed in the preprocess (see Fig. 2) through stub programming using dummy modules based on the results of the function decomposition following the functional requirements and the description in the non-functional documentations.

Directic coding from the design result:

With NSE software development methodology, coding can be performed directly by editing the dummy modules designed to extent the program logic as shown in Fig. 5.

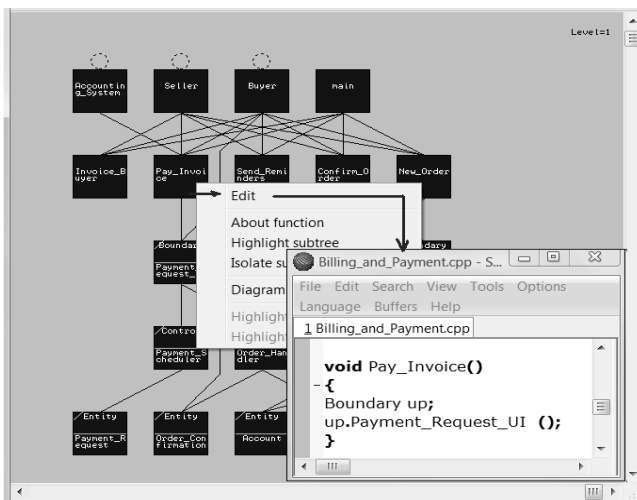


Fig. 5 Directly coding from a call graph generated in design process

With NSE software development methodology, coding becomes further design - for instance, in the case that the design shows function A calls function B, but the coding engineers find the function A should call function C and function C should call function B - after coding they can update the design documents by rebuilding the database to make the design result consistent with the code (in this case, they may choose the way to modify the design first, then edit and change the code) as shown in Fig. 6 and Fig. 7.

```

State.h - SciTE
File Edit Search View Tools Options Language Buffers Help
1 State.h
class state4:public state0
{
public:
int transition(unsigned char ch)
{
if( (ch >= '0') && (ch <= '9') ) return 4;
else if( (ch == '|') && (get_paren()>0) )
{
dec_paren();
return 9;
}
else if(ch=='+'||ch=='-'||ch=='/'||ch=='*') return 7;
else if(ch == '.') return 6;
else if( ch == null&&get_paren() == 0 )
return 8;
else return 10;
}
};
    
```

Fig. 6 Two function call statements are added in the coding process of the state4::transition(unsigned char) module designed without using them



Fig. 7 After rebuilding the database, the design result can be updated automatically

Holistic development :

With NSE software development methodology, the whole of a software system will be designed first using stub programs using dummy modules as shown in Fig. 8.



Fig. 8 A holistic call graph designed

Static traceability for defect removal through review:

With NSE software development methodology, the design results are traceable for static defect removal as shown in Fig. 9.

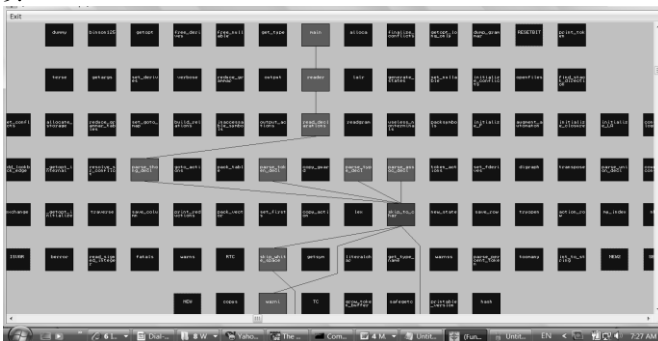


Fig. 9 A module and the related modules highlighted for static defect prevention and defect propagation prevention

System growing up incrementally with defect prevention:

With NSE software development methodology, a software system being developed will grow up incrementally as shown in Fig. 10 and Fig. 11.

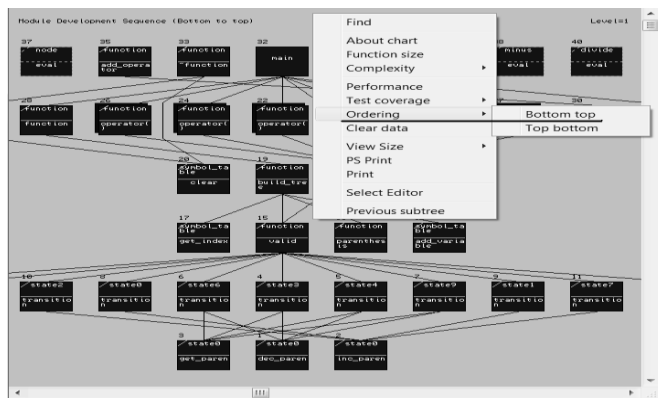


Fig. 10 Incremental coding ordering support

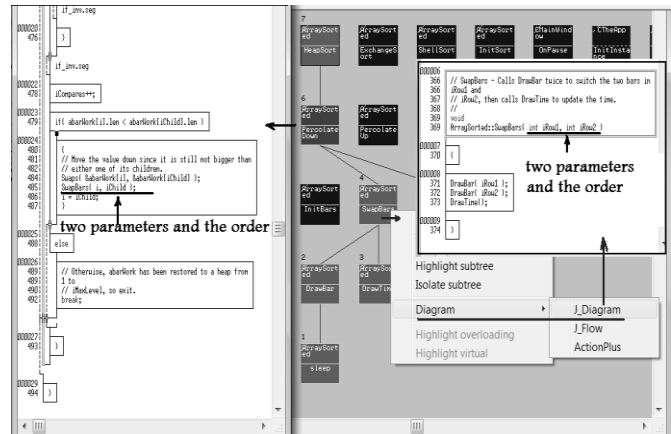


Fig. 11 Defect prevention through incremental ordering and visualization

Coding style independent visualization support:

With NSE software development methodology, it is supported by a coding-style-independent graphical representation technique and tools, so that the source code written by others is also easy to read and understand – see Fig. 12;

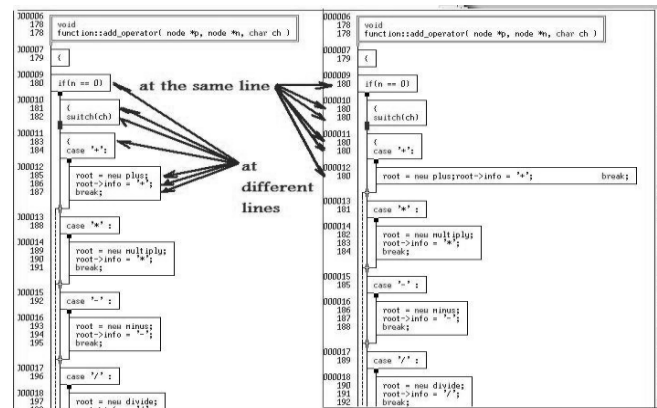


Fig. 12 code-style-independent program representation

Dynamic traceability among documents and test cases and source code for defect removal:

With NSE software development methodology, quality is ensured mainly through defect prevention and defect propagation prevention based on dynamic testing using the innovated transparent-box testing method to establish bi-directional traceability as shown from Fig. 13 to Fig. 16.

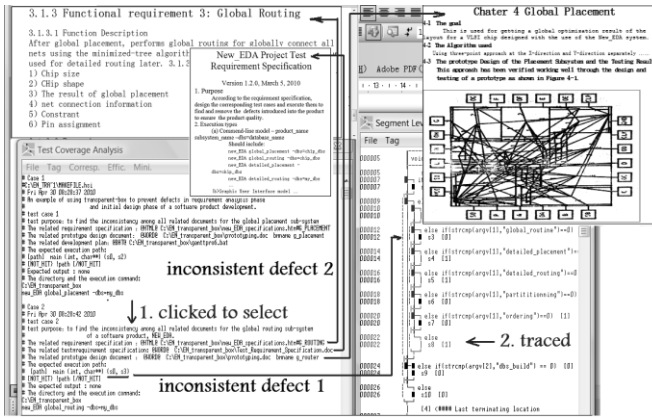


Fig. 13 Tracing a test case to find two inconsistency defects:

- (1) The real execution path did not cover the expected execution path `main (int, char**) {s0, s3} – segment s3 is highlighted as untested;`
- (2) The bookmark for opening the global routing section of the prototyping document, `g_router`, pointed to the wrong section – the global placement section.

Removing the defects:

- (1) find the location for the first defect and modify the `main()` program:

From:

```
else if(strcmp(argv[1], "global_routing") == 0)
    // calling g_routing(argv[2])
```

TO:

```
else if(strcmp(argv[1], "global_placement") == 0)
    // calling g_placement(argv[2])
```

- (2) find the mistake related to the bookmark, `g_router` (Fig. 14), and fix it (Fig. 15):

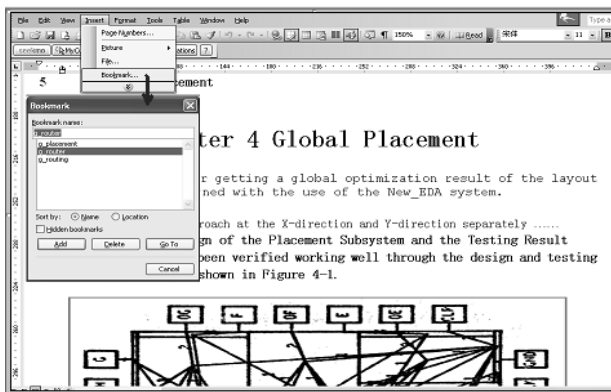


Fig. 14 Locating the mistake of the bookmark, g-router

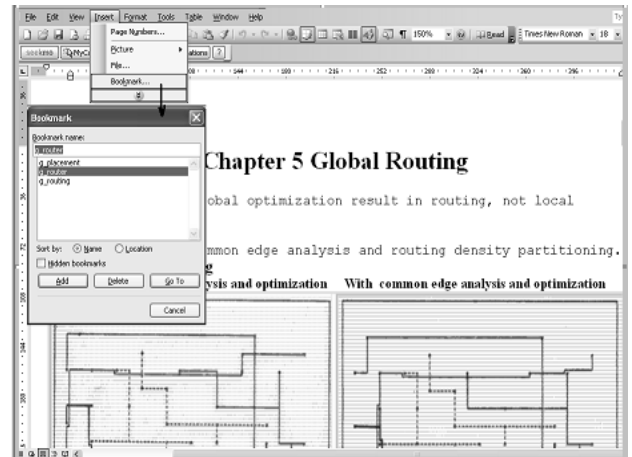


Fig. 15 Fixing the bookmark mistake

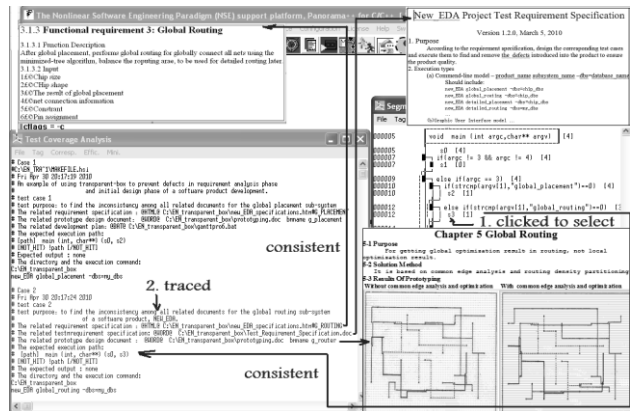


Fig. 16 Verifying that the two defects have been removed through backward tracing from segment s3 (the test case 2 was traced and the related documents were opened without defects)

7 Conclusion

This article presents the NSE software development methodology based on complexity science. It is driven by traceability, defect prevention, and defect propagation prevention through dynamic testing using the Transparent-box testing method and software visualization. Preliminary applications show that compared with the existing software development methodologies it is possible for the NSE software development methodology (with NSE process model and the support platform) to help software development organizations efficiently solve many critical issues in software development to ensure software quality and increase software development productivity.

8 References

[1] Mike Phillips, CMMI Program Manager, CMMI V1.1 and Appraisal Tutorial, <http://www.sei.cmu.edu/cmmi/>, slide 118, titled “Product Integration”.

- [2] Brooks, Frederick P. Jr., "The Mythical Man-Month", Addison Wesley, 1995, P200
- [3] Brooks, Frederick P. Jr., "The Mythical Man-Month", Addison Wesley, 1995, P267
- [4] Jay Xiong, Jonathan Xiong, *A Complete Revolution in Software Engineering Based on Complexity Science*, WORLDCOMP'09 – SERP ([Software Engineering Research and Practice 2009](#)) , 109-115.
- [5] Deming, W. Edwards (1986). *Out of the Crisis*. MIT Press. [ISBN 0-911379-01-0](#). [OCLC 13126265](#)
- [6] Alistair Cockburn, *Using Both Incremental and Iterative Development*, CrossTalk, May 2008 Issue

Software Engineering Process Revolution

Chi-Hung Kao¹, Jay Xiong²

¹ NSEsoftware., LLC, USA

² Northwestern Polytechnic University

Abstract - This article introduces the NSE (Nonlinear Software Engineering paradigm) process model based on complexity science indicating that almost all tasks/activities are performed holistically and globally. Some applications show that the techniques and supporting platforms of NSE process model can not only make revolutionary changes to almost all aspects in software engineering for efficiently handling software complexity, invisibility, changeability, and conformity, but solve the critical problems such as low productivity and quality, high cost and risk, existing with the old-established software engineering paradigm. The NSE helps software developers raising productivity, dropping costs, and removing dramatic amount of the defects in their products.

Keywords: software process model, software engineering revolution, methodology, testing, quality assurance, productivity, maintenance

1. Almost All of the Existing Software Engineering Process Models Are Outdated

Almost all existing software engineering process models, no matter if they are waterfall models, incremental development models, iterative development models, or a new one recommended by Alistair Cockburn combining both incremental and iterative development together[1], are outdated because they are linear models with only one track forward in one direction without upstream movement at all, requiring software developers and the customers always do all things right without making any mistake or any wrong decision – but it is impossible, complying with the superposition principle that the whole of a system is the sum of its parts, so that almost all tasks/activities are performed linearly, locally and partially, making the defects introduced into a software product at the upper phases easy to propagate to the lower phases and the defect removal cost increase tenfold several times.

The common drawbacks of the existing software process models also include:

- (a) None of them are created to efficiently handle the essential issues existing with software products – complexity, invisibility, changeability, and conformity, defined by Brooks[2].

- (b) None of them are able to efficiently solve the most critical problems with software development - low quality and productivity, and high cost and risk.

- (c) None of them are able to make significant improvement to the software project success rate, so that today the software project success rate is still at about 30%[3] - it is unacceptable in any other industry.

- (d) Incomplete - None of them are able to efficiently support software maintenance which takes 75% or more of the total effort and cost for software product development[4], because they do not satisfy the following listed essential conditions for an efficient software maintenance support:

- (1) being able to greatly reduce the amount of defects introduced into a software product and the defects propagated to the software maintenance phase through defect prevention and defect propagation prevention;

- (2) being able to help users perform software maintenance holistically and globally;

- (3) being able to help users prevent the side-effects for the implementation of requirement changes or code modifications;

- (4) being able to provide necessary means to help users greatly reduce the time, cost, and resources in regression testing after software modification, such as the capability for test case minimization, and intelligent test case selection through backward traceability from a modified module or segment (a set of statements with the same execution conditions);

- (5) being able to help the customer side maintain a software product developed by others with almost the same conditions as it is maintained by the product development side (see table 2 about the “Software” definition).

NSE process model with “two-way iteration and multiple tracks” satisfies the five conditions. It is possible for the NSE process model with the support techniques and tools to help software development organizations reduce 2/3 of the total effort and total cost spent in software maintenance - equal to double their productivity and halve their cost. It is important to point out that with NSE there is no major difference between the software development process and the software maintenance process, because:

* both processes support requirement changes and code modifications with side-effect prevention through various bidirectional traceabilities.

* When the NSE nonlinear process model is followed, the quality of a software product is ensured from the first step (see section 7) down to the last step in maintenance through defect prevention and defect propagation prevention, so that the defects propagated to the maintenance phase are greatly reduced.

2. The foundation of NSE and the NSE process model – complexity science

Complexity science has been called the science of the 21st century by Stephen Hawking and Edward O. Wilson.

The essential principles of complexity science complied with by the NSE process model include the:

Nonlinearity principle

Holism principle - that all the properties of a given [system](#) cannot be determined or explained by its components alone. Instead, the characteristics and behavior of the whole of a complex system emerge from the interaction of its components and the interaction between it and the environment.

Initial Condition Sensitivity principle

Sensitivity to Change principle

Dynamics principle

Openness principle

Self-organization principle, and

Self-adaptation principle

NSE engineering process model is innovated through the use of a paradigm-shift framework, FDS (the Five-

Dimensional Structure Synthesis method - a paradigm-shift framework innovated by Jay Xiong) as shown in Fig. 1.

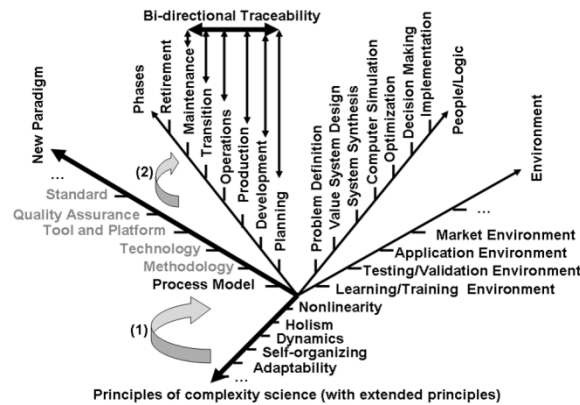


Fig. 1 The Five-Dimensional Structure Synthesis method - a paradigm-shift framework

3. Advanced techniques innovated to support NSE and the NSE process model

Fourteen advanced software engineering techniques are innovated to support NSE process model as shown in Fig. 2. Our two related papers titled as “Automated and Self-maintainable Traceability” and “Software Testing Revolution” are accepted by CrossTalk for publication. About the other 12 techniques, please see table 2, or read our published article titled as “A complete revolution in software engineering based on complexity science”[5]

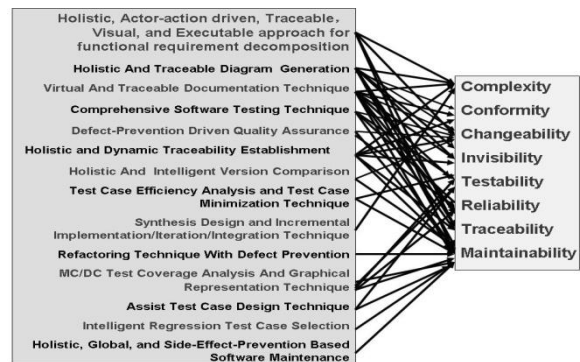


Fig. 2 The support techniques and the targeted issues

4. Description of the NSE process model

Number section and subsection headings consecutively in numbers and type them in bold. Use point size 14 for section headings and 12 for subsection headings and 10 for subsection within a subsection.

The NSE process model (Fig. 3) consists of the pre-process part and the main process part which is supported by a facility for automated and bi-directional traceability using Time Tags for data mapping and bookmarks for opening a document traced from the corresponding location.

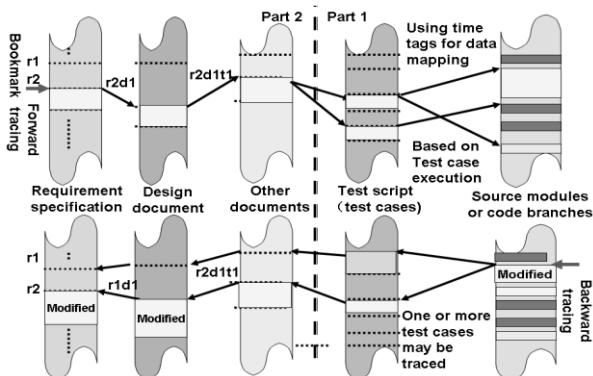
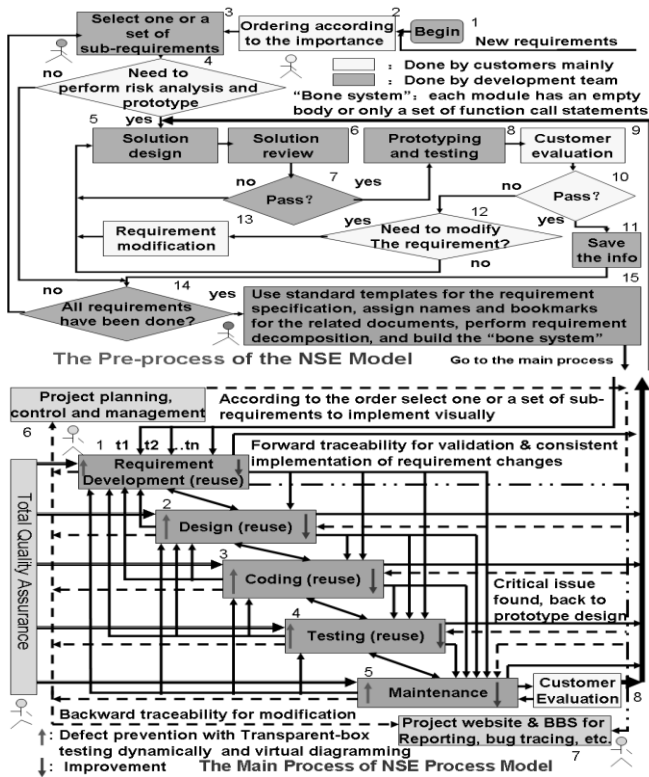


Fig. 3 The NSE Process Model and the automated and self-maintainable traceability facility

The objectives of the pre-process are:

a) Assigning priority to the requirements for better control of the development schedule and the budget, implementing the important requirements earlier;

b) Performing prototyping design and evaluation for some unfamiliar requirements to reduce project development risk;

c) Performing function decomposition of the functional requirements using the Holistic, Actor-Action and Event-Response driven, Traceable, Visual, and Executable technique (HAETVE) to replace the Use Case approach which is not holistic, and the results obtained are not traceable and not directly executable for defect removal. d) Making a primary version of the requirement specification document using standard templates provided to prevent defects of missing something;

e) Carrying out Synthesis Design of the system using the “Dummy programming” technique through the use of dummy modules to complete a dummy system. According to the Generative Holism Theory of complexity science, the whole of a complex system may not be “built” from its components, but exists (like a human embryo) earlier than its parts, then “grows up” with its parts;

f) Organizing the document hierarchy using bookmarks, including the test scripts and the test case numbers, so that when there is a need to modify a requirement, it is easy to find the related test scripts and the test cases to perform forward tracing to find the related documents and the source code.

The Major Steps Of The Main Process

Step 1: According to the project development plan, the priority assigned to the requirements, take one or a set of requirements to implement visually. It is recommended to select the critical and essential requirements (about 20% of the initial requirements) first to implement and form an essential version of the software product through incremental integration development to make a software system grow up incrementally.

The NSE process model supports the NSE software development methodology based on Generative Holism and driven by defect prevention and various traceabilities - see Fig. 4.

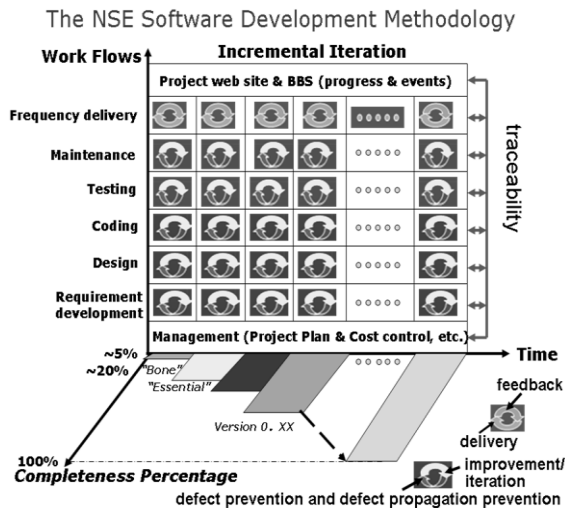


Fig. 4 NSE software development methodology

Step 2: Apply the Synthesis Design and Incremental Implementation, Iteration, and Integration Technique with the Holistic And Traceable Diagram Generation technique to further perform preliminary design for the selected requirement(s) according to the detailed requirement specification to improve the corresponding part of the dummy system obtained in the preprocess phase, then perform formal inspection and review using traceable documents, and design the corresponding test cases to dynamically test the result of the preliminary design using the Transparent-box method (see table 2) to prevent inconsistency defects through bidirectional traceability established automatically. After that, perform detailed design for the selected requirement(s) according to the result of the preliminary design with formal inspection and review using traceable documents, and dynamic testing like what was done in the preliminary design process. If something critical is found, go to the requirement development phases, or if the solution method does not satisfy the requirement(s), go back to the preprocess.

Step 3: Apply the Synthesis Design and Incremental Implementation, Iteration, and Integration Technique to perform incremental coding: on the generated system decomposition chart, highlight the corresponding key module(s) and the related modules for the selected requirement(s), then assign an incremental bottom-up coding order to the modules automatically with the NSE support platform.

When we are writing a function call statement to a called module which has been coded according to the order assigned, we can read the diagrammed source code in another window to know how many parameters are needed, their types, and their sequence to prevent inconsistency defects between the module interfaces.

If something critical is found in the coding process, go to the upper phases through backward tracing, or if the solution

method does not satisfy the requirement(s), go back to the preprocess again.

Step 4: Perform incremental unit testing with integration testing, and finally system testing, mainly using the transparent-box method to combine functional and structural testing together. At the same time, perform MC/DC (Modified Condition/Decision Coverage) test coverage analysis, performance analysis, memory leak analysis and memory usage violation check. According to the incremental coding and unit testing order, when we code a module, all modules called by it must have been coded already so that there is no need to design and use a stub module to replace a called module – in this way the unit testing also becomes integration testing with all modules being called together.

If something critical is found in the testing process, go to the upper phases through backward tracing, or if the solution method does not satisfy the requirement(s), go back to the preprocess again.

In the system testing process, NSE offers the capability to capture users' GUI operations and play them back automatically for regression testing, and the capability for MC/DC test coverage analysis for the entire product, plus performance analysis, test case efficiency analysis and test case minimization for efficient regression testing after code modification. With system testing, an automated and bidirectional traceability among all artifacts including the source code will be established for defect prevention.

Step 5: Perform systematic, disciplined, and quantifiable software maintenance using the Holistic, Global, and Side-Effect-Prevention Based Software Maintenance Technique:

- (1) Respond to requirement changes and new requirements or code modifications in real-time to implement them holistically and globally with side-effect prevention.
- (2) Bring great savings to regression testing after requirement changes or code modification through test case efficiency analysis and test case minimization, plus intelligent test case selection through backward traceability between test cases and the source code.
- (3) Make it possible to reduce the cost and effort spent in software maintenance from 75% or more of the total with the old-established paradigm to about 25% of the total with NSE through side-effect prevention. If there is still something wrong after the implementation of requirement change or code modification, perform intelligent version comparison to help users locate the defects in system-level, file-level, module-level, and statement level.

Step 6: Closely combine the project management process and the product development process together, making the project plan, schedule charts, and cost estimation reports traceable with the requirement implementation and the source code, for better control of the cost and project development schedule.

Step 7: Establish a project web site and the related technical forum for real time communication and technical discussion among team members to report progress of the project, and to open technical discussions for brainstorming, reporting a variety of related events, error handling processes and results, and especially unexpected events in order to discuss the response, which can all be traced back through the bi-directional and automatic traceability mechanism to update them in real time.

Step 8: Frequently deliver working products to the customer for review and evaluation, even if there is no real output for a dummy system designed in the requirement development phase. Get the customer's feedback to improve the product development.

5. Quality assurance with NSE process model –NSE SQA

The quality assurance priority of NSE-SQA is as follows:

- a) Defect prevention in all phases for preventing repeatable defects and possible new defects
- b) Defect propagation prevention (removing defects from the source) through:
 - (1) semi-automated inspection and walkthrough using traceable artifacts and diagrammed source code; (2) transparent-box testing in all phases
- c) Refactoring based on complexity analysis (20% of complex code causes 80% of the defects)
- d) Deep and broad testing, and quality measurement.

5.1. Highlights of the NSE-SQA

(a) based on defect prevention, defect propagation prevention, inspection and review in the entire lifecycle using traceable documents and source code, refactoring based on complexity measurement and performance analysis, and deeper and broader software testing plus quality measurement

(b) possible to remove 99% to 99.99% of defects of a software product - a detailed comparison on defect removal efficiency is shown in table 1 (The data reported by SPR [6] through the analysis of more than 12,000 projects with the

old-established software engineering paradigm is shown in *italic*; data with NSE SQA is shown in **bold**).

6. The major features of NSE process model

The major features and characteristics of the NSE process model include:

- (1) Dual-process: NSE model consists of the pre-process and the main process. They are different but also closely linked together.
- (2) Nonlinear: The NSE model is nonlinear, complying with the Nonlinearity principle and the Holism principle.
- (3) Parallel with Multiple tracks: "Much of software architecture, implementation, and realization can proceed in parallel." [7]. For reducing waiting time and speeding up software development processes, the NSE process model supports tasks being performed in parallel with multiple tracks through bidirectional traceability.
- (4) Real time: "Timely updating is of critical importance." [8].
- (5) Incremental development with two-way iteration: The NSE process model supports incremental development with two-way iteration, including refactoring to handle highly complex modules and performance bottlenecks with side-effect prevention. When a critical issue is found in the main process, the work flow may go back to the preprocess for selecting a better solution method, and so on.
- (6) The software development process and software maintenance process are combined together seamlessly: With the NSE process model, there is no big difference between the software development process and the software maintenance process – both support requirement changes through side-effect prevention.
- (7) The software development process and the project management process are combined together closely: all documents including the project management documents such as the project development plan, the schedule chart, and the cost estimation report are traceable with the requirement implementation and the source code for better

control of the product development. NSE process model also supports the critical requirements and most important requirements being implemented early with the assigned priority to avoid budget overuse – if necessary, some optional requirements and not so important requirements can be ignored temporarily.

- (8) Adaptation focused rather than predictability focused: the entire world is always changing, so the NSE process model is adaptation focused rather than predictability focused – it supports requirement changes, code modifications, data modifications, and document modifications to make them consistent and updated with side-effect prevention.
- (9) Defect prevention driven
- (10) People are considered as the first order driver for software development - When people consider “people as the first-order” to software development, they focus on how to trust and support people better for their jobs, but ignore the other side of people’s effect on software development – almost all defects introduced into software products are made by people, the developers and the customers. So NSE supports people in two ways: one is to support them with better methodology, technology, and tools; another one is to prevent the possible defects to be introduced into the software products by people - it is done mainly through various automated and bidirectional traceabilities.

7. Conclusions

This paper described a new revolutionary software engineering process model based on complexity science – the NSE process model, where almost all software engineering tasks/activities are performed holistically and globally. Preliminary applications show that compared with the existing linear process models it is possible for NSE to help users double their productivity and halve their cost, and remove 99% to 99.99 defects in their software products.

8. References

- [1] Alistair Cockburn, *Using Both Incremental and Iterative Development*, CrossTalk, May 2008 Issue
- [2] Brooks, Frederick P. Jr., “The Mythical Man-Month”, Addison Wesley, 1995, P182
- [3] Watts S. Humphrey, The Software Engineering Institute , Why Big Software Projects Fail: The 12 Key Questions , [CrossTalk Mar 2005](#)
- [4] Scott W. Ambler. “A Manager’s Introduction to The Rational Unified Process (RUP)”, Ambysoft. 2005
- [5] Jay Xiong, Jonathan Xiong, *A Complete Revolution in Software Engineering Based on Complexity Science*, WORLDCOMP’09 – SERP ([Software Engineering Research and Practice 2009](#)) , 109-115.
- [6] Jones, Capers, SOFTWARE QUALITY IN 2002: A SURVEY OF THE STATE OF THE ART, Six Lincoln Knoll Lane , Burlington, Massachusetts 01803 <http://www.SPR.com> July 23, 2002
- [7] Brooks, Frederick P. Jr., “The Mythical Man-Month”, Addison Wesley, 1995, P233
- [8] Brooks, Frederick P. Jr., “The Mythical Man-Month”, Addison Wesley, 1995, P235
- [9] Pressman, Roger S., “Software Engineering: A Practitioner’s Approach”, McGraw-Hill, 2005, P4

Software Testing Revolution Based on Complexity Science - An Introduction to NSE Software Testing Paradigm

WanYuan Huang¹, Jay Xiong²

¹The Jumpulse Center of Research and Incubation of Northwestern Polytechnic University

²NSEsoftware, LLC., USA

Abstract - This paper presents a new software testing paradigm (NSE software testing paradigm) based on complexity science using the Transparent-Box testing method which combines functional testing and structural testing together seamlessly with close logic connection and a capability to automatically establish bidirectional traceability among the related documents and test cases and the corresponding source code. To each test case it checks not only whether the output (if any, can be none when it is dynamically used in requirement development phase and design phase) is the same as what is expected, but also helps users to check whether the real execution path covers the expected one specified in the control flow, and whether the execution hits some modules or branches which are prohibited for the execution of the corresponding test case, so that it can be used to find functional defects, logic defects, and inconsistency defects. Having an output is no longer a condition to apply this method, so that it can be used dynamically in the entire software development lifecycle for defect prevention and defect propagation prevention. With NSE (Nonlinear Software Engineering paradigm) software testing is performed nonlinearly, holistically. Globally, and quantitatively in all phases of a software product development.

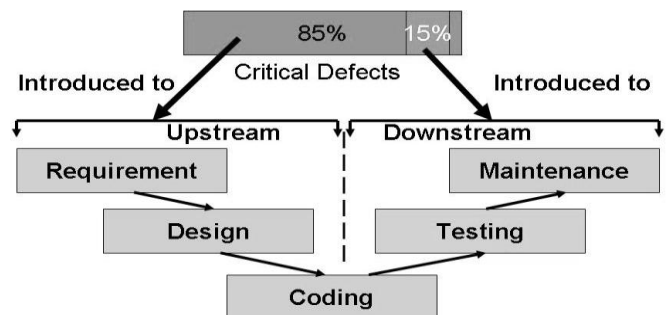
Keywords: software testing, method, reliability, quality assurance

1. Introduction

The purpose of software testing is to validate/verify whether a software product meets the customers' needs and the requirement specifications, find and fix bugs to help users increase the reliability of a software product.

Unfortunately, current software testing methods are outcomes of linear thinking, reductionism, and the superposition principle that the whole of a nonlinear system is the sum of its parts, so that with them almost all software testing activities are performed linearly, partially, locally, qualitatively, inefficiently, and blindly such as the regression testing of the implementation of requirement changes or code modifications. For instance,

as shown in Fig. 1 most critical software defects are introduced into a software product in the requirement development phase and the design phase, but current dynamic software testing is performed after detailed coding – it is inefficient and too late.



About 85% of the critical defects are introduced to a software at upstream, but the testing methods can only be used dynamically in downstream

Fig. 1 Current software testing is inefficiently performed after coding

2. The major existing software testing methods, techniques, and tools are outdated

Current software quality assurance is mainly based on (1) functional testing using Black-Box testing method being applied after the entire product is produced; (2) structural testing using White-Box testing method being applied after each software unit is coded; and (3) product review using untraceable documents and untraceable source code. It violates Deming's Product Quality Assurance Principles that *"Cease dependence on inspection to achieve quality. Eliminate the need for inspection on a mass basis by building quality into the product in the first place."*^[1]

Both testing methods are applied separately without internal logic connection. The white-box testing is mainly performed in unit testing to test an **Existing product** rather than a **Required product**, while the black-box testing is mainly performed in system testing, so that both methods and the corresponding techniques and tools cannot be used dynamically in the requirement development phase and the software design phase. Even

if a requirement development defect or a design defect can be found by both methods after coding, it is too late: the cost for removing the defect will increase tenfold several times.

For those software testing methods, NIST (National Institute of Standards And Technology) concluded that “Briefly, experience in testing software and systems has shown that testing to high degrees of security and reliability is from a practical perspective not possible. Thus, one needs to build security, reliability, and other aspects into the system design itself and perform a security fault analysis on the implementation of the design.” (“Requiring Software Independence in VVSG 2007: STS Recommendations for the TGDC.” November 2006

Those software testing methods and the related techniques and tools are designed to work with the old-established software engineering paradigm based on linear thinking and the superposition principle that the whole of a system is the sum of its parts, so that almost all tasks/activities are performed linearly, partially, and locally, making the defects introduced in upper phases easy to propagate to the lower phases to increase the defect removal cost more than 100 times. This old-established software engineering paradigm is entirely outdated, and should be replaced by a new revolutionary software engineering paradigm based on nonlinear thinking and complexity science[2].

3. The Transparent-Box testing method

The Transparent-Box testing method is graphically described in Fig. 2.

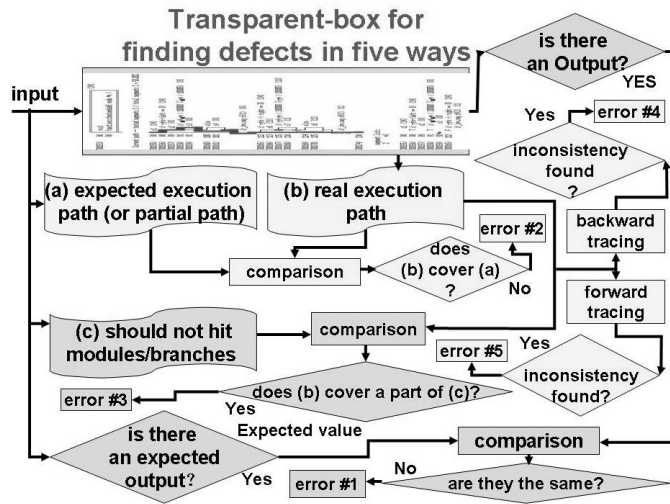


Fig. 2 Transparent-Box testing method

As shown in Fig. 2, with the Transparent-Box testing method, to each test case, the corresponding tool will not

only check whether the output (if any, can be none when it is dynamically used in the requirement development phase and design phase) is the same as what is expected, but also check whether the execution path covers the expected one specified in control flow, and whether the execution hits some modules or branches which are prohibited for the execution of the corresponding test case, so that it can be used to find functional defects, logic defects, and inconsistency defects. Having an output is no longer a condition to apply this method, so that it can be used dynamically in the entire software development lifecycle for defect prevention and defect propagation prevention.

The bidirectional traceability between test cases and the source code tested is established through the use of Time Tags to be automatically inserted into the description of the test cases and the database of the source code test coverage analysis for mapping them together accurately. Examples of Time Tags are automatically inserted into the description path of test cases shown in Fig. 3.

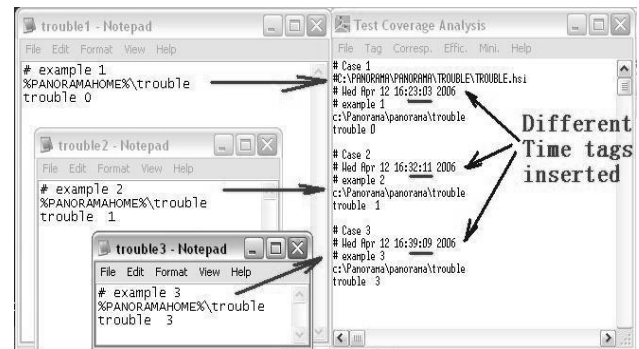


Fig. 3 Time Tag Examples

For extending the traceability to include the related documents, Some keywords such as @word@, @HTML@, @PDF@, @BAT@ are used for automatically opening the corresponding documents traced at a location specified by a bookmark.

The simple rules for designing a test case description are as follows:

- a '#' character at the beginning position of a line means a comment.
- an empty line means a separator between different test cases.
- Within comments, users can use some keywords such as @WORD@, @HTML@, @PDF@, and @BAT@ to indicate the format of a document, followed by the full path name of the document, and a bookmark.

- Within comments, users can use [path] and [/path] pair to indicate the expected path for a test case.
- Within comments, users can use Expected Output to indicate the expected value to be produced.
- Within comments, users can also use Not_Hit keyword to indicate modules or branches (segments) which are prohibited to enter for the related test case.
- After the comment part, there is a line to indicate the directory for running the corresponding program.
- The final line in a test case description is the command line (which may start a program with the GUI) and the options.

An sample test case script file with some test case descriptions is listed as follows (TestScript1) :

```
# test case 1 for New Order
#
#@HTML@
C:\Billing_and_Payment10\Requirement_s
pecification.htm#New_Order
#@WORD@
C:\Billing_and_Payment10\Prototype_des
ign.doc bmname New_Order
#@WORD@
C:\Billing_and_Payment10\TestRequireme
nts.doc bmname New_Order
#[path] main(int, char**) {s0, s1,
s9} [/path]
# Expected output : none
C:\Billing_and_Payment10
Billing_and_Payment.exe      new_order
Confirm

# test case 2 for Pay Invoice
#@HTML@
C:\Billing_and_Payment10\Requirement_s
pecification.htm#Pay_Invoice
#@WORD@
C:\Billing_and_Payment10\Prototype_des
ign.doc Pay_Invoice
#@BAT@
C:\isa_examples\ganttpro\ganttp9.bat
#[path] main(int, char**) {s1, s6, s9,
}B-Pay_Invoice(void) [/path]
# Expected output : none
C:\Billing_and_Payment10
Billing_and_Payment.exe Pay_Invoice

.....
```

4. The new software testing paradigm based on complexity science using the Transparent-Box testing method

Based on complexity science using the Transparent-box method, a new revolutionary software testing paradigm is established which offers comprehensive functions and capabilities for software testing, including the support not only for functional testing, but also for MC/DC (Modified Condition/Decision Coverage) test coverage analysis, memory leak and usage violation check, performance analysis, runtime error type analysis and the execution path tracing, GUI operation capture and selective playback, test case efficiency analysis and test case minimization for efficient regression testing after code modification, incremental unit testing and integration testing combined together seamlessly, semi-automatic test case design, and more.

Application examples of this new software testing paradigm in the requirement development phase for finding logic defects and inconsistent defects efficiently with the Holistic, Actor-Action and Event-Response Driven, Visual, Traceable, and Executable (HAETVE) software requirement development technique innovated by Jay Xiong to be used to replace the Use Case approach (which is not holistic, not suitable for event-response type applications, not traceable, and not directly executable for defect removal) are shown in Fig. 3 to Fig. 4.

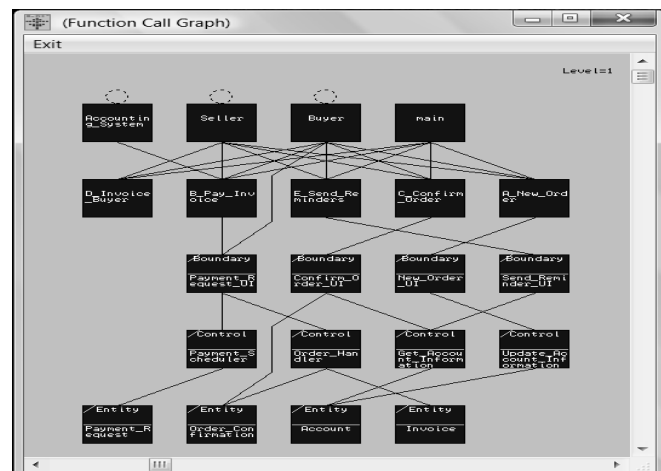


Fig. 4 An application result of the HAETVE technique for the function decomposition of the functional requirements of a Billing_and_Payment product through dummy programming using dummy modules (there are some function call statements in the body of a module (or an empty body) without real program logic)

The dummy programming source code of the main() module is listed as follows:

```
void main(int argc,char** argv)
{
```

```

int key;
if(argc==1 /* Missing a parameter */
  || argc > 2 /* Having an extra parameter */)
{
  cout << "Invalid Commands: \n" << argv;
}
else
{
  if(strcmp(argv[1],"New_Order")==0
  || strcmp(argv[1],"New_order")==0
  || strcmp(argv[1],"new_order")==0 )
  {
    A_New_Order();
    cout << "*** A_New_Order () called. ***\n";
  }
  else if (strcmp(argv[1],"Confirm_Order")==0 ||
  strcmp(argv[1],"Confirm_order")==0
  || strcmp(argv[1],"confirm_order")==0 )
  {
    C_Confirm_Order();
    cout << "*** C_Confirm_Order () called.
***\n";
  }
  else if (strcmp(argv[1],"Invoice_Buyer")==0 ||
  strcmp(argv[1],"Invoice_buyer")==0
  || strcmp(argv[1],"Invoice_buyer")==0 )
  {
    D_Invoice_Buyer();
    cout << "*** D_Invoice_Buyer() called. ***\n";
  }
  else if (strcmp(argv[1],"Pay_Invoice")==0 ||
  strcmp(argv[1],"Pay_invoice")==0
  || strcmp(argv[1],"pay_invoice")==0 )
  {
    B_Pay_Invoice();
    cout << "\n *** B_Pay_Invoice() called. ***\n";
  }
  else if (strcmp(argv[1],"Send_Reminders")==0 ||
  strcmp(argv[1],"Send_reminders")==0
  || strcmp(argv[1],"send_reminders")==0 )
  {
    E_Send_Reminders ();
    cout << "\n *** E_send_Reminders() called.
***\n";
  }
  else
    cout << "Invalid Commands: \n" << (char**)
argv << endl;

  cout << " *** Executed. *** \n" << (char**) argv
<< endl;
}
}

```

After the execution of the test script file, TestScript1, using this new software testing paradigm through the Panorama++ product, one logic defect and another inconsistency defect were found as shown in Fig. 5.

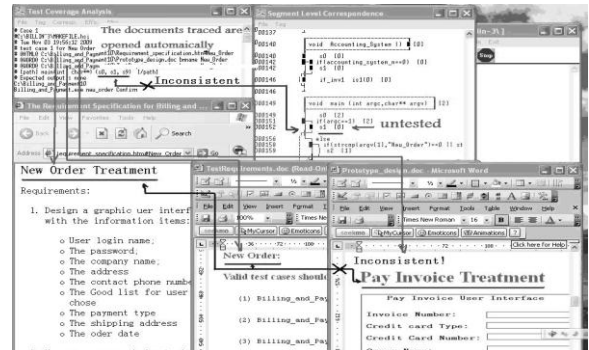


Fig. 5 Two defects found through dynamic testing using the Transparent-Box method when performing a forward tracing operation (Note: all the related documents are opened from the locations indicated by the corresponding bookmarks)

After checking the source code, we can easily find that there is a defect coming from an extra space character:

```

An extra space character is added:      |
                                         |
                                         v
if(argc==1 /* Missing a parameter */ /
  || argc > 2 /* Having an extra parameter
*/)
{
  cout << "Invalid Commands: \n" << argv;
}
else
{
  if(strcmp(argv[1],"New_Order")==0      ||
  strcmp(argv[1],"New_order")==0
  || strcmp(argv[1],"new_order")==0 )
  {
    A_New_Order ();
    cout << "*** A_New_Order () called.
***\n";
  }
}

```

After checking the bookmarks, we found that in the TestRequirements.doc file the bookmark Now_Oder is pointing to the Pay Invoice Treatment position rather than the New Order Treatment position. After removing the two defects, a correct result is obtained as shown in Fig. 6.

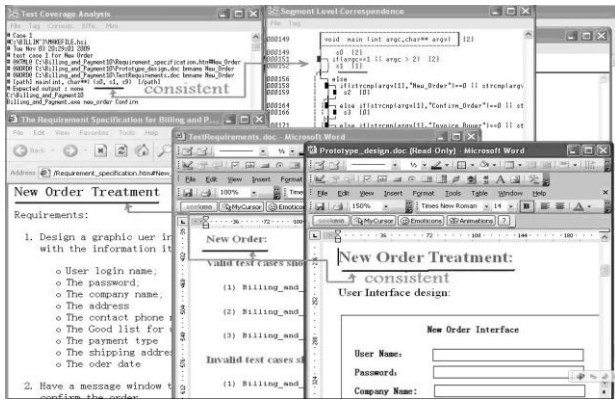


Fig. 6 After modification, the two defects shown in Fig. 5 are removed

When this new software testing paradigm is applied to test a software program without the source code, we can design a virtual main() to indicate the corresponding operations and call the program indirectly through dummy programming too. In this way the GUI operation can be captured and automatically play back after code modification with the capability to establish bidirectional traceability among the test cases, the test requirements, and user's manual, and other related documents even if the source code is not available.

5. The major features of the new software testing paradigm

The new presented software testing paradigm brings revolutionary changes to software testing. The major features of the new software testing paradigm include:

- It is based on the Transparent-Box testing method which combines functional testing and structural testing together seamlessly with close logic connection and a capability to automatically establish bidirectional traceability among the related documents and test cases and the corresponding source code tested, as shown from Fig. 3 to Fig. 5.
- It can be used in the entire software development lifecycle dynamically, from the requirement development phase down to the maintenance phase.
- It can be used to find functional defects, structural defects, inconsistency defects, memory leaks and memory usage violation defects, and performance bottlenecks.
- It supports MC/DC test coverage analysis required for the RTCA/DO-178B level A standard, being able to show the test coverage analysis results graphically with

untested branches and conditions highlighted as shown in Fig. 7.

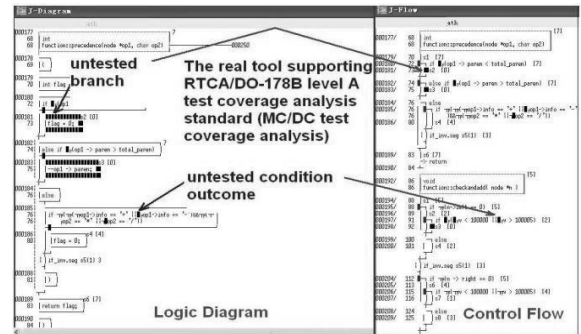


Fig. 6 MC/DC test coverage analysis and the analysis results shown graphically

- It supports memory leak analysis and memory usage violation check. An application example is shown in Fig. 87

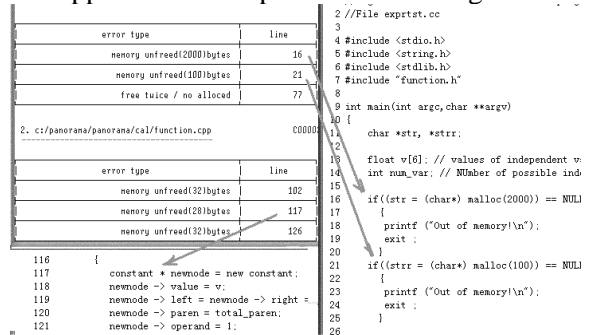


Fig. 7 A report on memory leak and usage violation check

- It supports performance analysis with the capability to report the branch execution frequency to locate performance bottlenecks better as shown in Fig. 8.

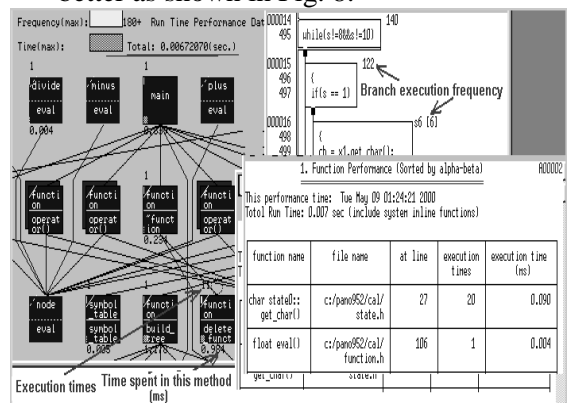


Fig. 8 An application example of performance analysis performed by Panorama++

- It supports efficient test case design by automatically choosing a typical path with

the most untested branches and automatically extracting the execution conditions of the chosen path as shown in Fig. 9.

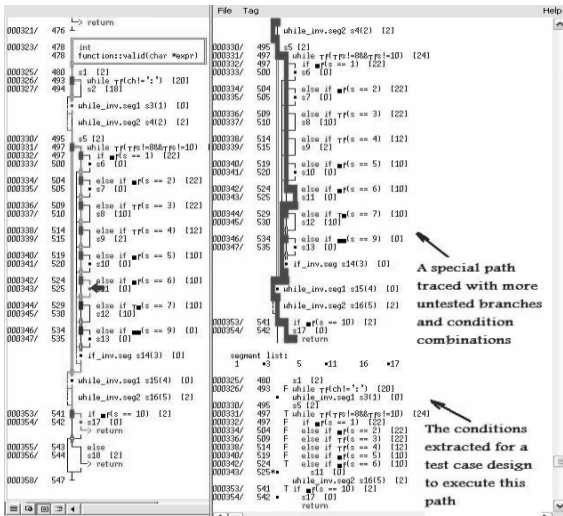


Fig. 9 Assisted test case design performed by Panorama++

- It supports embedded software testing too, as shown in Fig. 10.

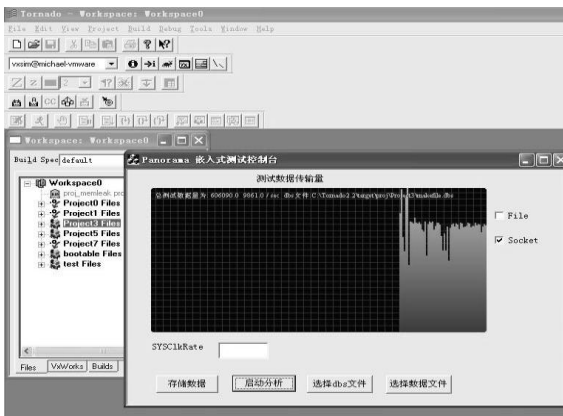


Fig. 10 An application example shows that the MC/DC test coverage data are sent from the target to the test server

6. A general comparison between the new software testing paradigm and the old one

(a) The defect finding efficiency

The old testing paradigm used for incremental software development is shown in Fig. 11[3].

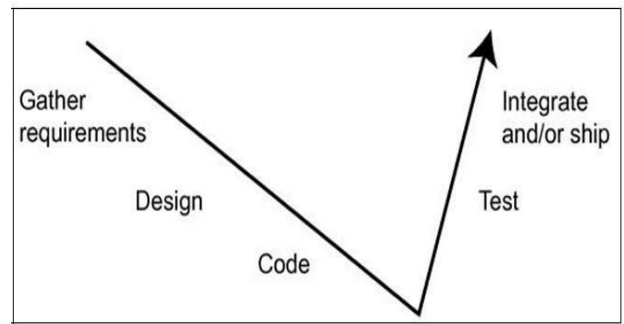


Fig. 11 Traditional software testing performed with incremental software development

The old testing paradigm used for the iterative software development is shown in Fig. 12.

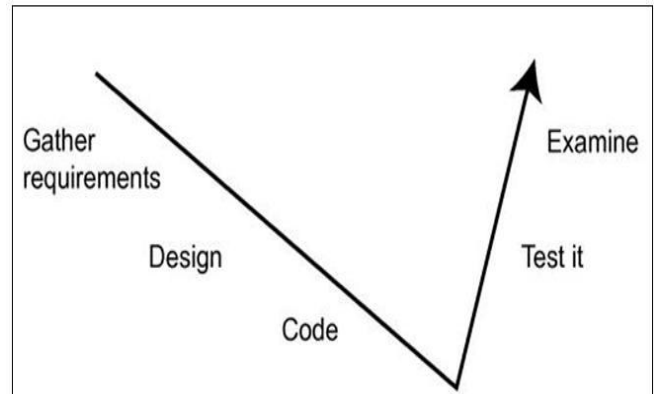


Fig. 12 The old testing paradigm used for the iterative software development

The presented new software testing paradigm used for incremental or iterative software development is shown in Fig. 13.

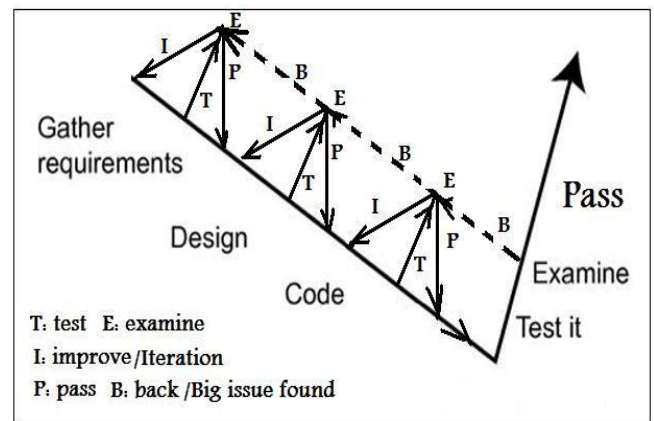


Fig. 13 The presented new software testing paradigm used for incremental or iterative software development

Comparing Fig. 11, Fig. 12, and Fig. 13, it is clear that the new software testing paradigm is

much more efficient in finding defects in a software product development process.

(b) The timing in finding the defects

The traditional software testing methods can be performed after coding, but it is too late; in comparison, the new presented software testing paradigm can be used in all phases of a software development lifecycle, including the requirement development phase and the design phase.

(c) The defect types that can be found

The traditional black-box method can be used to find functional defects; the traditional structural white-box method can be used to find some structural defects for the Existing product no matter it is the customer-required product or not.

The presented new software testing paradigm can be used to find functional defects, structural defects, logic defects, and inconsistency defects.

Some functional defects can not be found by the black-box method, but can be found by the new software testing paradigm as shown in Fig. 14.

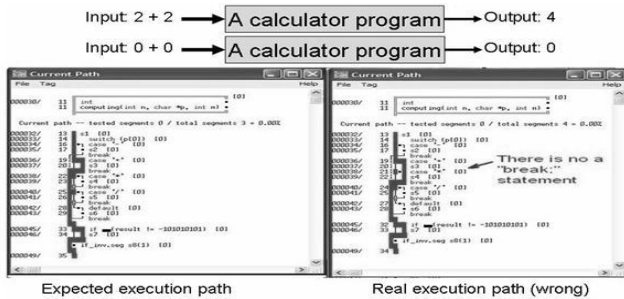


Fig. 14 An application example of transparent-box testing: a bug found even if the output is the same as what is expected (this defect comes from that a “break” statement is missing, so that the result 4 is produced through 2 times 2 rather than 2 plus 2)

(d) The graphical representation techniques for displaying the test results

The test results obtained from the applications of most traditional software testing methods and tools are shown in textual formats or value tables.

But the test results obtained from the applications of the presented new software testing paradigm is graphically shown in the

system-level and in the detailed source code level as shown in Fig. 15.

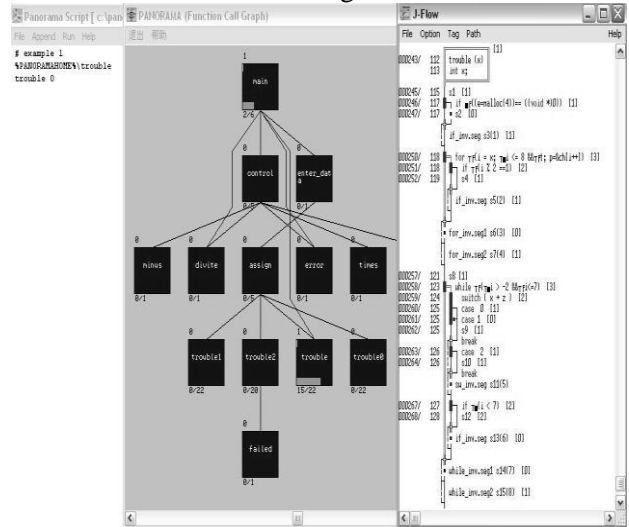


Fig. 15 An example of test coverage analysis result obtained using the presented new software testing paradigm (the untested branches and conditions are highlighted with small black boxes)

(e) The capability to support automated traceability

It is only supported by the presented new software testing paradigm.

Conclusion

This paper presented a new software testing paradigm based on the Transparent-Box testing method which brings revolutionary changes to software testing in the 21st century by combining structural testing and functional testing together seamlessly with internal logic connections, which can be used dynamically in the entire software development lifecycle from requirement development down to maintenance.

References

[1] Deming W E. *Out of the Crisis*. MIT Press, 1982.
 [2] Jay Xiong, Tutorial, *A Complete Revolution in Software Engineering Based on Complexity Science*, WORLDCOMP'09, Las Vegas, July 13-17, 2009.
 [3] Alistair Cockburn, *Using Both Incremental and Iterative Development*, *CrossTalk*, May 2008 Issue

Software Quality Assurance Revolution Based on Complexity Science - An Introduction to NSE-SQA

WanYuan Huang¹, Linda Li²

¹The Jumpulse Center of Research and Incubation of Northwestern Polytechnic University

²ISA Shanghai, China

Abstract -The old-established software quality assurance paradigm is an outcome of linear thinking, reductionism, and the superposition principle that the whole of a complex software system is the sum of its components, so that with it almost all software quality assurance activities are performed linearly, partially, and locally through inspection and testing after production – violating Deming's product quality principle of "Cease dependence on inspection to achieve quality. Eliminate the need for inspection on a mass basis by building quality into the product in the first place."

This paper describes a new software quality assurance paradigm based on complexity science by complying with the essential principles of complexity science, particularly the Nonlinearity Principle and the Holism Principle, so that with it almost all software quality assurance activities are performed nonlinearly, holistically, and globally through defect prevention and defect propagation prevention performed with dynamic testing using the innovated Transparent-box testing method and semi-automatic inspection supported by bi-directional traceability in the entire software development lifecycle from the first step down to the retirement of a software product.

Keywords: software quality assurance, defect prevention, software testing, software maintenance, complexity science

1 Introduction

Today software has become the driving force for the development of all kinds of businesses, engineering, sciences, and the global economy. As pointed by David Rice, "like cement, software is everywhere in modern civilization. Software is in your mobile phone, on your home computer, in cars, airplanes, hospitals, businesses, public utilities, financial systems, and national defense systems." But unfortunately, software itself is not well engineered. For instance, the reliability of today's cloud computing software is too low to be accepted - only in 2011 many cloud computing systems failures were reported (Tim Perdue, 2011), including the following listed ones caused mainly by software problems:

- Sony's [Playstation Network](#) (4/21/2011)
- [Amazon Web Services](#) (4/21/2011)
- [Twitter Service](#) (2/25/2011)
- [Netflix Streaming Service](#) (3/22/2011)
- [Intuit Service and Quickbooks](#) (3/28/2011)

NIST (National Institute of Standards and Technology) concluded that "Briefly, experience in testing software and systems has shown that testing to high degrees of security and reliability is from a practical perspective not possible. Thus, one needs to build security, reliability, and other aspects into the system design itself and perform a security fault analysis on the implementation of the design."

This paper introduces a new software quality assurance paradigm based on complexity science, called NSE-SQA with which software quality is ensured mainly through defect prevention and defect propagation prevention supported by various bi-directional traceability established dynamically using the innovated Transparent-box testing method.

2 What Does a Revolution Mean?

According to "The Structure of Scientific Revolutions" (Kuhn T, 1962), science does not progress continuously, by gradually extending an established paradigm. It proceed as a series of revolutionary upheavals. **A revolution means a drastic, complete, and fundamental change of paradigm to resolve some outstanding and generally recognized problem that can be met in no other way.**

Kuhn described that there are three phases with Scientific Revolutions: the first phase, which exists only once, is the **pre-paradigm phase**, in which there is no consensus on any particular theory, though the research being carried out can be considered scientific in nature – this phase is characterized by several incompatible and incomplete theories; the second phase is the **normal science** – if the actors in the pre-paradigm community eventually gravitate to one of these conceptual frameworks and ultimately to a widespread consensus on the appropriate choice of to increased insights, then the **normal science** begins, in which puzzles are solved within the context of the dominant paradigm. As long as there is general consensus within the discipline, normal science continues; the third phase is the **revolutionary science** phase – over time, progress in normal science may reveal anomalies, facts which are difficult to explain within the context of the existing paradigm. While usually these anomalies are resolved, in some cases they may accumulate to the point where normal science becomes difficult and where weaknesses in the old paradigm are revealed; Kuhn refers to this as a crisis. After significant efforts of normal science within a paradigm fail, science may enter the third phase, that of **revolutionary science**, in which the underlying

assumptions of the field are reexamined and a new paradigm is established. After the new paradigm's dominance is established, scientists return to normal science, solving puzzles within the new paradigm. A science may go through these three phases cycles repeatedly, though Kuhn notes that it is a good thing for science that such paradigm shifts do not occur often or easily.

3 Bringing drastic, complete, and fundamental changes to Software Quality Assurance Foundation

3.1 The foundation of the Existing Software Quality Assurance paradigm is wrong

Software and software engineering paradigms are complex systems where a small change may bring big impact to an entire software product – «Butterfly-Effects». But the existing software quality assurance paradigm is based on linear thinking, reductionism, and the superposition principle that the whole of a nonlinear system is the sum of its parts, so that with it almost all software quality assurance activities are performed linearly, partially, and locally.

3.2 The corresponding software modeling approaches with the old-established software quality assurance paradigm are outdated

The existing software modeling approaches are outdated because they are outcomes of the reductionism and superposition principles, using different sources for human understanding and computer understanding of a software system separately with a big gap between them (see Fig.1). The obtained models are not traceable for static defect removal, not executable for debugging, not testable for dynamic defect removal, not consistent with the source code after code modification, and not qualified as the road map for software development.

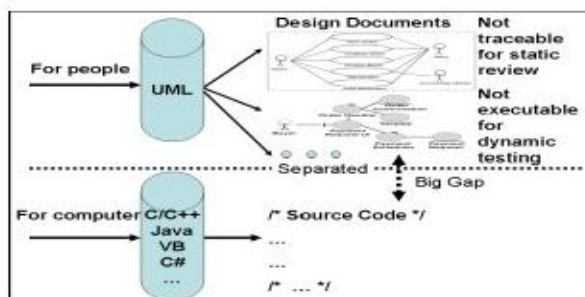


Fig. 1 The Two Source Approach for software modeling (TSA)

3.3 The corresponding software process models are wrong

All existing software process models are linear ones.

3.4 The corresponding software development methods are outdated

components are developed first, then the system of a software product is built through the integration of the components developed. From the point of view of quality assurance, those methodologies are test-driven but the functional testing is performed after coding - it is too late. These methodologies handle a software product as a machine rather than a logical product created by people. They all comply with the superposition principle. With those methodologies, all activities are performed linearly, partially, and locally too.



Fig. 3 Software development methods based on constructive holism

3.5 The corresponding software testing methods with the old-established software quality assurance paradigm are outdated

The current software testing paradigm is mainly based on functional testing plus structural testing, load testing, and stress testing, being performed after coding. It is too late. The functional testing approach using the Black-box method cannot be performed in the requirement development phase and the design phase dynamically, so that there is no way to find defects introduced in the requirement development phase and the design phase dynamically using the existing software testing paradigm.

3.6 The corresponding software maintenance paradigm with the old-established software quality assurance paradigm is wrong

The existing software maintenance paradigm offers a blind, partial, and local approach for software maintenance without support of various traceabilities. There is no way to prevent the side-effects of the implementation of requirement changes or code modifications. Local and partial software maintenance is risky - each time when a bug is fixed, there is a 20%-50% of chance of introducing another into the software product. It is why today, software maintenance takes more than 75% of the total effort and total cost for software product development.

3.7 The existing visualization paradigm, documentation paradigm, and project management paradigm are also outdated Conclusion:

those issues show that only improving the quality assurance process, the visualization, and the management process without making revolutionary changes to the

foundation of software engineering, the software modeling approaches, the software development methodologies, the software testing methods, and the software maintenance paradigm, will be possible to greatly improve the quality and reliability of a software product.

4. The Solution Offered

The solution provided is called New Software Quality Assurance Paradigm Based on Complexity Science. It consists of two major steps: (1) Making revolutionary changes to all the major components of the software engineering paradigm from that based on linear thinking, reductionism, and the superposition principle (so that with it almost all software quality assurance activities are performed linearly, partially, and locally) to that based on nonlinear thinking and complexity science (so that with it almost all software quality assurance activities are performed nonlinearly, holistically, and globally); (2) After the revolutionary changes done to all the major components of the software engineering paradigm, making the desired characteristics and behaviors of the whole software quality assurance paradigm emerge from the interactions of the all major software engineering components – for instance, making all the components of the new software engineering paradigm work together to ensure the quality of software maintenance through side-effect prevention in the implementation of software changes supported by various traceabilities automatically established by the new software testing paradigm, new software visualization paradigm, new documentation paradigm, and new software maintenance paradigm.

As shown in Fig. 4, with the existing software quality assurance paradigm, improving the quality of a software product will reduce the productivity or increase the cost.



Fig. 4 The relationship among quality, productivity, cost, and risk with today's software development

Fig. 5 shows the objectives of the solution offered – making it possible to help software development organizations double their productivity, halve their cost, and improve their product quality tenfold many times, compared with the existing approaches.

NSE's Objectives

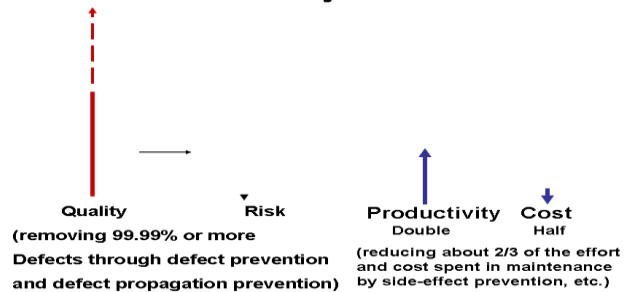


Fig. 5 The objective of the NSE-SQA solution

4.1 Foundation of the solution

This solution is based on complexity science by complying with the essential principles of complexity science, particularly the nonlinearity principle and the holism principle that the whole of a complex system is greater than the sum of its components, and that the characteristics and behaviors of the whole emerge from the interaction of its components, so that with it almost all of the tasks and activities in software quality assurance are performed nonlinearly, holistically, and globally.

4.2 Dynamic Software Modeling driven by source code (DSM)

The basic idea of DSM and the major differences between TSA and DSM is shown in Fig. 1 (TSA) and Fig. 6 (DSM). DSM is the key technique to ensure software quality in software requirement development and software design.

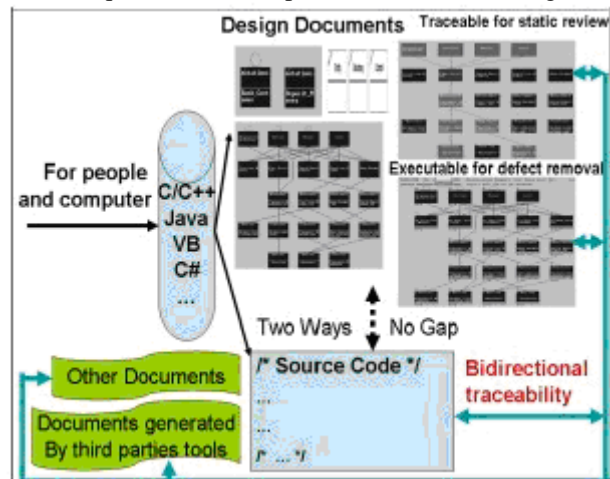


Fig. 6 Dynamic Software Modeling driven by source Code

As shown in Fig. 6, with DSM, one kind of source is used for both human understanding and computer understanding of a software product. The models/diagrams are automatically generated from the source code, either a dummy program using dummy modules having an empty body or only a set of function call statements, or a regular program through reverse engineering. The generated diagram and the source code are traceable.

With TSA, there is an one-time design process that complying with the linear process models (either a one time waterfall model, or an iterative/incremental model which is

“a series of Waterfalls”[4]) without upstream movement at all – the designers have no right to be wrong. But we are human beings rather than God – people are nonlinear and easily make mistakes in thinking, reading, writing, hearing, making wrong decisions, etc.

With DSM, we have the right to be wrong in the design, but we also have the right to be right – in the coding phase, if we find something wrong with the product design, then we correct them with coding – we can easily update the design by rebuilding the database for automatically generating all related design documents/diagrams making design become pre-coding, and coding become further design (top-down plus bottom-up).

HAETVE technique:

With DSM, three type of interactive and traceable diagrams (J-Chart, J-Diagram, and J-Flow innovated by Jay Xiong) can be automatically generated at any phase from the source code of a dummy program (in forward engineering) or regular program (in reverse engineering).

J-Chart notations are shown in Fig. 7.

Function	Non-member function
	Press the right mouse button to pop up a function menu.
class	Member function
Function	Press the right mouse button to pop up a member function menu.
Macro	Macro function
	Press the right mouse button to pop up a function menu.
Function	Overloading non-member function
	Press the right mouse button to pop up an overloading menu.
Class	Overloading member function and virtual function
Function	Press the right mouse button to pop up a function menu.
Class	Overloading member function
Function	Press the right mouse button to pop up a function menu
Class	Virtual function
Function	Press the right mouse button to pop up a function menu
Class	Class
	Press the right mouse button to pop up a class menu.
Class	Template Class
	Press the right mouse button to pop up a class menu.

Fig. 7 J-Chart notations

HAETVE means Holistic, Actor-Action and Event-Response driven, Traceable, Visual, and Executable technique for dynamic requirement modeling. With HAETVE the graphical notations for representing an actor and an action for C/C++ programs are shown in Fig. 8 where the notation used for representing an actor is originally designed for representing a recursive program module.

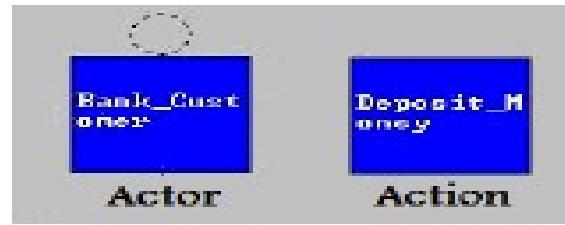


Fig. 8 Notations for representing actor and action for C/C++

The corresponding dummy source code written in C/C++ is listed as follows separately:

```
Bank_Customer ()
{
    Bank_Customer ();
}
Void Deposit_Money ()
{
}
```

For the Actor-Action type applications, HAETVE is similar to the Use Case approach, and is easy to map to Use Case notations as shown in Fig. 9 and Fig. 10.

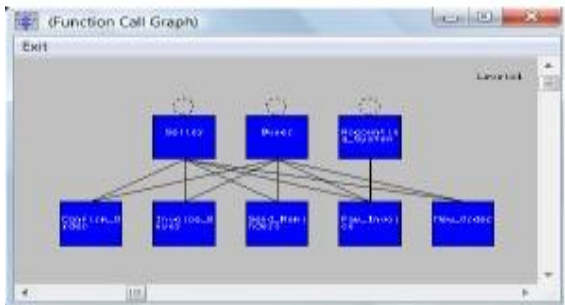
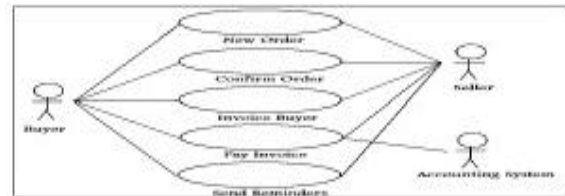


Fig. 10 Notation mapping between Use Cases (Top) and HAETVE (Bottom)

The analysis result of Use Cases can also be mapped to HAETVE as shown in Fig. 11.

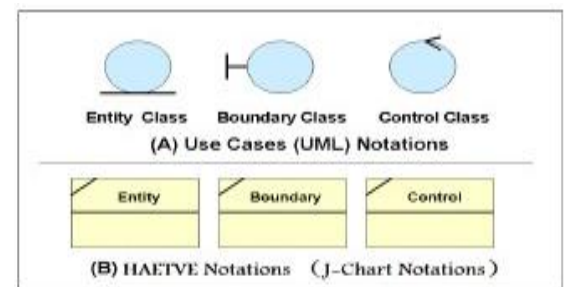


Fig. 11 Analysis notation mapping between Use Cases (UML) and HAETVE

But there are some special things with HAETVE:

- (a) The obtained results are traceable for static defect removal - see Fig. 12 – found and fixed a defect through traceability: the Order_Handler should handle Order_Confirmation too.

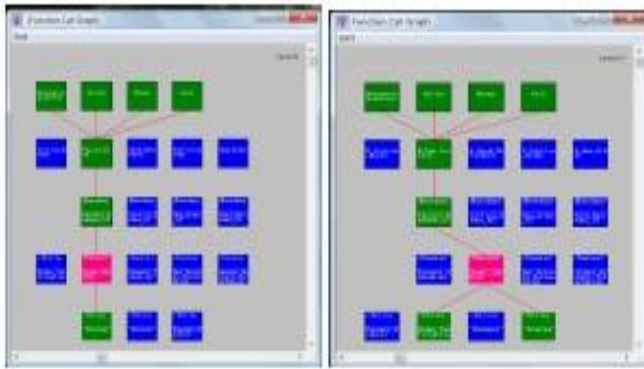


Fig. 12 Defect removal through review with traceability

- (b) The obtained results are executable for dynamic defect removal using the Transparent-box testing method innovated by me which not only checks whether the output (if any, can be none) is the same as what is expected, but also helps users to check whether the execution path covers the expected one, and establish automated traceability among related documents and test cases and source code using Time Tags to map test cases with the source code tested (see Fig. 13), keywords to indicate the document formats, and bookmarks to open the traced documents (see Fig. 14).

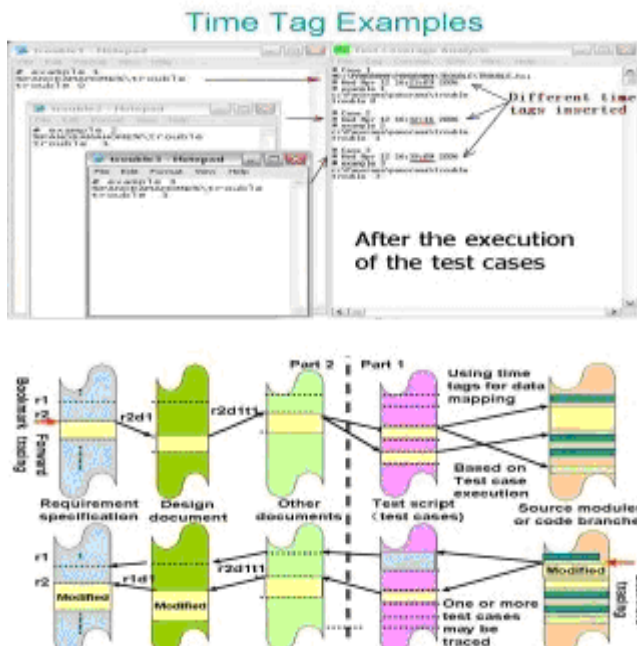


Fig. 14 Self-maintainable facility for bi-directional traceability

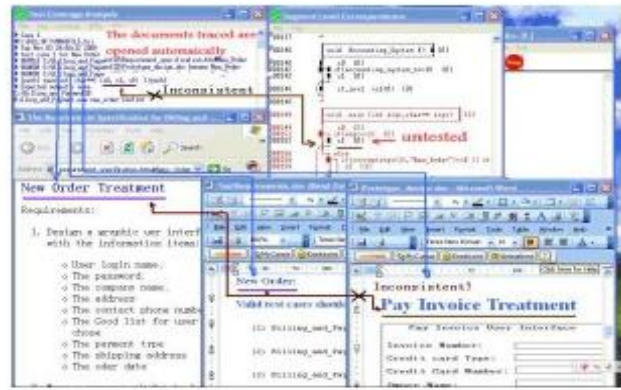


Fig. 15 Two defects found: the execution path didn't cover the expected one; one bookmark used is wrong.



Fig. 16 New result: after the two defects removed

4,3 NSE process model

The NSE process model is nonlinear, consisting of the pre-process part and the main process part. The detailed process steps are shown in Fig.17.

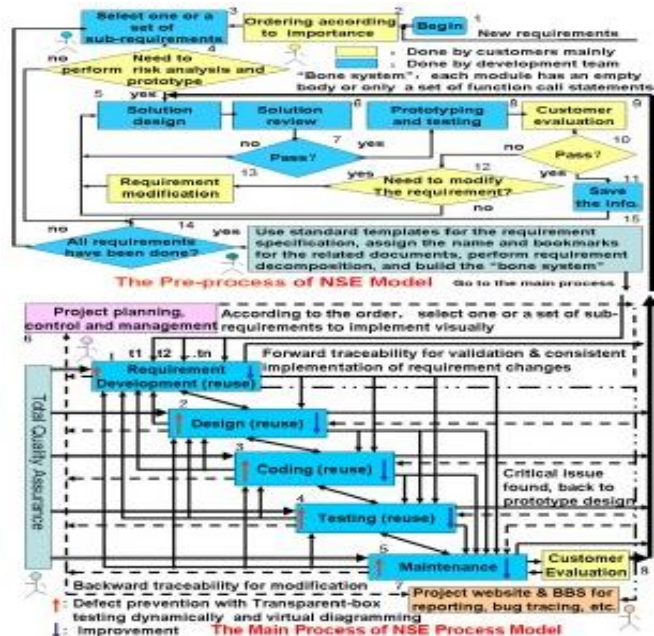


Fig. 17 NSE process model

Fig. 15 shows two defects are found; Fig. 16 shows the correct result after the two defects are removed.

4.4 NSE software development methodology

Fig. 18 shows that the NSE software development method is based on Generative Holism.

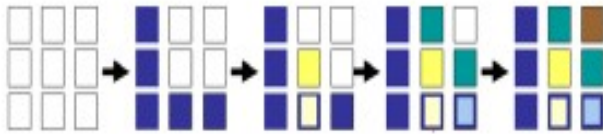


Fig. 18 Generative Holism based software development

Fig. 19 shows that the NSE software development methodology is driven by defect prevention and defect propagation prevention.

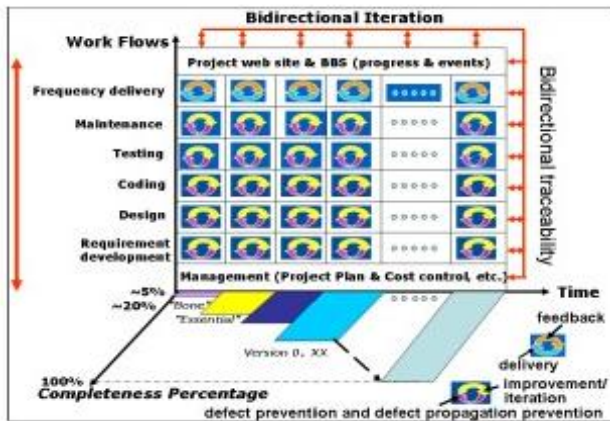


Fig. 19 NSE software development methodology

Fig. 20 shows that with assigned bottom-up coding orders, inconsistent defects in the interface design can be prevented.

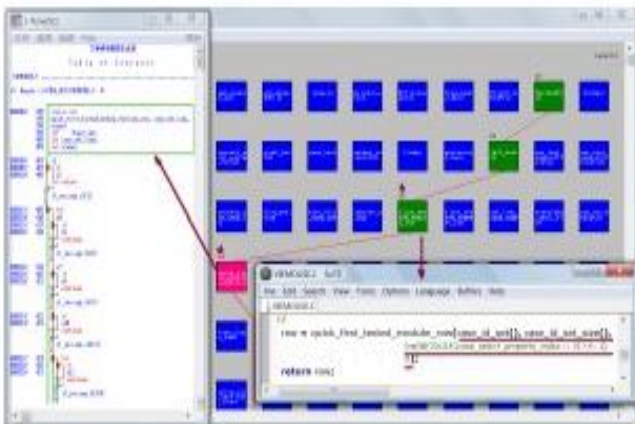


Fig. 20 An example for defect prevention

4.5 The innovated Transparent-box method for software testing

See Fig. 21, where the Transparent-box testing method combines functional testing and structural testing together – to each test case it checks whether the output (if any, can be none) is the same as what is expected, but also helps users

check whether the code execution path covers the expected one.

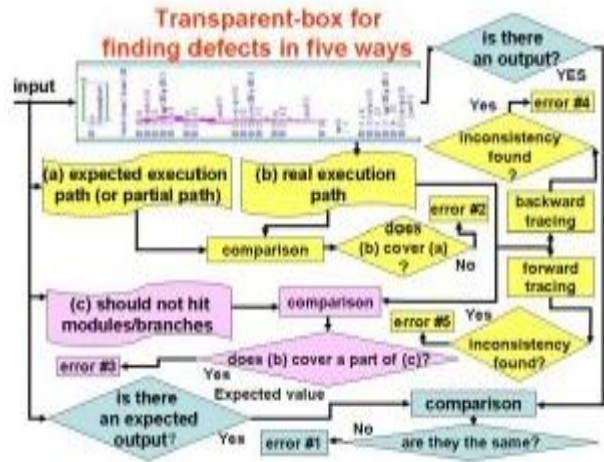


Fig. 21 The Transparent-box testing method

Fig. 22 shows the comparison result of the defect detection efficiency.

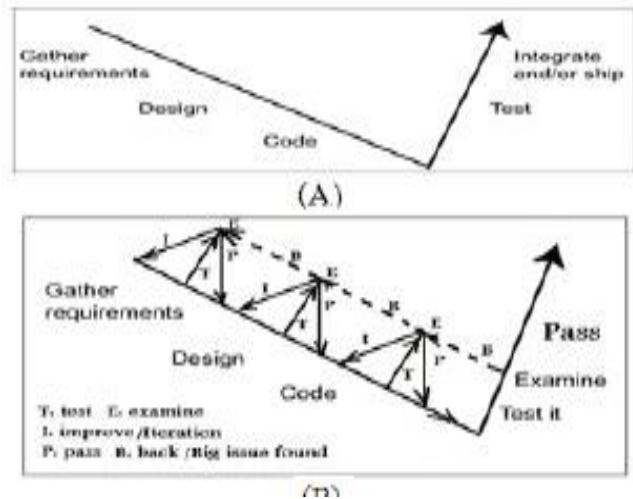


Fig. 22 Defect detecting efficiency: (A) traditional approaches; (B) Transparent-box testing method

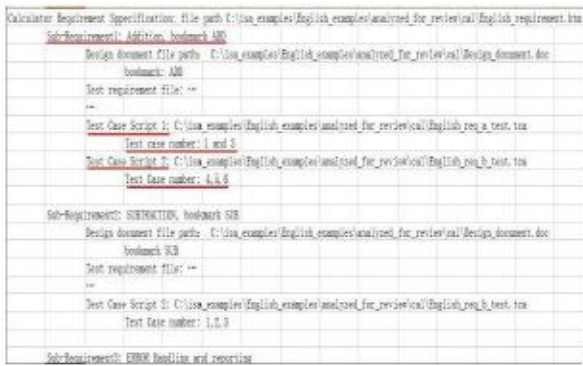
Fig. 23 shows the supported capability for Modified Condition/Decision Coverage analysis.

```

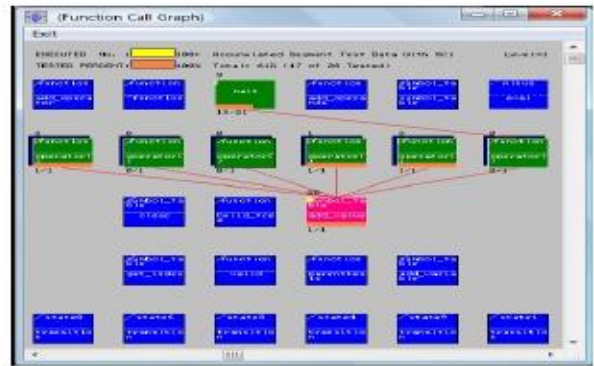
int
transit (unsigned char ch)
{
    < untested condition
    if (T{Tch== '4'} || Tch== '1' || Tch== '2' || Tch== '3')
    {
        return 7;
    }
}
    
```

Execution times: 000337s, 10, 62 18

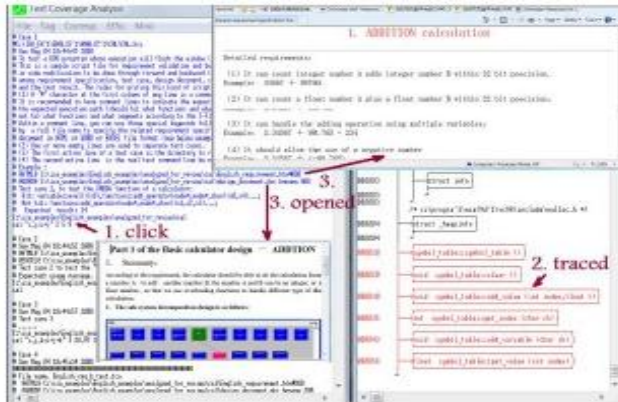
Fig. 23 MC/DC test coverage analysis supported



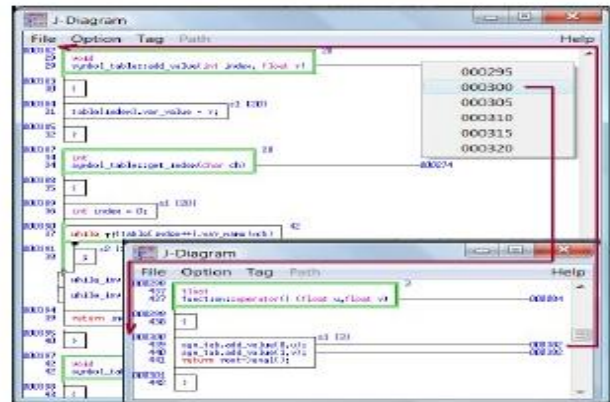
(A)



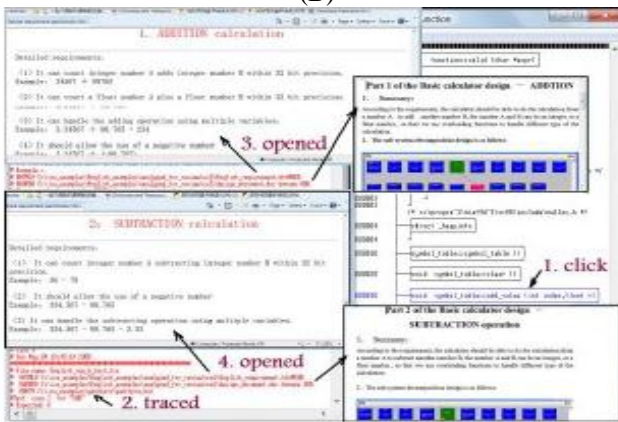
(D)



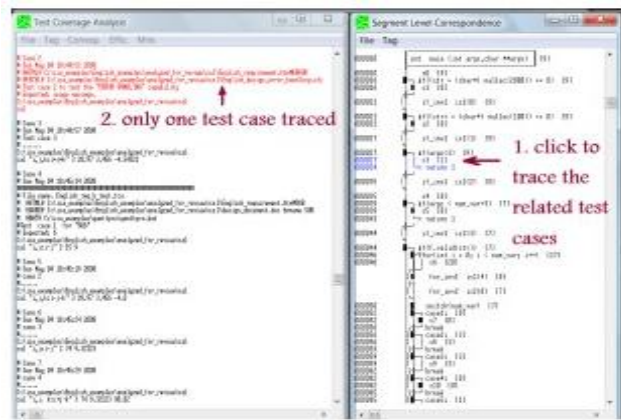
(B)



(E)



(C)



(F)

Fig. 24 Implementation of a requirement change with side-effect prevention: (A) From the requirement to be changed to find the related test cases through the document hierarchy description table; (B) Perform forward tracing from the test case(s) to find the related program modules; (C) Perform backward tracing from the module(s) to be modified to find how many requirements are related (in this case, two requirements are related, so that the modification must satisfy both); (D) Tracing the module to be modified to find how many other modules are related (which may need to be modified too) from the corresponding call graph; (E) Check the consistency between a modified module and all the statements calling it using the logic diagram automatically generated from the source code with traceability; (F) Tracing a modified source code segment (a set of statements with the same execution conditions) or a modified module to find the corresponding test case(s) which can be used to re-test it efficiently.

4.6 NSE Software Maintenance paradigm

As shown in Fig. 24, with the NSE Software Maintenance paradigm software maintenance is performed holistically and show in Fig. 25.

globally with side-effect prevention through various traceabilities.

The corresponding software maintenance process model is shown in Fig. 25.

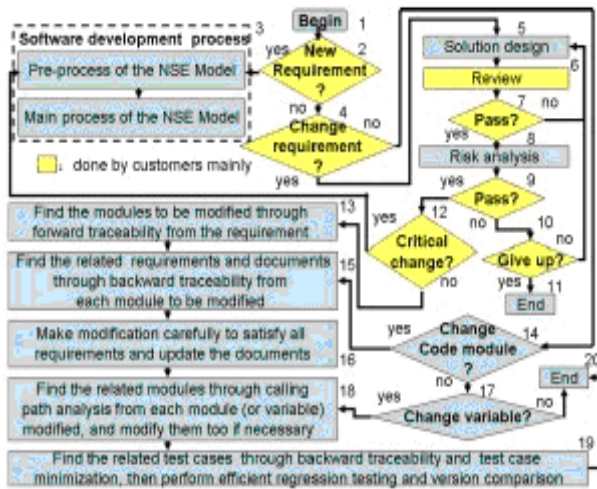


Fig. 25 The NSE software maintenance process model
 As shown in Fig. 26, with the new software maintenance paradigm, it is possible to save about 2/3 of the effort and cost spent in software maintenance because (1) most defects introduced into a software product in the requirement development phase and the design phase can be removed through dynamic modeling driven by source code; (2) the entire software development process are driven by defect prevention and defect propagation prevention; (3) There is no major difference between the software development process and the software maintenance process – both support changes with side-effect prevention through various traceabilities automatically established.

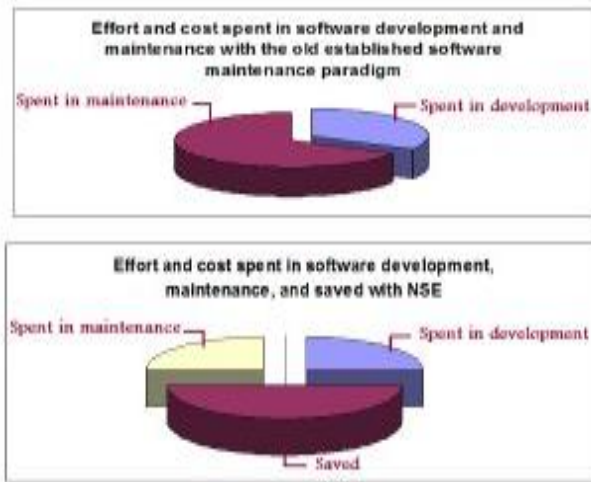


Fig. 26 Possible effort and cost savings

5. The Applications

With the new revolutionary paradigm for software quality assurance (NSE-SQA), it is possible to remove 99.99% of the defects in a software product - see Table 1. Note: the item and the data written in italics come from the published reports provided by Software Productivity Research based on the analysis of 12000 software projects [5].

Table 1 Defect Removal Efficiency

	Defect Removal Technology	Highest Defect Removal Efficiency
1	<i>Requirement review</i> Requirement review with traceable documents	50%
2	<i>Top level design review</i> Top level design review using traceable documents and charts	60%
3	<i>Detailed functional design review</i> Detailed functional design review using traceable documents	65%
4	<i>Detailed logic design review</i> Detailed logic design review using traceable diagrams	75%
5	<i>Code inspection</i> Code inspection with bi-directional traceability	85%
6	<i>Unit testing</i> Unit testing incrementally according to the assigned bottom-up testing order plus MC/DC test coverage analysis capability and graphical representation of the test result	90%
7	<i>New function testing</i> New function testing	65%
8	<i>Integration testing</i> Integration testing incrementally	60%
9	<i>System testing</i> System testing combining structural and function testing seamlessly	65%
10	<i>External beta testing</i> External beta testing with traceable documents and the source code	75%
11	<i>Cumulative Efficiency</i> Cumulative Efficiency with Defect prevention in the entire software development life cycle	99.99%

5. Conclusion

This paper introduced a new software quality assurance paradigm based on complexity science (NSE_SQA). With it, almost all software quality assurance activities are performed non-linearly, holistically, and globally through defect prevention and defect propagation prevention in the software development lifecycle.

References

- [1] David Rice, GEEKONOMICS The Real Cost of Insecure Software, 1E, Pearson Education,inc, Publishing as Addison Wesley, 2008
- [2] Jones, Capers, Social and Technical Reasons for Software Project Failures, CrossTalk, Jun 2006 Issue
- [3] Alistair Cockburn, Using Both Incremental and Iterative Development, CrossTalk, May 2008 Issue
- [4] Condensed GSAM Handbook, chapter 2, CrossTalk, 200
- [5] Jones, Capers, SOFTWARE QUALITY IN 2002: A SURVEY OF THE STATE OF THE ART, <http://www.SPR.com> July 23, 200

Software Traceability Establishment Revolution Based on Complexity Science

Po-Kang Chen¹, Jay Xiong²

¹Y&D Information system, Inc. USA

²International Software Automation, Inc. (ISA, currently being reorganized), USA

Abstract - *software becomes system of complexity in the modern time. 40 years passing, modern software design tends to complexity and precision. Until 2012, software design tool doesn't support excellent model's and function's traceability during design program. On the other hand, software design won't adapt on modern software design, which needs a methodology based on complex science for its maintenance and design. Traceability is a important factor influencing quality of software. This paper presents automated, dynamic, accurate, precise, and self-maintainable traceability among related software documents and test cases and source code, established through test case execution and some keywords used within the test case descriptions to indicate the format of the documents as well as the file paths and the bookmarks for automatically opening the documents from the corresponding positions when the related test case is selected for forward tracing or traced from the corresponding source code backwardly. When a test case is executed a Time Tag will be automatically inserted into both the test case description and the database of the test coverage measurement results for mapping them together. No matter if the contents of a document is modified, or the parameters of a test case are changed, or the corresponding source code is modified, after rerunning the test case the traceability will be updated automatically without any manual rework. Here a "document" means a regular file for requirement specification, design description, test requirement specification, user manual, project development plan, cost report, or a web page as well as a batch file for dynamically running a related program such as a tool for selectively playing back the GUI operations captured with the test case execution, and displaying the test coverage measurement result shown in a new type control flow diagram which is interactive and traceable with untested source modules and branches highlighted at the same time for automated software acceptance testing. Above all, it will bring drastic, complete, and fundamental change of paradigm, resolving some outstanding and generally recognized problems. No other way can efficiently resolve those outstanding and generally recognized problems.*

Keywords: : software traceability, requirement traceability, validation, verification, testing, quality assurance , maintenance

1 Introduction

Software is a nonlinear complex system where a small change can ripple through the entire system to cause

major unintended impacts – “Butterfly-Effects”, so that prior to performing the actual change, maintainers need facilities in order to understand and estimate how a change will affect the rest of the system. Traceability offers benefits to organizations in the areas of project management, process visibility, requirement validation and verification, and software maintenance. Traceability needs to be hardcoded into a process to be replicated iteratively on each and every project[1]. Without bidirectional traceabilities software maintenance can not be performed globally and holistically to prevent side-effects. Local and blind software changes will make the software product unstable and unlierable.

1.1 The problems addressed

The lack of traceability among software documents, test cases, test results, and source code is caused by several factors, including: (1) the fact that these artifacts are written in different languages (natural language vs. programming language); (2) they describe a software system at various abstraction levels (requirements vs. implementation); (3) processes applied within an organization do not enforce maintenance of existing traceability links; (4) a lack of adequate tool support to create and maintain traceability[2] ; (5) there are many different types of documents, some of which are created manually, some of which are generated automatically by internal tools, some of which are generated automatically by third parties' tools, some of which are designed using graphic editors; (6) some documents are stored locally, some documents are stored in other places through a network; (7) some related documents are web pages, which can be read through the internet only; (8) some documents are related to the software development, while some documents are related to the project management which should also be traceable; and (9) some documents are not static materials, must be viewed dynamically through a program execution. Unfortunately, neither manual traceability methods nor existing COTS traceability tools available on the market are adequate for the current needs of the software engineering industry. Poor methods and tool support are perhaps the biggest challenge to the implementation of traceability - when those tools are used, the traceability information is not always maintained, nor can it always be trusted to be up-to-date and accurate. [1]. Studies have shown that existing commercial traceability tools provide only simplistic support for traceability [3]. Why does software maintenance take 75%

or more of the total effort and total budget[4] in most software project development? One of the critical issues is the lack of bidirectional traceabilities among the requirement specification, the design documents, the test cases, the test results, and the source code of a software product.

1.2 The solution

The new requirement traceability approach proposed and implemented by the authors is graphically shown in Fig. 1.

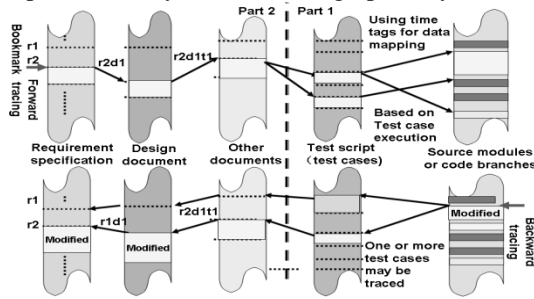


Figure 1. The facility for automated, bi-directional, and self-maintainable traceability among the documents and the test cases and the source code of a software product

- The objectives of this traceability facility are:
- Helping software developers to prevent side-effects in the implementation of software changes;
- Solving the inconsistency issue to make the documents and the source code traceable with each other to keep consistency;
- Removing the problems existing with a man-made Requirement-Traceability-Matrix which is inaccurate, time consuming to do, and almost un-maintainable;
- Making the software development process visible;
- Making the requirement validation and verification much easy to perform;
- Making the software product much easy to understand, test, and maintain.

As shown in Figure 1, this facility for bidirectional traceability consists of two parts:

(1) Part 1

Part 1 of the facility is related to the traceability between test cases and the corresponding source code executed by running the test cases. It is done with the use of Time Tags which are automatically inserted into both the test case descriptions and the corresponding test coverage database. For instance, if test case 1 is executed at 09:00 AM on September 2, 2009, and test case 2 is executed at 10:00 AM on the same day, and test case 3 is executed at 11:00 AM on the same day, then the three different Time Tags will be inserted into the three test cases and the corresponding test coverage database separately. So, when test case 2 is selected for forward tracing, the Time tag of 10:00 AM on September 2, 2009 will be taken from the test case description to search the test coverage data with the same

time tag, so the corresponding test coverage data will be read and the corresponding modules and branches will be highlighted on a control flow diagram. On the other hand, when a module or code segment shown on a control flow diagram is selected, the related time tags (which can be more than one) used to indicate what time the module or segment was executed will be taken to search the test case descriptions to see how many test cases with the mapping time tags through backward tracing, then highlights all test cases mapped on the window showing the test case script.

(2) Part 2

Part 2 of the facility is to extend the bi-directional traceability from test cases and the source code to include all related documents, the test cases, and the source code. It is done using some key words (written into the comment part of the description of a test case) such as @WORD@, @HTML@, @BAT@, @PDF@, and @EXCEL@ followed with the corresponding file path and a bookmark to indicate the format of the document, the full path name of the file, and the corresponding location in the document, so that when a test case is selected for forward tracing, or traced from a module or segment backwardly, the corresponding document will be opened and shown from the location indicated by the bookmark.

It is recommended to organize the requirement specification and the related documents hierarchically (even if some documents have not been really designed) with inherited (or meaningful) bookmarks as shown in table 1.

Table 1 Document Hierarchy.

Project Name		Project Code	
Project Description			
The full path name of the Project feasibility report		Version number	
The full path name of the requirement specification		Version number	
Requirement 1	Bookmark	r1	
Description			
The full path name of the related design document		Bookmark	
Description		r1d1	
The full path name of the related test specification		Bookmark	
Description		r1d1t1	
Requirement 2	Bookmark	r2	
...			

It is important to make the document hierarchy include the test case scripts (test cases numbers) so that when a requirement needs to be changed or selected for validation, it is easy to find what test cases to be used.

The major steps for establishing and applying the bidirectional traceability are as follows:

- Step 1: Organize the requirement specification and the related documents hierarchically with the bookmarks, clearly indicate each requirement and the corresponding test scripts and the test case numbers;
- Step 2: Design the test case scripts with the corresponding keywords to indicate the formats and the file paths and the bookmarks for the related documents;
- Step 3: Perform code instrumentation for test coverage analysis to the entire program;

Step 4: Compile the program instrumented;
 Step 5: Execute the test case scripts with the corresponding tool.
 Step 6: Show the modified test case script files with time tags inserted in a window;
 Step 7: Show the program test coverage measurement result using a control flow diagram in another window;
 Step 8: Perform forward tracing from a test case with a tool to map and highlight the corresponding modules and code branches tested by the test case through the inserted time tag – at the same time, open the related documents according to the document formats, file paths, as well as the bookmarks (or run the corresponding batch file if a @BAT@ keyword is used);
 Step 9: Perform backward tracing from a program module or code branch with a tool to map and highlight the related test cases through the inserted time tags - at the same time, open the related documents according to the document formats, file paths, as well as the bookmarks (or run the corresponding batch file if a @BAT@ keyword is used);
 Step 10 : After the implementation of code modifications, go to step 3.
 Step 11: If a related document is modified in the contents only without changing the bookmarks, there is nothing to do; but if the bookmarks are modified (such as the name of a bookmark is changed), modify the corresponding test case scripts according to the new bookmarks, then go to step 5;
 Step 12: if only the test cases are modified, go to step 5;
 Step 13: if the source code is modified, go to step 3;
 Step 14: If it is the time to perform requirement validation and verification (V&V), use the document hierarchy information organized in step 1 to get each requirement and the corresponding test cases to perform forward tracing one by one to see whether the requirement is completely implemented;
 Step 15: if a requirement is needed to modify: (1) get the test cases related to this requirement to perform forward tracing to locate the documents needed to update, and the source modules or branches needed to modify; (2) perform backward tracing from those modules or branches to see whether more requirements are related – if it is related to more requirements, the implementation of the code modification must satisfy all of the related requirements to avoid requirement conflict.
 Step 16: if it is the time to perform regression testing after modification, get the modules or branches modified to perform backward tracing to collect the corresponding test cases which can be used to re-test the modified program efficiently. Sometimes, there may be a need to add new test cases.

2 The major features

2.1 Automated

This facility works automatically with the capability to insert the Time Tags into both the test cases description part (see Fig. 2) and the database of the program test coverage measurement result, and highlight the test cases selected on the corresponding test script window, and the source code modules/branches shown in a control flow diagram on the corresponding source code window, or vice versa, as well as open the related documents traced from the locations pointed by the bookmarks.

and headers (final page numbers and running heads will be inserted by the publisher). Select a standard size paper such as A4 (210 X 297 mm) or letter (8.5 X 11 in) when preparing your manuscript.

2.2 Self-maintainable

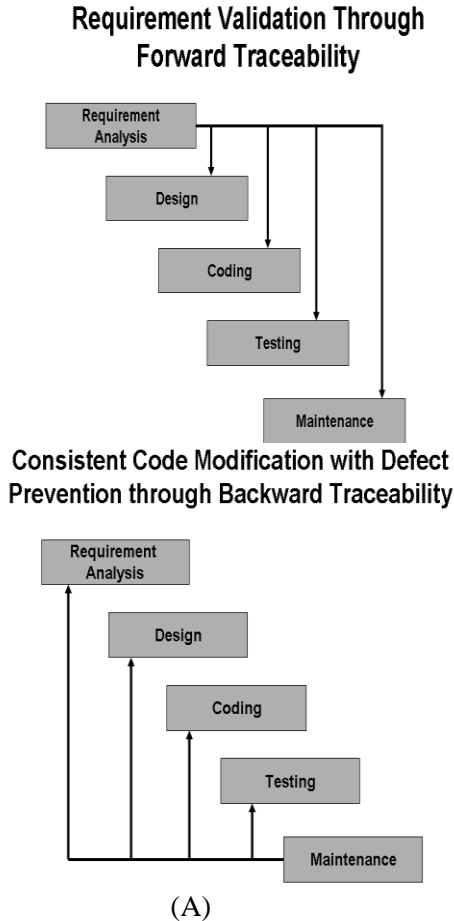
This facility is self-maintainable no matter if the contents of a document is modified, the parameters of a test case is modified, or the source code is modified - after rerunning the test case scripts, the traceability will be automatically updated without manual rework.

2.3 Methodology-independent

This facility is methodology-independent no matter which methodology or process models are used to develop the product.

2.4 Nonlinear, bidirectional, and parallel

This facility works in a nonlinear, bidirectional, and parallel style as shown in Figure 3 and Figure 4. For example, when a design defect is found after the product delivery, the developers can perform forward tracing to check the related requirement, and backward tracing to find and fix the related source code, etc. as shown in Figure 5.



(B)
Figure 3 Supporting parallel work: (A) application in requirement validation through forward traceability; (B) application for defect prevention in code modification

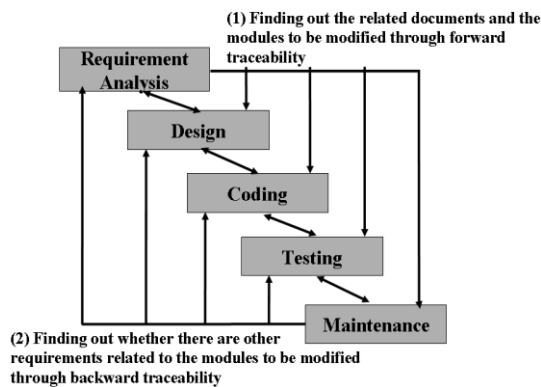


Figure 4 Safe implementation of a requirement change with side-effect prevention

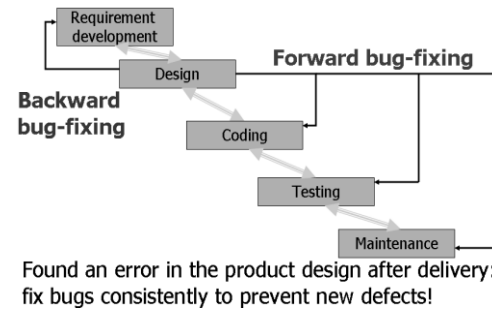


Figure 5 Fixing a design defect through forward and backward traceability

2.5 Accurate

This facility is based on the dynamic execution of the test cases and test coverage measurement and the time tags to map the test cases and the source code tested, so that it is accurate. After code modification or parameter changes of the test cases, we can re-run the test cases to automatically update the facility.

2.6 Precise

This facility is precise to the highest level – up to the code statement/segment (a set of statements to be executed with the same conditions) level bi-directionally. It is particularly useful for side-effect prevention in software maintenance.

2.7 Extended to include software project management documents

This facility is extended to include not only the software development documents, but also include the project management documents such as the product development schedule charts, the cost estimation reports, and so on to combine the software development process and the software management process together. If a project management document (such as a gantt chart) is designed using a third party's tool, a corresponding batch file should be designed and used with the @BAT@ keyword to indicate the location of the batch file in the test case description part such as the following example:

@BAT@ C:\isa_examples\ganttpro\ganttpro.bat

2.8 Extended to include web pages

For supporting web-based software development and applications, this facility is extended to include web pages to be traced and automatically opened through the use of @HTML@ keyword to indicate the URL address and the bookmark (#NAME) such as the following example:

- [1] Andrew Kannenberg, et al. Why Software Requirements Traceability Remains a Challenge, CrossTalk, Jul/Aug 2009 Issue.
- [2] Juergen Rilling , et al. CASCON 2007 Workshop Report ,Traceability in Software Engineering - Past, Present and Future ,IBM Technical Report: TR-74-211 October 25, 2007
- [3] Ramesh, Balasubramaniam, and Matthias Jarke. Toward Reference Models for Requirements Traceability. IEEE Transactions on Software Engineering 1 (2001): 58-93.
- [4] Ambler S W. A Manager's Introduction to The Rational Unified Process (RUP), Ambysoft. 2005
- [5] Jay Xiong, Tutorial, A Complete Revolution in Software Engineering Based on Complexity Science, WORLDCOMP'09 - , Las Vegas, July 13-17, 2009.
- [6] Jay Xiong, Jonathan Xiong, A Complete Revolution in Software Engineering Based on Complexity Science, WORLDCOMP'09 – SERP (Software Engineering Research and Practice 2009) , 109-115.

Software Visualization Revolution Based on Complexity Science - An Introduction to NSE Software Visualization Paradigm

Po-Kang Chen¹, Jay Xiong²

¹Y&D Information system, Inc. USA

²NSEsoftware, LLC., USA

Abstract

This article presents a component of the Nonlinear Software Engineering paradigm (NSE) – the NSE software visualization paradigm, with which the automatically generated charts and diagrams are colorful, interactive, holistic, virtual, dynamic, and traceable. It is innovated for making the entire software development life-cycle and the obtained work products visible. "One picture is worth ten thousand words." (Chinese idioms) – a holistic, interactive, colorful, visual, dynamic, and traceable chart/diagram will be more useful in the description of a complex software product. NSE software visualization paradigm makes a software product much easier to develop, understand, test, and maintain.

Keywords: Software Visualization, Software Diagramming, Call Graph, Logic Diagram, Control Flow, Tool, NSE

1. introduction — the old-established software engineering visualization paradigm is outdated

The old-established software engineering visualization paradigm is outdated because it is:

- a. **based on reductionism, and superposition principle that the whole of a system is the sum of its parts** - so that almost all diagramming tasks and activities are performed locally and partially.
- b. **not Holistic** – often the application results obtained consist of many small pieces without a complete chart/diagram to show an entire complex software product.
- c. **not automated in most cases** - most charts/Diagrams are created using graphic editors, not automatically generated.
- d. **not interactive** - most charts/diagrams generated are not interactive, hard to manipulate.
- e. **not traceable** - even if a complete chart/diagram for an entire software product can be obtained using a few diagramming tools, it is still useless because that without traceability and the capability to highlight an element with all of the related elements, there are too many connection lines to make the chart/diagram hard to view and hard to understand.
- f. **not accurate** - often when the source code is modified, the generated charts and diagrams can not be automatically updated to keep consistency with the source code.
- g. **not precise** - for instance, when a logic diagram is used to show the result of program test coverage measurement, it can not show whether an invisible "else" part (a "if" statement without an explicit "else" part) is tested or not. Almost all existing visualization tools can not graphically show whether a condition in a decision statement is tested or not when applied to

show the result of MC/DC (Modified Condition/Decision Coverage) test coverage measurement results.

- h. **often not consistent with the source code** - the charts/diagrams obtained are often not consistent with the source code after software modification.
- i. **not consistent among all related charts and diagrams** - often they are created/generated with different formats using different information, hard to keep consistency among them.
- j. **not virtual** - the charts and diagrams obtained are stored in hard copies or XML or Postscript format in the memory and/or hard disk, requiring much more spaces to store and long loading time to display.
- k. **not complete** -the traditional software engineering visualization techniques and tools do not integrated together to efficiently support the following visualizations:
 - visualization of the entire software engineering lifecycle
 - visualization for software inspection
 - visualization for software testing
 - visualization for software maintenance
 - visualization for the source code of an entire software product
 - visualization of dynamic program behavior
 - visualization for software debugging

2. The Revolutionary Solution Offered by NSE Visualization Paradigm

The revolutionary solution offered by NSE[1] for software visualization will be described in details in this paper late. Here is the outline of the solution:

- a. **Based on nonlinear thinking and complexity science**
- b. **Holistic**
- c. **Automatic**

- d. Interactive
- e. Traceable
- f. Accurate
- g. Precise
- h. Consistent among all related charts and diagrams
- i. Linkable automatically between different charts and diagrams
- j. Virtual
- k. Complete in software engineering visualization, including
 - visualization of the entire software engineering lifecycle
 - visualization for requirements engineering
 - visualization for design engineering
 - visualization for coding engineering
 - visualization for software inspection
 - visualization for software testing
 - visualization for software maintenance
 - visualization for software verification/validation
 - visualization for software architectures
 - visualization for the source code of an entire software product
 - visualization for reverse engineering
 - visualization of dynamic program behavior
 - visualization for software debugging

display incremental unit test order or the related test coverage and quality data using bar graphics overlaid on each object-box (module-box) to help users view the overall results of testing and quality measurement. J-Chart is useful in system design, understanding, inspection, test planning, test result display, and re-engineering. The J-Chart notations are shown in Fig. 1.

A comparison between J-Chart and the most traditional call graphs

	J-Chart	Traditional Call Graph
Is it holistic for directly showing a very complex software product?	Yes	No
Is it interactive for highlighting a path or getting related information?	Yes	No
Is it traceable to highlight a module with the all related modules?	Yes	No *1
Is it supported to use a module as the root to generate a sub-chart?	Yes	No
Can a bar-chart be added to a module-box to show related information?	Yes	No
Can the source code be directly edited from a module-box?	Yes	No
Can the logic diagram be linked from a module-box?	Yes	No
Can the control flow diagram be linked from a module-box?	Yes	No
Can a bottom-up coding orders be assigned to the modules?	Yes	No
When used for software version comparison, can different colors be used to show "un-changed" modules, "changed modules", "deleted modules", and "added new modules" separately?	Yes	No

3. The Foundation for Establishing NSE Visualization Paradigm

NSE software visualization paradigm is established through FDS (the Five-Dimensional Structure Synthesis method - a paradigm-shift framework innovated by me) by complying with the essential principles of complexity science, particularly the Nonlinear principle and the Holism principle – with NSE software visualization paradigm, almost all diagramming tasks/activities are performed holistically and globally to make the entire software product and the entire software development process visible.

4. 3J graphics (J-Chart, J-Diagram, and J-Flow)

The 3J graphics (J-Chart – a new type call graph, J-Diagram – a new type logic diagram, and J-Flow – a new type control flow diagram) are innovated by me and implemented by me and my colleagues. J-Chart/J-Diagram/J-Flow is a trinity: an Object-Oriented and structured chart/logic diagram/Control flow diagram, the chart/diagram generator which is always running when the chart/diagram is shown, and the interface (using the chart/diagram itself) between the generator and the user for controlling the chart/diagram dynamically.

J-Chart

J-Chart not only can be used to represent the class inheritance relationship, the function call graph, and the class-function coupling structure graphically, but can also be used to

J-Diagram

J-Diagram not only can be automatically generated from source code in all levels including the class hierarchy tree, class structure diagram, and the class member function logic diagram with un-executed class/function/segments/conditions highlighted, but also can be automatically linked together for an entire software product to make the diagrammed source code traceable in all levels. J-Diagram can be automatically converted into J-Flow diagram. J-Diagram is particularly useful in Object-Oriented software understanding, inspections, walkthroughs, testing, and maintenance.

J-Diagram notations are shown in Fig. 2. Interactive and

*1: Some tools claim that they can provide dynamic function call graph, but I have not seen their application examples provided.

traceable J-Diagram not only makes a software product much easier to read, understand, test, and maintain, but also makes the code inspection and walk through much easier to perform in a semi-automated way (see Fig. 5 B1).

The major differences between J-Diagram and most Flow Charts

	J-Diagram	Flow Charts
Is it structured?	Yes	No
Can it show an entire software product very complex?	Yes	No
Is it uniqueness?	Yes	No
Is the location of the program logic indicated?	Yes	No
Can it show the result of test coverage measurement?	Yes	No
Can it show the branch execution frequency?	Yes	No
Does it offer traceability between related elements?	Yes	No
Can it be converted to a control-flow diagram?	Yes	No
Does it exist virtually without huge space to store?	Yes	No

Interactive and traceable J-Diagram not only makes a software product much easier to read, understand, test, and maintain, but also makes the code inspection and walk through much easier to perform in a semi-automated way.

J-Flow

Most traditional control flow diagrams are un-structured. They often use the same notation to represent different program logic, and cannot display the logic conditions and the source code locations. J-Flow diagram, on the other hand, is Object-Oriented and structured, uses different notations to represent different logics with capability to show logic execution conditions and the corresponding source code locations. J-Flow is particularly useful in logic debugging, path analysis, test case and code correspondence analysis, and class/function level test coverage result display with unexecuted elements (path, segments, and unexecuted condition outcomes) highlighted.

The notations of J-Flow diagram are shown in Fig. 3. Interactive and traceable are the important features of J-Flow diagram, particularly useful for software testing.

The major differences between J-Flow and traditional control flow diagram:

	J-Flow	Traditional Control Flow
Is it structured?	Yes	No
Can it show an entire software product very complex?	Yes	No
Is it uniqueness?	Yes	No (arbitrary)
Is the source code locations of the control flow indicated?	Yes	No
Can it show the result of test coverage measurement?	Yes	No
Can it show the branch execution frequency?	Yes	No
Can it be automatically converted to a logic diagram?	Yes	No
Can it highlight a path with most untested elements?	Yes	No
Does it exist virtually without huge space to store?	Yes	No

Interactive and traceable are the important features of J-Flow diagram, particularly useful for software testing.

5. Applications

NSE Software Visualization Paradigm has been successfully applied in the entire software development process and the maintenance process for a software product development.

Figure 4 shows some application examples of J-Chart, J-Diagram, and J-flow separately.

Figure 5 shows some application examples combining J-Chart and J-Diagram together, J-Chart and J-Flow diagram together, etc.

6. Conclusion

NSE software visualization paradigm is based on complexity science, complying with the Nonlinearity principle and the Holism principle, so that with NSE almost all visualization tasks/activities are performed holistically and globally to automatically generate virtual, interactive, colorful, and traceable 3J graphics (J-Chart, J-Diagram, and J-Flow) innovated to make the entire software development process and the obtained work products visible. NSE software visualization paradigm makes a software product much easier to design, understand, test, and maintain.

References

[1] Jay Xiong, Jonathan Xiong, *A Complete Revolution in Software Engineering Based on Complexity Science*, WORLDCOMP'09 – SERP (Software Engineering Research and Practice 2009), 109-115.
 [2] Brooks, Frederick P. Jr., "The Mythical Man-Month", Addison Wesley, 1995, P249.


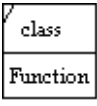


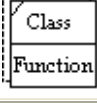
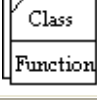
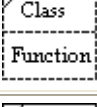


	Non-member function Press the right mouse button to pop up a function menu.
	Member function Press the right mouse button to pop up a member function menu.
	Macro function Press the right mouse button to pop up a function menu.
	Overloading non-member function Press the right mouse button to pop up an overloading menu.
	Overloading member function and virtual function Press the right mouse button to pop up a function menu.
	Overloading member function Press the right mouse button to pop up a function menu
	Virtual function Press the right mouse button to pop up a function menu
	Class Press the right mouse button to pop up a class menu.
	Template Class Press the right mouse button to pop up a class menu.

Fig. 1 J-Chart notations

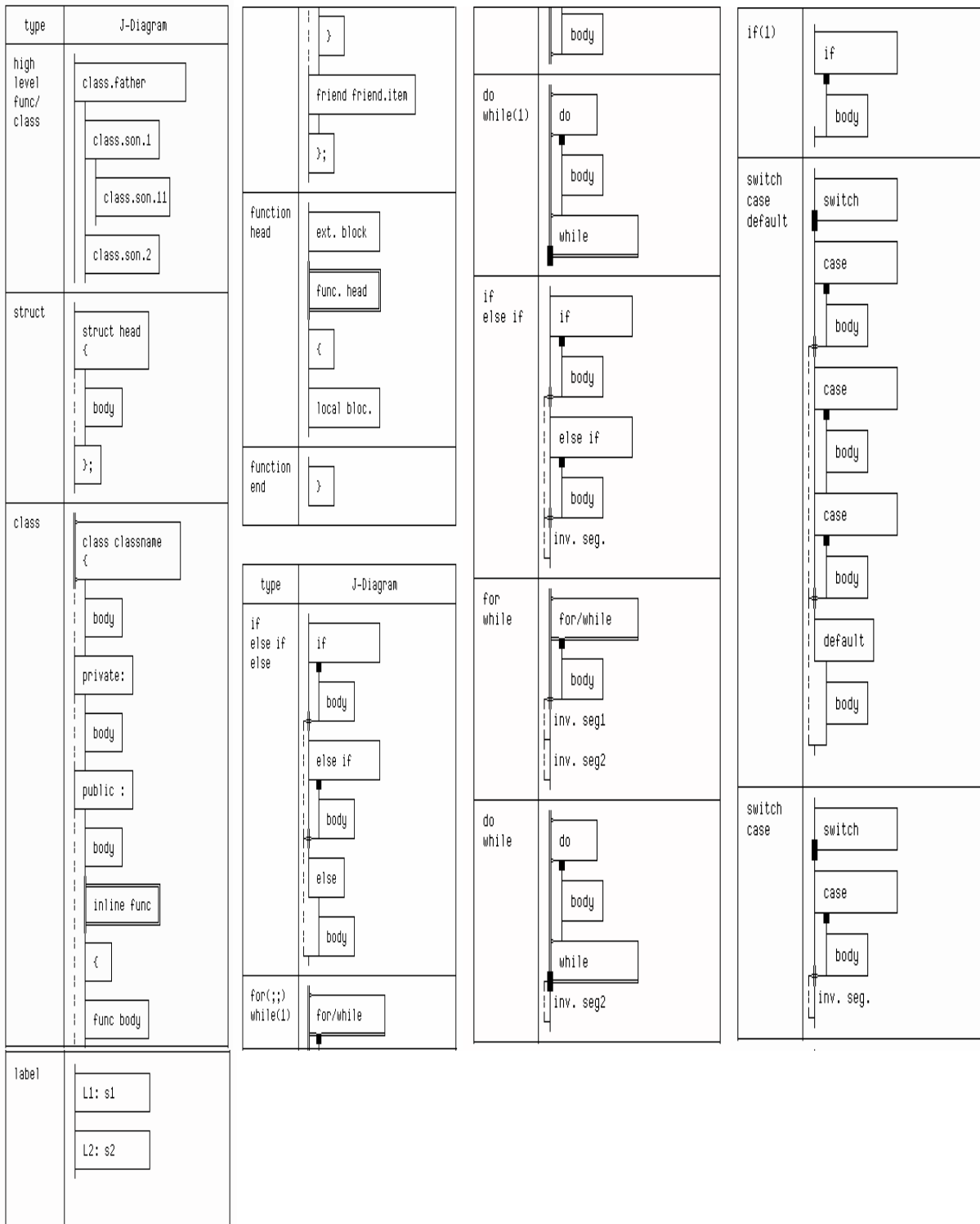
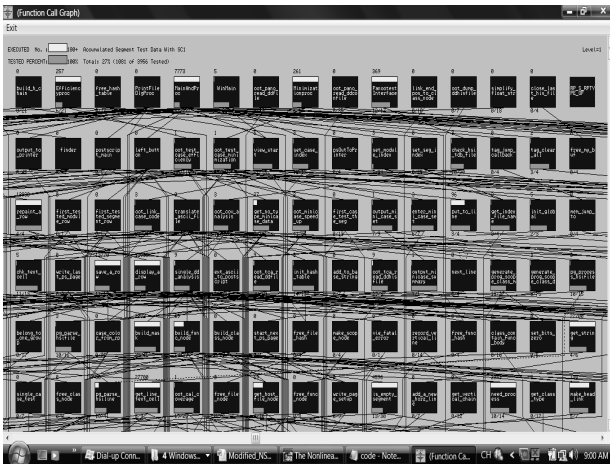
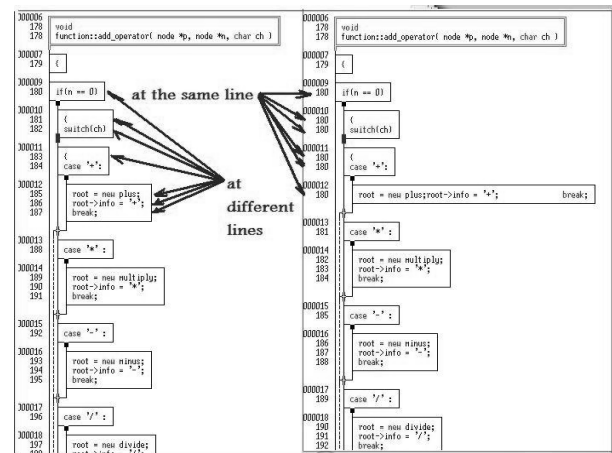


Fig. 2 J-Diagram notations



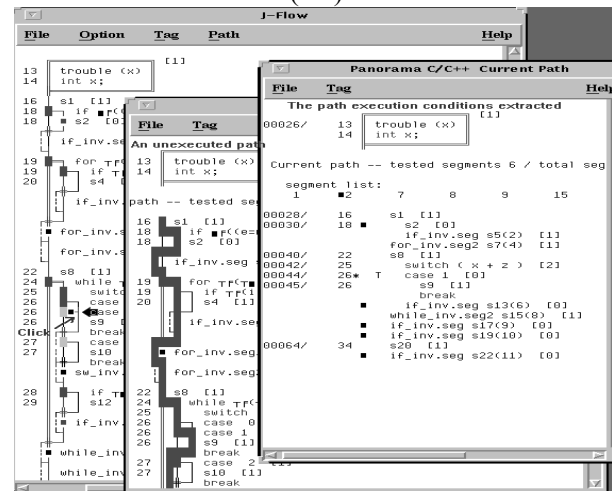
(A1)



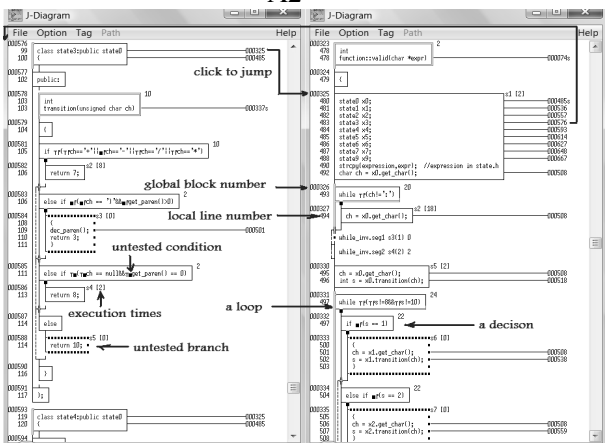
(B2)



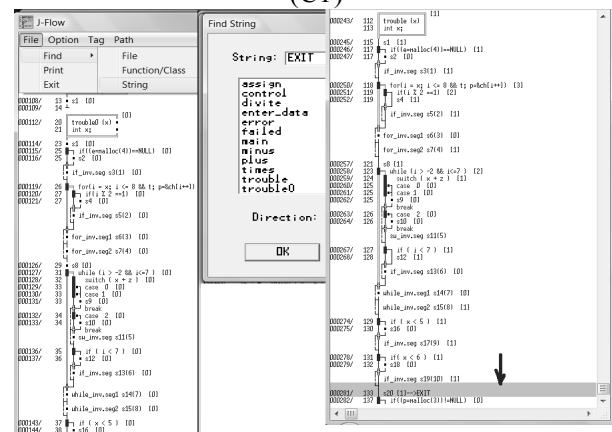
A2



(C1)

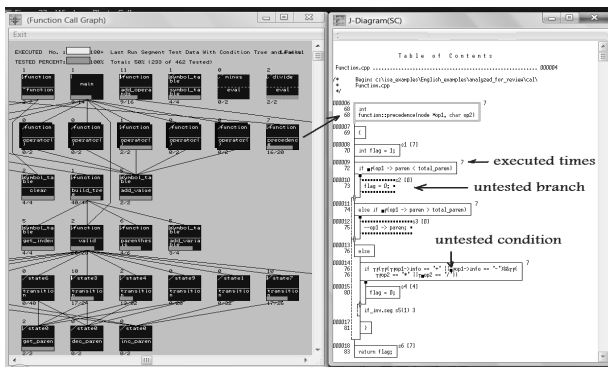


(B1)

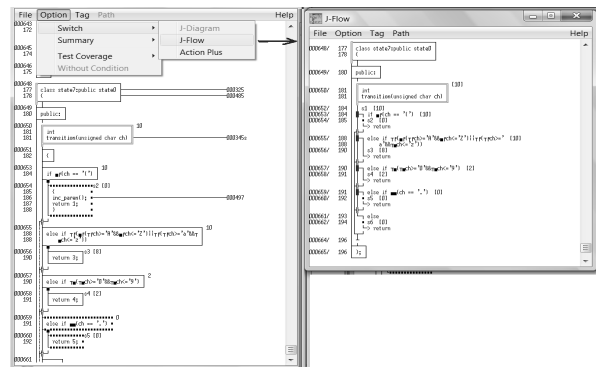


(C2)

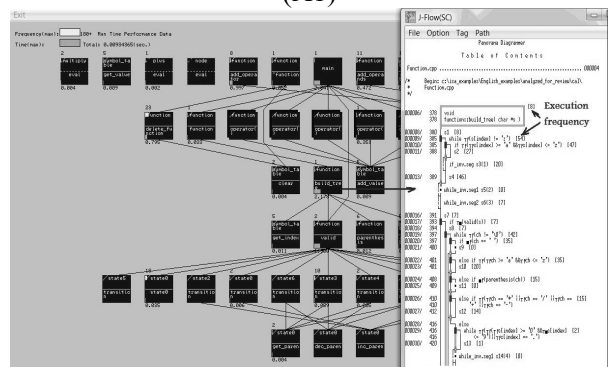
Fig. 4 Application examples of 3J graphics - A1: a call graph shown in J-Chart notations with test coverage analysis result (a small bar chart on the bottom of each module-box presents the percentage of the source code tested) ; A2: A call graph shown with Cyclomatic complexity (the number of decision statements) measurement result, and the traceability – tracing a module with the all related modules calling and called by it; B1: A J-Diagram shows a program logic and the related information as well as the traceability facility; B2: A J-Diagram shows that the logic diagram is independent from the writing styles of source code; C1: A J-flow diagram used for semi-automated test case design by automatically choosing a path with most untested elements and the test conditions extracted; C2: A J-Flow diagram used for automated debugging (through an “EXIT” word automatically added at the runtime error location).



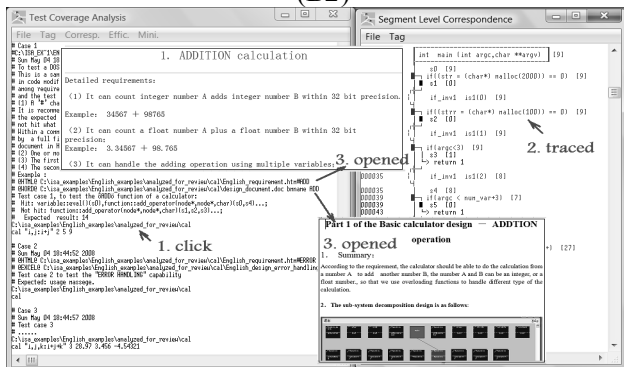
(A1)



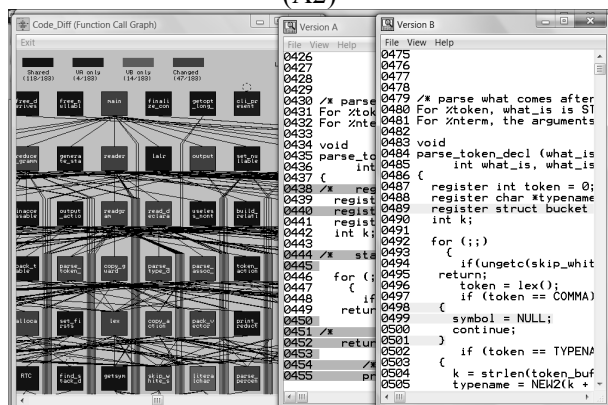
(B2)



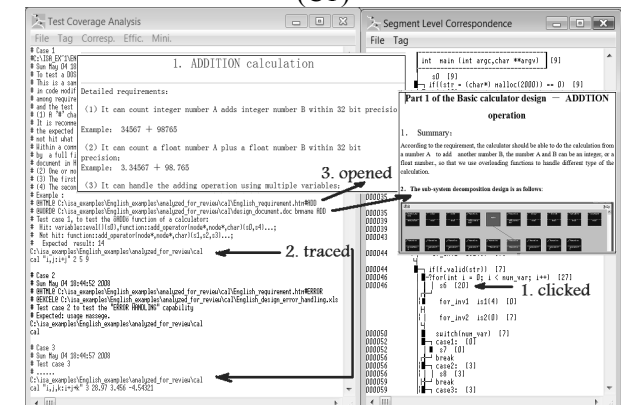
(A2)



(C1)



(B1)



C2

Fig.5 Combination applications of the 3J graphics – A1: An overview of the test coverage measurement results of an entire software product shown holistically in J-Chart with the untested branches and untested conditions of a module being highlighted in J-Diagram using small black boxes; A2: An overview of the performance measurement result of an entire software product shown holistically in J-Chart with the branch execution frequency of a module shown in J-Flow diagram for locating the performance bottleneck easier; B1: A J-Chart showing the version comparison result of an entire software product (where a unchanged module is shown in blue, changed in red, deleted in brown, and added in green) with the detailed differences of the source code of a changed module shown in different colors in two separated Windows; B2: AN example of converting J-Diagram to J-Flow diagram automatically; C1: Tracing a test case (automatically shown in blue) to the source code with the tested segments highlighted in red color in J-Flow diagram; C2: Tracing a source code segment shown in J-Flow (automatically shown in blue) to find the corresponding test cases (in red) and the related documents open.

Software Maintenance Engineering Revolution

Po-Kang Chen¹, Jay Xiong²

¹Y&D Information system, Inc. USA

²International Software Automation, Inc. (ISA, currently being reorganized), USA

Abstract - *software tends to make system of complexity in the modern software system. 40 years passing, software maintenance hasn't been an excellent solution. Software always costs a amount of payment for its maintenance because they don't have efficient solutions to maintained problems. This article presents a revolutionary paradigm for software maintenance engineering – the NSE (Nonlinear Software Engineering) software maintenance engineering paradigm based on complexity science by complying with the essential principles of complexity science, particularly the Nonlinearity principle and the Holism principle, so that with this paradigm almost all software maintenance tasks/activities are performed holistically and globally with side-effect prevention through many kinds of automated and self-maintainable traceabilities. Preliminary applications show that compared with the old-established software maintenance engineering paradigm, it is possible for the NSE software maintenance engineering paradigm to help software development organizations reduce about 2/3 of the total effort and total cost spent in software product maintenance.*

Keywords: software maintenance, defect prevention, traceability, software testing, nonlinear , complexity science

1 The Old-Established Software Maintenance Engineering Paradigm Is Outdated

After delivery, software products need to be modified for meeting requirement changes, fixing bugs, improving performance, and keeping it usable in a changed or changing environment.

But unfortunately, the old-established software maintenance engineering paradigm is outdated because: It is based on reductionism and the superposition principle that the whole of a complex system is the sum of its components, so that almost all of the tasks and activities in software maintenance engineering are performed partially and locally.

The corresponding software development process models used are linear ones with no upstream movement at all - requiring software engineers to do all things right at all times without making any mistake, but that is impossible.

With the linear process models, the defects brought into a software product in the upper phases easily propagate down to

the maintenance phase to make the maintenance tasks much harder to perform.

The corresponding software development methodologies do not offer “maintainable design” without support of various kinds of bidirectional traceabilities.

It is not systematic – the old-established software maintenance engineering paradigm does not offer systematic approaches for software maintenance: there is no systematic software maintenance process model defined.

It is not quantifiable – for instance, when a module is modified, there is no facilities provided to get quantifiable data about how many requirements and how many modules may be affected.

It is not disciplined – there is no engineering approach and model defined to guide maintainers to perform software maintenance step by step to prevent side-effects and ensure the quality of the modified products.

It is invisible – the maintenance engineering process and the results obtained are invisible, making it hard to review and evaluate.

It is blind – for instance, after the implementation of a requirement change or code modification, it requires the maintainers to use all test cases to perform regression testing blindly, no matter whether a test case is useful or useless to re-test the modified software product.

It is costly - As pointed out by Scott W. Ambler, “The Unified Process suffers from several weaknesses. First, it is only a development process... it misses the concept of maintenance and support... It's important to note that development is a small portion of the overall software life cycle. The relative software investment that most organizations make is allocating roughly 20% of the software budget for new development, and 80% to maintenance and support efforts.”[1].

It makes a software product being maintained unstable day by day – As pointed out by Frederick P. Brooks Jr., “The fundamental problem with program maintenance is that fixing a defect has a substantial (20-50 percent) chance of introducing another. ... All repairs tend to destroy the structure, to increase the entropy and disorder of the system.”[2]

It makes a software product developed by others much harder to maintain at the customer site – today a software product is delivered with the program, the data used, and the documents separated from the source code without bidirectional traceability and intelligent agents (intelligent tools) to support testability, visibility, changeability,

conformity, reliability, and maintainability.

It is easy to become a project killer or even a business killer - as pointed out by Roger S. Pressman, "Over three decades ago, software maintenance was characterized as an 'iceberg'. We hope that what is immediately visible is all there is to it, but we know that an enormous mass of potential problems and cost lies under the surface. In the early 1970s, the maintenance iceberg was big enough to sink an aircraft carrier. Today, it could easily sink the entire navy!"[3].

2 Outline of the Revolutionary Solution Offered by NSE

The revolutionary solution offered by NSE for software maintenance will be described in detail in this article later. Here is the outline of the solution:

It is based on complexity science that the whole of a complex system is greater than the sum of its components – the characteristics and behaviors of the whole emerge from the interaction of its components, so that with NSE almost all of the tasks and activities in software maintenance engineering are performed holistically and globally.

The corresponding software development process model used is a nonlinear one with two way iteration (see Fig. 1) supported by automated and self-maintainable traceability to prevent defects brought into software products by the product developers and the customers.

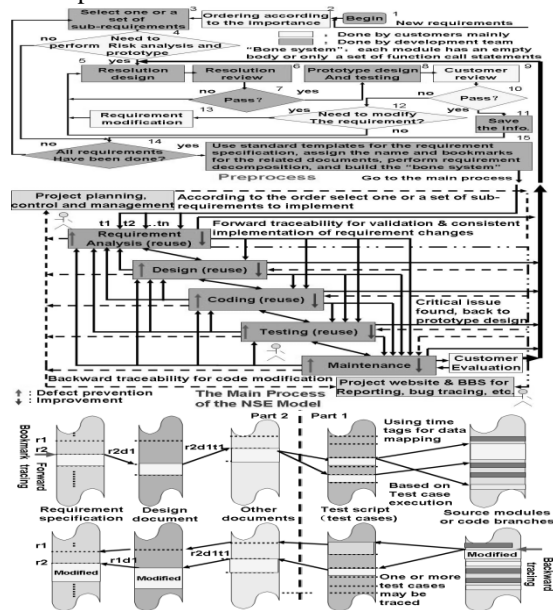


Fig. 1 The NSE Process Model which includes the preprocess part, the main process part, and the automated and self-maintainable facility to support bi-directional traceability using Time Tags automatically inserted into both the test case description part and the corresponding test coverage database for mapping test cases and the tested source code, and some keywords to indicate the related document types such as @WORD@, @HTML@, @PDF@, @BAT@, @EXCEL@ written in the test case description part followed by the file paths and the bookmarks to be used to open the traced documents from the specified positions.

With the nonlinear process models used, most of the defects brought into a software product can be efficiently removed through defect propagation prevention mainly by dynamic testing in the entire software development life cycle using the Transparent-box testing method [4] innovated by me to combine functional testing and structural testing together seamlessly: to each test case it not only checks whether the output (if any, can be none when the method is applied in the requirement development phase and the design phase - having an output is no longer a condition to use this software testing method dynamically) is the same as what is expected, but also checks whether the real execution path covers the expected one specified in J-Flow (a new type control flow diagram innovated by me), and automatically establishes bidirectional traceability among the related documents, the test cases, and the source code to help the developers remove inconsistency defects. NSE complies with W. Edwards Deming's product quality principle, "Cease dependence on inspection to achieve quality. Eliminate the need for inspection on a mass basis by building quality into the product in the first place." [5]

The corresponding software development methodology offers "maintainable design" supported by various kinds of bidirectional traceabilities for defect prevention, defect propagation prevention, and side-effect prevention in the implementation of requirement changes and code modification [4]– as pointed out by Frederick P. Brooks Jr., "Clearly, methods of designing programs so as to eliminate or at least illuminate side effects can have an immense payoff in maintenance costs."[2].

It is systematic – NSE software maintenance engineering paradigm offers systematic approaches for software maintenance: there is a systematic software maintenance process model defined to guide users to perform software maintenance holistically and globally (see section 4).

It is quantifiable – for instance, when a module or even only one statement of the source code is modified, NSE software maintenance engineering paradigm can help users get quantifiable data on exactly about how many requirements and other modules may be affected.

It is disciplined – there is a defined engineering approach and model to guide maintainers to perform software maintenance step by step to prevent side-effects, ensure the quality of the modified products, and perform regression testing efficiently.

It is visible – with NSE the maintenance engineering process and the results obtained are visible and easy to review and evaluate, because it is supported with a set of Assisted Online Agents including software visualization tools to automatically generate huge amount of graphical documents which are interactive and traceable – see Fig. 2 a sample call graph shown in J-Chart notations innovated by me.



Fig 2 An automatically generated call graph shown with the cyclomatic complexity (the number of decision statements such as 'if', 'for', 'while', 'do', 'switch') measurement result and a module highlighted with the all related modules calling and called by it

It is not blind – for instance, after the implementation of a requirement change or code modification, it help the maintainers efficiently select the useful test cases through backward traceability and test case minimization for performing regression testing efficiently.

It is not costly – it is possible for NSE software maintenance engineering paradigm to help software organization to greatly reduce the cost and effort spent in software maintenance because that With NSE, quality assurance is performed in the entire lifecycle through defect-prevention and defect propagation prevention using the Transparent-box testing method dynamically, plus inspection using traceable documents and traceable source code, so that the defects propagated into the maintenance phase are greatly reduced.

The implementation of requirement changes and code modifications are performed holistically and globally, rather than partially and locally.

The side-effects in the implementation of requirement changes and code modification are prevented through various kinds of automated and self-maintainable traceabilities.

Regression testing after software modification is performed efficiently through backward traceability to select the corresponding test cases and test case minimization to select the useful test cases to greatly reduce the required time, resources, and cost.

It makes a software product being maintained stable – with NSE there is no big difference between the product development process and the product maintenance process: in both processes, requirement changes are welcome to support the customers' market strategy, and implemented holistically and globally with side-effect prevention through various kinds of traceabilities.

It makes a software product developed by others easy to maintain at the customer site – even if a software product is maintained at the customer site rather than the product development site, software maintenance engineering can be performed with almost the same conditions as those at the product development site, because with NSE the delivery of a

software product includes not only the computer program, the data used, and the documents traceable to and from the source code, but also the database built through static and dynamic measurement of the program, and a set of Assisted Online Agents to make the software adaptive and truly maintainable (see section 5 to know how those Assisted Online Agents work together to support testability, reliability, changeability, visibility, conformity, traceability, adaptability, and maintainability).

NSE software maintenance engineering paradigm becomes a key to make it possible for NSE to help software organization double their productivity and halve their cost in their software product development – with the NSE, not only the most defects are removed in the development process through defect prevention and defect propagation prevention, but new defects are also prevented in the maintenance process through various kinds of traceabilities and dynamic testing using the Transparent-box testing method - all software maintenance tasks are performed holistically and globally with side-effect prevention, so that the effort and cost spent in the software maintenance will be almost the same as that spent in the software development process – each one takes about 25% of the original cost: about half of the total effort and total cost can be saved.

It can be efficiently applied to the worst case where no documents exist at all – in this case, NSE software maintenance engineering paradigm will use the Assisted Online Agents to automatically generate huge amount of various documents through reverse engineering, then help users set bookmarks in the generated documents. After users re-design the test cases with some simple rules and re-test the product, NSE software maintenance engineering paradigm will automatically establish various automated and self-maintainable traceability to make the product adaptive and maintainable.

3 The Foundation for the Establishment of NSE Software Maintenance Engineering Paradigm

The foundation for the establishment of NSE software maintenance engineering paradigm is complexity science by complying with the essential principles of complexity science, particularly the Nonlinearity principle and the Holism principle that the whole of a complex system is greater than the sum of its components, and that the characteristics and behaviors of the whole emerge from the interaction of its components, so that with the NSE software maintenance paradigm almost all software maintenance engineering tasks/activities are performed holistically and globally to prevent the side-effects in the implementation of requirement changes or code modifications.

The establishment of the NSE software maintenance paradigm is done through the use of the FDS (the Five-Dimensional Structure Synthesis method - a paradigm-shift framework innovated by me) as shown in Fig. 3.

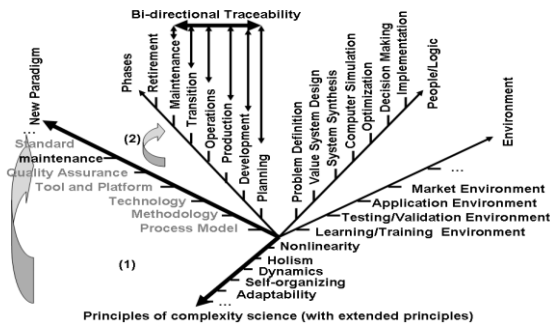


Fig. 3 The framework for establishing NSE software maintenance engineering paradigm

4 Description of NSE Software Maintenance Engineering Paradigm

With NSE a software maintenance process model is defined as shown in Fig. 4.

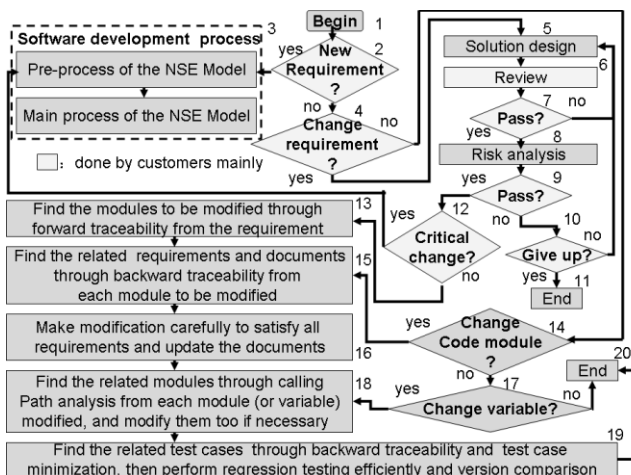


Fig. 4 NSE Software Maintenance Process Model

As shown in Fig. 4, the major steps for performing software maintenance engineering are as follows:

- Step 1: Begin.
- Step 2: Check the maintenance task type. If it is for the implementation of a new requirement, go to step 3; otherwise go to step 4.
- Step 3: Perform the implementation of the requirement through the preprocess and the main process regularly as what was performed in the software development process.
- Step 4: Is a critical change of the requirement? If not, go to step 14.
- Step 5: Perform solution design.
- Step 6: Go through the solution review process.
- Step 7: If the review result is not good enough, go to step 5.
- Step 8: Perform risk analysis.
- Step 9: If the risk analysis result is good enough, go to step 12.
- Step 10: Give up? If not, go to step 5.

- Step 11: End the process without changes.
- Step 12: Is a critical change? If so, go to step 3.
- Step 13: Find the modules to be modified through forward traceability (from requirement -> the corresponding test cases -> the corresponding source code, see section 5 an application example). Go to step 15.
- Step 14: Is it not for changing the source modules? If so, go to step 17.
- Step 15: Find the related requirements and documents through backward traceability from each module to be modified.
- Step 16: Make modifications carefully to satisfy all of the related requirements (often a module is used for the implementation of more than one requirement) and update the related documents. If necessary, add some new modules and perform unit testing (including memory leak measurement and performance measurement) for the new modules. Go to step 18.
- Step 17: Is it to change a global or static variable? If not, go to step 20 (end the process).
- Step 18: Find the related modules through calling path analysis from each module/variable modified, and modify them too if necessary.
- Step 19: Find the related test cases through backward traceability and perform test case minimization, then perform regression testing efficiently (including MC/DC test coverage analysis, memory leak measurement, performance measurement, quality measurement, and runtime error location through execution path tracing, see section 5 for an example), and version comparison holistically.
- Step 20: End the process.

5 Application

As described, with NSE a software product will be delivered with the computer program, the data used, and the documents traceable to and from the source code, plus the database built through static and dynamic measurement of the program, and a set of Assisted Online Agents to support testability, visibility, changeability, conformity, traceability, and maintainability. Those Assisted Online Agents are listed as follows:

- (1) NSE-CLICK interface
- (2) OO-Browser for generating interactive and traceable call graphs and class inheritance charts shown in J-Chart notations
- (3) OO-Diagrammer for generating interactive and traceable logic diagrams shown in J-Diagram notation or control flow diagram in J-Flow notation innovated by me
- (4) OO-V&V for Requirement Validation and Verification through bidirectional traceability
- (5) OO-SQA for software quality measurement
- (6) OO-MemoryCheck for checking memory leaks and usage violations
- (7) OO-Analyzer for dynamic and static program measurement

- (8) OO-Performance for performance measurement
 - (9) OO-DefectTracer for tracing each runtime error to the execution path
 - (10) OO-MiniCase for test case efficiency analysis and test case minimization in order to perform regression testing efficiently after code modification
 - (11) OO-Playback for GUI operation capture and playback after code modification
 - (12) OO-CodeDiff for holistic and intelligent software version comparison, etc.
- [4] Jay Xiong, Jonathan Xiong, A Complete Revolution in Software Engineering Based on Complexity Science, WORLDCOMP'09 – SERP (Software Engineering Research and Practice 2009) , 109-115.
 - [5] Deming W E. Out of the Crisis. MIT Press, 1982.

6 Conclusions

As described in the NSE software maintenance process model and shown in the application examples, the major features of NSE software maintenance engineering paradigm can be briefly summarized as follows: Based on complexity science Performed holistically and globally, side-effect prevention driven, supported by various traceabilities, visual in the entire software maintenance process, intelligent in the test case selection for regression testing through backward traceability systematic, quantifiable, and disciplined conclusion.

Today software maintenance takes 75% or more of the total effort and total cost in software product development, because the existing software maintenance engineering paradigm is based on reductionism and the superposition principle, so that almost all of the tasks and activities in software maintenance engineering are performed partially and locally.

This article presented the NSE software maintenance engineering paradigm based on complexity science. With NSE software maintenance engineering paradigm almost all software maintenance tasks/activities are performed holistically and globally with side-effect prevention in the implementation of requirement changes and code modifications through various traceabilities. Preliminary applications show that compared with the old-established software maintenance engineering paradigm, it is possible for NSE software maintenance engineering paradigm to reduce about 2/3 of the total effort and total cost in software maintenance to help software organization double their productivity and halve their cost in their software product development.

7 References

- [1] Ambler S W. A Manager's Introduction to The Rational Unified Process (RUP), Ambysoft. 2005
- [2] Brooks, Frederick P. Jr., "The Mythical Man-Month", Addison Wesley, 1995, P120
- [3] Pressman, Roger S., "Software Engineering: A Practitioner's Approach", McGraw-Hill, 2005, P409

Should Linear, Partial, Local, and Qualitative Software Engineering Paradigm Be Replaced by Nonlinear, Holistic, Global, and Quantitative Software Engineering Paradigm?

Jay Xiong

NSEsoftware, LLC., USA

jayxiong@yeah.net; jay@nsesoftware.com

Abstract

This paper summarizes the major drawbacks of the old-established software engineering paradigm offering linear, partial, local, and qualitative approaches for software product development, and lists the reasons why the old-established software engineering paradigm should be replaced by a new one offering nonlinear, holistic, global, and quantitative software product development approaches.

Keywords: Software engineering paradigm, complexity science, nonlinear system, software development methods, software maintenance

1. Introduction

As pointed out by Capers Jones that “Major software projects have been troubling business activities for more than 50 years. Of any known business activity, software projects have the highest probability of being cancelled or delayed. Once delivered, these projects display excessive error quantities and low levels of reliability.”[1]

Why? What is the root cause?

There are many different answers to this question:

Several researchers have suggested that “CMM does not effectively deal with the social aspects of organizations” [2].

Timothy K. Perkins believes that “the cause of project failures is knowledge: either managers do not have the necessary knowledge, or they do not properly apply the knowledge they have.”[3].

Capers Jones concluded that “Both technical and social issues are associated with software project failures. Among the social issues that contribute to project failures are the rejections of accurate estimates and the forcing of projects to adhere to schedules that are essentially impossible. Among the technical issues that contribute to project failures are the lack of modern estimating approaches and the failure to plan for requirements growth during development. However, it is not a law of nature that software projects will run late, be cancelled, or be unreliable after deployment. A careful program of risk analysis and risk abatement can lower the probability of a major software disaster.”[1].

Joe Marasco pointed out that “All the effort has gone into two areas: managing requirements and something called ‘requirements traceability.’ Requirements management is the art of capturing requirements, cataloging them, and monitoring their evolution throughout the development cycle. Requirements are added, dropped, changed, and so on, and we now have requirements management systems that allow us to keep track of all this. That is a good thing. Traceability is a bit more ambitious. It attempts to link later-stage artifacts, such as pieces of a system and their test cases, back to the original requirements. That way, we can assess if we are actually meeting the requirements that were called out. This is a harder problem, but, once again, there has been substantial progress. To all this I say, wonderful, but not good enough.” (For more information, see the Standish Group Website at <http://www.standishgroup.com/>).

“Poor Estimation: Major Root Cause of Project Failure” (Galarath Incorporated, <http://www.galarath.com/wp/poor-estimation-major-root-cause-of-project-failure.php>).

“IT projects have been considered a tough undertaking and have certain characteristics that make them different from other engineering projects and increase the chances of their failure. Such characteristics are classified in seven categories (Peppers, Gengler & Tuunanen, 2003; Salmeron & Herrero, 2005): 1) abstract constraints which generate unrealistic expectations and overambitious projects; 2) difficulty of visualization, which has been attributed to senior management asking for over-ambitious or impossible functions, the IT project representation is not understandable for all stakeholders, and the late detection of problems (intangible product); 3) excessive perception of flexibility, which contributes to time and budget overrun and frequent requests of changes by the users; 4) hidden complexity, which involves difficulties to be estimated at the project's outset and interface with the reliability and efficiency of the system; 5) uncertainty, which causes difficulty in specifying requirements and problems in

implementation of the specified system; 6) the tendency to software failure, which is due to assumptions that are not thought of during the development process and the difficulty of anticipating the effects of small changes in software; 7) the goal to change existing business processes, which requires IT practitioners' understanding of the business and processes concerned in the IT system and good processes to automate and make them quicker. Such automation is unlikely to make a bad process better.” (International Management Review, 2009 by Al-Ahmad, Walid, et al., *A Taxonomy of an IT Project Failure: Root Causes*, Business Publications, http://findarticles.com/p/articles/mi_qa5439/is_200901/ai_n31965631/?tag=content;col1)

In the article “Why Big Software Projects Fail: The 12 Key Questions”[4], Watts S. Humphrey listed those questions as follows:

“Question 1: Are All Large Software Projects Unmanageable?

Question 2: Why Are Large Software Projects Hard to Manage?

Question 3: Why Is Autocratic Management Ineffective for Software?

Question 4: Why Is Management Visibility a Problem for Software?

Question 5: Why Can't Managers Just Ask the Developers?

Question 6: Why Do Planned Projects Fail?

Question 7: Why Not Just Insist on Detailed Plans?

Question 8: Why Not Tell the Developers to Plan Their Work?

Question 9: How Can We Get Developers to Make Good Plans?

Question 10: How Can Management Trust Developers to Make Plans?

Question 11: What Are the Risks of Changing?

Question 12: What Has Been the Experience So Far?”

“Root causes of project failure ...

- Ad hoc requirements management.
- Ambiguous and imprecise communication.
- Brittle architectures.
- Overwhelming complexity.
- Undetected inconsistencies in requirements, designs, and implementations.
- Insufficient testing.
- Subjective project status assessment.
- Failure to attack risk.
- Uncontrolled change propagation.
- Insufficient automation.” (devdaily, http://www.devdaily.com/java/java_oo/node7.shtml)

In my opinion, they are reasonable answers to the question, but not the fundamental reason for software project failure.

According to the essential principles of complexity science, particularly the **Nonlinearity principle** and the **Holism principle**, software is a nonlinear complex system where the whole is greater than the sum of its parts, the behaviors and characteristics of the whole emerge from the interaction of its parts and the interaction between the system and its environment, small differences in the initial condition or a small change to the system may produce large variations in the long term behavior of the system – the “**Butterfly-Effect**”. But unfortunately, the existing software engineering paradigm is based on linear thinking, reductionism, and the superposition principle that the whole is the sum of its parts, so that almost all tasks/activities are performed linearly, partially, locally, and qualitatively. It means that the foundation of the existing software engineering paradigm is wrong. The wrong foundation makes almost all things wrong in software engineering, particularly the process models, the development methods, the modeling approaches, the visualization paradigm, the testing paradigm, the quality assurance paradigm, the documentation paradigm, the maintenance paradigm, and the project management paradigm – in fact the existing software engineering paradigm is entirely outdated.

2. The Major Drawbacks of the Old-Established Software Engineering Paradigm

The Major Drawbacks of the Old-Established Software Engineering Paradigm can be summarized as follows:

- (a) **Incomplete** – for instance, there is no defined process model and support for software maintenance which takes 75% or more of the total effort and cost for a software product
- (b) **Unreliable** – the quality of a software product mainly depends on software inspection and testing after production which has been proven impossible to ensure high quality
- (c) **Invisible** – the existing visualization methods, techniques, and tools do not offer the capability to make the entire software development lifecycle visible, the generated charts and diagrams are not holistic and not traceable
- (d) **Inconsistent** – the documents and the source code are not traceable to each other and not consistent after code modification again and again
- (e) **Unchangeable** – the implementation of requirement change or code modification is performed locally and blindly with high risks

- (f) **Not maintainable** – software maintenance is performed partially, locally, and qualitatively without support for bidirectional traceability to prevent side effects, so that each code modification will have a 20–50% of chance to introduce new defects into the software product
- (g) **Low productivity and quality** – most resources are spent in inefficient software maintenance, the quality cannot be ensured with the blind and local implementation of software changes
- (h) **High cost and risk** – most cost is spent in blind and local maintenance of the software products, which makes a software product unstable day by day in responding to needed changes
- (i) **Low project success rate** – it is still less than 30% for projects with budgets over \$1 million
- (j) **Often the software projects developed with the old-established software engineering paradigm are capable of becoming a monster of missed schedules, blown budgets, and flawed products** – because the old-established software engineering paradigm is based on linear thinking, reductionism, and superposition principle, so that almost all tasks/activities are performed linearly, partially, locally, and qualitatively. It is clear that those problems are related to the entire software engineering paradigm with all of its components, including the process models, the software development methodologies, the modeling approaches, the visualization paradigm, the software testing paradigm, the quality assurance paradigm, the documentation paradigm, the maintenance paradigm, the project management paradigm, and the related techniques and tools. It means that a local, partial, and qualitative solution will not work – **we need a holistic, global, and quantitative solution in almost all aspects of software engineering: a complete revolution.**

3. What Is NSE

For solving those critical problems existing with today's software development efficiently, a new software engineering paradigm, NSE (Nonlinear Software Engineering paradigm based on complexity science) is established. The essential difference between the old-established software engineering paradigm and NSE is how to handle the relationship between the whole and its parts of a software system. **The former adheres to the reductionism principle and superposition principle that the whole is the sum of its parts**, so that nearly all software development tasks/activities are performed locally, such as the implementation of requirement changes. **The latter complies with the Holism principle of complexity science, that a software product is a Complex Adaptive System (CAS [5]) having multiple interacting agents (components), of which the overall behavior and characteristics cannot be inferred simply from the behavior of its individual agents but emerge from the interaction of its parts**, so that with NSE nearly all software development tasks/activities are performed

globally and holistically to prevent defects in the entire software lifecycle [6], [7]. Some primary applications show that the NSE paradigm with its support platform, Panorama++, can make revolutionary changes to almost all aspects in software engineering to efficiently handle software complexity, invisibility, changeability, and conformity, and solve the critical problems (low productivity and quality, high cost and risk) existing with the old-established software engineering paradigm – NSE makes it possible to help software development organizations double their productivity, halve their cost, and remove 99.99% of the defects in their software products.

4. Should Linear, Partial, Local, and Qualitative Software Engineering Paradigm be replaced by Nonlinear, Holistic, Global, and Quantitative Software Engineering Paradigm?

4.1 Linear Engineering Vs. Nonlinear Engineering

The old-established software engineering paradigm offers linear engineering for software development. Fig. 1 shows various different linear approaches whose process models are linear with no upstream movement at all.

The major drawbacks of linear software engineering:

- (a) It violates the nature law of human that people are nonlinear, easy to make mistakes in thinking, working, reading, and writing so that there is a need for them to correct the mistakes by themselves.
- (b) Linear software engineering assumes that customers know their all requirements in details at the beginning of the corresponding software project, but it is impossible – customers need time to learn by themselves to understand what they really need.
- (c) Linear software engineering makes software defects introduced in upstream to easily propagate to downstream and the defect removal cost increase by several orders of magnitude.
- (d) Linear software engineering makes software design documents inconsistent with source code after code modification again and again.
- (e) Linear software engineering makes a software product much difficult to change and maintain.

Differently, NSE offers nonlinear software engineering whose process model is shown in Fig. 2. The major features of NSE nonlinear engineering approach :

- (a) NSE process model always assumes that there may be defects introduced in the upper phases so that there is a need to check and remove the defects in the upper phases through dynamic testing using the innovated Transparent-box method (see Fig. 3) and backward

traceability that is established automatically using Time Tags and some keywords to indicate the format of the documents, the file paths, and the bookmarks

(see Fig. 4). Similarly, changes made in the upper phases may affect the work products obtained in lower level phases, so that there is also a need to check and remove the inconsistency defects in lower level phases through forward traceability.

- (b) NSE offers source code driven approach for dynamic software modeling and top-down plus bottom-up software development through stub programs using dummy modules (having an empty body or only a list of function call statements without detailed program logic) in forward engineering, or regular programs in reverse engineering. With it all models/diagrams are automatically generated from source code.
- (c) NSE makes design become pre-coding, and coding become further design – after coding, all related models/diagrams and documents can be automatically updated through rebuilding the corresponding database.
- (d) With NSE software documents are consistent and traceable with the source code to make a software product much easy to change and maintain.

4.2 Partial Engineering Vs. Holistic Engineering

With the old-established software engineering paradigm, software engineering is performed partially rather than holistically. For instance, in the modeling process, many small pieces of models/diagrams will be drawn partially but missing the big picture of the entire software product – software components will be completed first, then the entire software product will be built through integration from the components.

Differently, with NSE software engineering will be performed holistically. For instance, in the modeling process the models/diagrams for the entire system will be generated from the stub programs as shown in Fig. 5, then the whole system with dummy modules as an embryo will grow up incrementally with the detailed module design and coding, so that customers can try the entire system early (even if it could be a stub system) and the system-level testing can be performed early, and that when defects are found while the system is growing up incrementally (each time only a module is allowed to be added to the system), the defects will be much easy to locate.

4.3 Local Engineering Vs. Global Engineering

The old-established software engineering paradigm offers local engineering for software development, such as the process of the implementation of requirement changes or code modifications according to the top-down linear process model and the lack of bi-directional traceability, so

that each time when a bug is fixed, there is a 20-50% of chance to introduce another to the system.

Differently, NSE offers global engineering for software development, such as the process of the implementation of requirement changes or code modifications according to the nonlinear process model and the rich support of bi-directional traceability to prevent the possible side-effect. Fig. 6 shows that when a class member function of a Java program is modified what class member functions may be affected globally; Fig. 7 shows how many statements may be affected in system-level globally.

4.4 Qualitative Engineering Vs. Quantitative Engineering

Software is a logic product, not a machine product. The algorithm is the soul of software. For realizing a solution to solve a problem, such as the sorting issue for student names, different people innovate different algorithms to solve the same problem efficiently or inefficiently. Algorithm innovation should not follow engineering steps, otherwise the algorithms created by different people will be very similar.

But the implementation of any algorithm for solving any problem should follow engineering rules and steps, otherwise the quality of a software will be very difficult to ensure.

Unfortunately, current software engineering is a qualitative engineering which in fact is not a real engineering. For instance, when a program unit needs to be modified in software maintenance, the software maintainers do not know how many requirements are related to that unit, how many other program units may be affected by the change of that unit, and how many test cases can be efficiently used to re-test the software product, and more.

Often a software product developed with qualitative engineering is not reliable and not maintainable

Qualitative software engineering is the fundamental reason why the critical issues (low quality and productivity, and high cost and risk) have existed for more than 40 years.

A quantitative software engineering paradigm is established by complying with the essential principles of complexity science, particularly the Nonlinearity Principle and Holism Principle, and supported by automated, bi-directional, and self-maintainable traceability among requirements and test cases and source code.

For instance, in responding to a requirement changes, with the current qualitative software engineering paradigm, the maintainers do not know

- (a) how many classes and program modules are related to the change;
- (b) If a related class or function needs to be modified, how many other requirements may be affected;
- (c) If a related class or function needs to be modified, how many classes or functions may also need to be modified;

- (d) If a related class or function needs to be modified, how many test cases can be used to re-tested the modified system;
- (e) How many global variables and static variables may be affected;
- (f) How many documents may need to be modified; and more.

With NSE supporting quantitative software engineering in responding to a requirement changes, the maintainers know the all information listed above.

Some application examples of quantitative software engineering are shown in Fig. 8 and Fig. 9 for safe implementation of a requirement change.

5. Conclusion

It is concluded that for efficiently solving the critical issues (low quality and productivity, and high cost and risk) existing with today's software development, the old-established linear, local, and qualitative software engineering paradigm should be replaced by nonlinear, holistic, global, and quantitative software engineering paradigm such as NSE.

References

[1] Capers J (2006) Social and technical reasons for software project failures. CrossTalk, Jun Issue

[2] Ngwenyama O, Nielsen PA (2003) Competing values in software process improvement: an assumption analysis of CMM from an organizational culture perspective. IEEE Trans Eng Manag 50(1):100 - 112. doi:10.1109/TEM.2002.808267

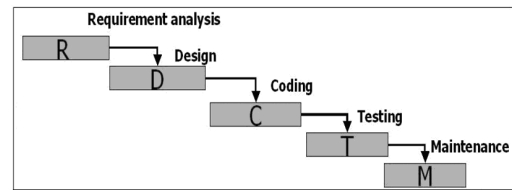
[3] Perkins TK (2006) Knowledge: the core problem of project failure. CrossTalk, Jun Issue

[4] Humphrey WS (2005) The Software Engineering Institute, Why big software projects fail: the 12 key questions. CrossTalk, Mar Issue

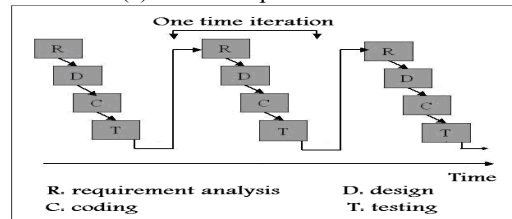
[5] Holland JH (1995) Hidden order: how adaptation builds complexity. Addison- Wesley, Reading

[6] Jay X (2009) Tutorial, a complete revolution in software engineering based on complexity science, WORLDCOMP' 09, Las Vegas, July 13 - 17, 2009

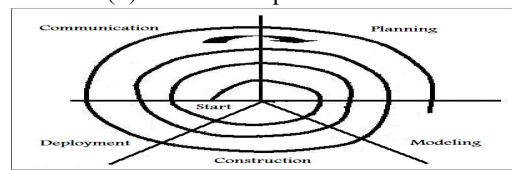
[7] Jay X, Jonathan X (2009) A complete revolution in software engineering based on complexity science, WORLDCOMP' 09 - SERP (Software Engineering Research and Practice 2009), pp 109 - 115



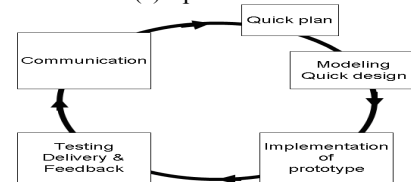
(a) Waterfall process model



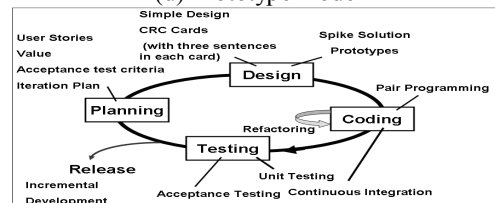
(b) Incremental process model



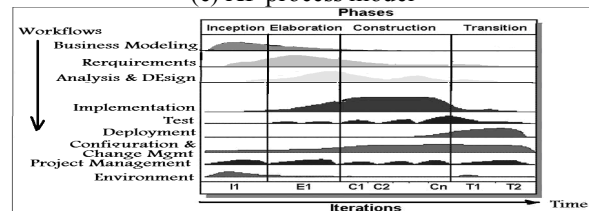
(c) Spiral model



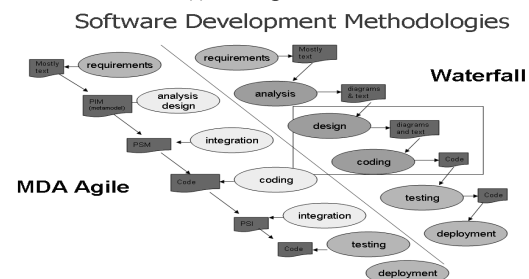
(d) Prototype model



(e) XP process model



(f) RUP process model



(g) MDA Agile process model

Fig. 1 Various linear software engineering process models

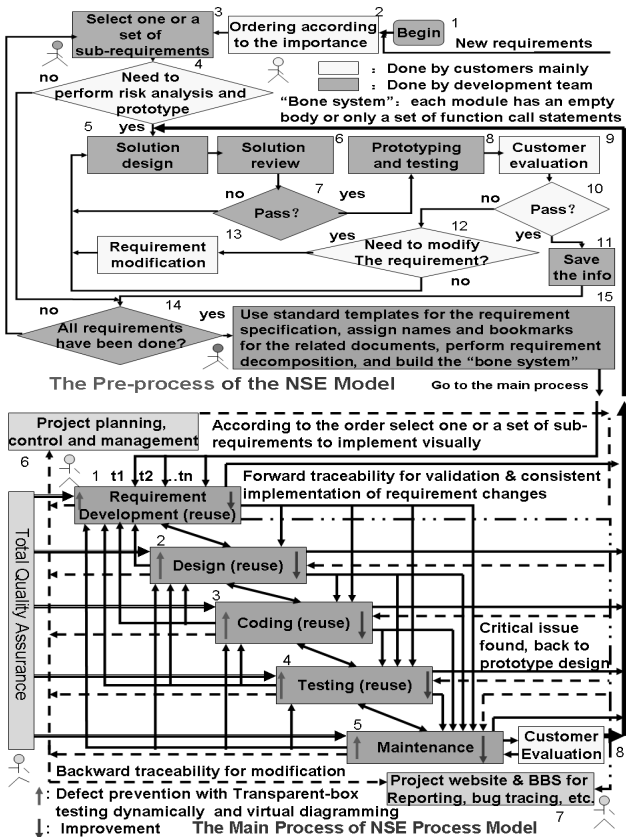


Fig. 2 NSE nonlinear process model

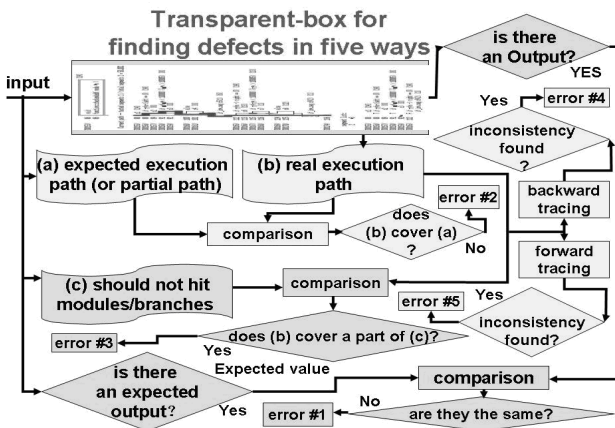


Fig. 3 Transparent-box software testing method

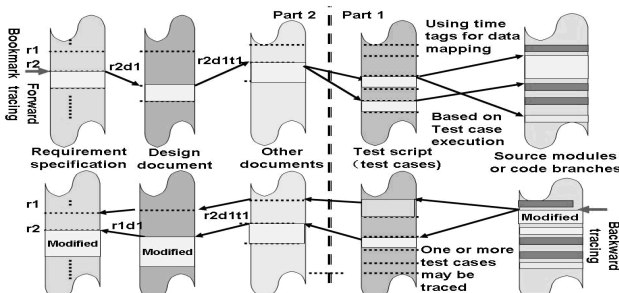


Fig. 4 The facility for automated and self-maintainable traceability



Fig. 5 A call graph of a entire software product designed (on the left side) and a module with its all related modules traced/highlighted

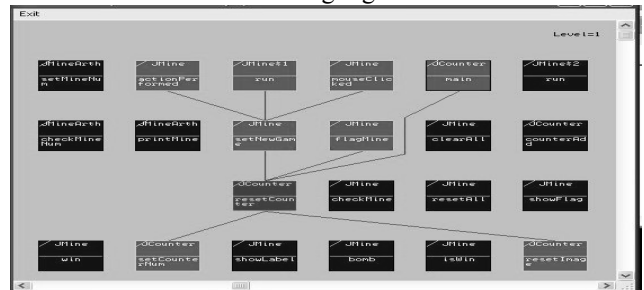


Fig. 6 A call graph with a class member function and all related class member functions traced

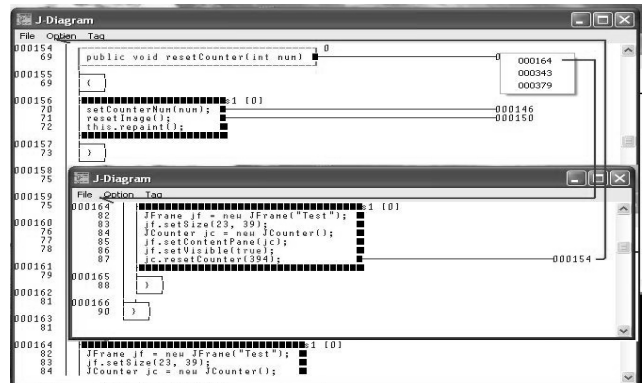


Fig. 7 statements which may be affected globally by the modification of the class member function Jconter::resetCounter

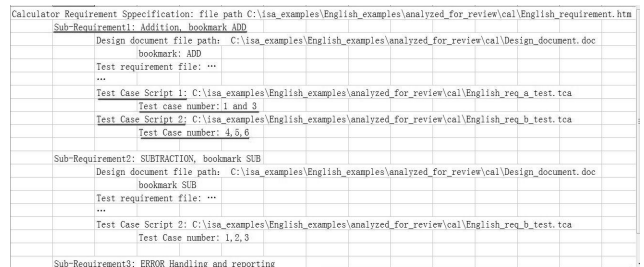


Fig. 8 From the requirement(s) to be changed to find the related test cases through the document hierarchy description table

General Comparison between the Old-Established Software Engineering Paradigm and NSE (Nonlinear Software Engineering Paradigm)

Jay Xiong

NSEsoftware, LLC., USA

jayxiong@yeah.net, jay@nsesoftware.com

Abstract

This paper describes the major differences between the old-established software engineering paradigm and a new software engineering paradigm called NSE (Nonlinear software engineering paradigm based on complexity science) in detail. The essential difference between them is how to handle the relationship between the whole and its parts of a software system. The former adheres to the reductionism principle and superposition principle that the whole is the sum of its parts, so that nearly all software development tasks/activities are performed partially and locally, such as the implementation of requirement changes. The latter complies with the Holism Principle of complexity science, that a software product is a Complex Adaptive System having multiple interacting agents (components), of which the overall behavior and characteristics cannot be inferred simply from the behavior of its individual agents but emerge from the interaction of its parts, so that with NSE nearly all software development tasks/activities are performed globally and holistically to prevent defects in the entire software lifecycle.

Keywords: Silver Bullet, software engineering paradigm, modeling testing, quality assurance, maintenance

1. Introduction

Low quality, productivity, and project success rate, and high cost and risk are the critical issues which have existed with the old-established software engineering paradigm for more than 40 years. The root cause is that software is a nonlinear system where a small change may bring big impact to the entire system – the “Butterfly-Effect”, but the old-established software engineering paradigm is an outcome of reductionism and the superposition principle that the whole of a nonlinear system is the sum of its parts, so that with it almost all software engineering activities are performed linearly, partially, and locally.

NSE (Nonlinear Software Engineering paradigm) was first time introduced in our previous paper titled “A Complete Revolution in Software Engineering Based on Complexity Science” published in 2009 with SERP'09[1], and described in more detail in my book, “New Software Engineering Paradigm Based on Complexity Science” published in 2011 [2]

This paper will further compare the differences between the old-established software engineering paradigm and NSE in almost all parts, including the modeling approaches, the software development methods, the software development processes, the testing paradigms, the quality assurance paradigms, the documentation paradigms, the visualization paradigms, the maintenance paradigms, and the project management paradigms.

2. General Comparison between the Old-Established Software Engineering

Paradigm and NSE

2.1 Software Definition

A. The software definition of the old-established software engineering paradigm

Software is defined as

- * instructions (computer programs) that when executed provide desired features, function, and performance;
- * data structures that enable the programs to adequately manipulate information; and
- * documents that describe the operation and use of the programs [3].

B. The software definition of NSE

Software is defined as

- * instructions (computer programs) that when executed provide desired features, function, and performance;
- * data structures that enable the programs to adequately manipulate information; and
- * documents that describe the operation and use of the programs (including the test case script files too); **plus**
- * **the database built though static and dynamic measurement of the programs; and**
- * **a set of Associated Online Agents (AOA, automated and intelligence tools working with the database) for supporting testability, reliability, visibility,**

changeability, conformity, and traceability to make the software program maintainable, adaptive, and that the static and dynamic measurement results can be viewed easily, and the acceptance testing can be dynamically done in a fully automated way through mouse clicks only.

2.2 Software Engineering foundation

A. The old-established software engineering paradigm

The old-established software engineering foundation is based on linear thinking, reductionism, and the superposition principle that **the whole of a system is the sum of its parts**, so that **with it** almost all software development tasks/activities are performed linearly, partially, and locally.

B. NSE

The NSE foundation is based on complexity science with a set of essential principles including the **Nonlinearity** principle, the **Holism** principle that **a whole is greater than the sum of its parts - the characters and the behavior of a complex system is an emergent property of the interactions of its components (agents)**, the **Dynamics** principle, the **Self-organization** principle, the **Self-adaptation** principle, the **Openness** principle, the **Initial Condition Sensitivity** principle, the **Sensitivity to Change** principle, the **Complexity Arises From Simple Rules** principle, etc., so that with NSE, almost all tasks/activities are performed globally and holistically through a nonlinear process.

2.3 Software Development Methods

A. The old-established software engineering paradigm

A Top-Down or Bottom-Up method is used linearly.

B. NSE

A nonlinear Top-Down plus Bottom-up method is used, driven by defect prevention supported by various traceability (see Fig. 1).

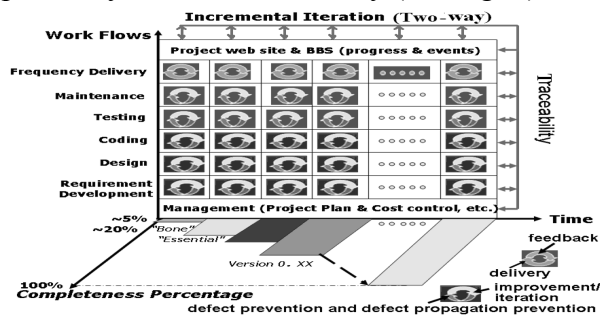
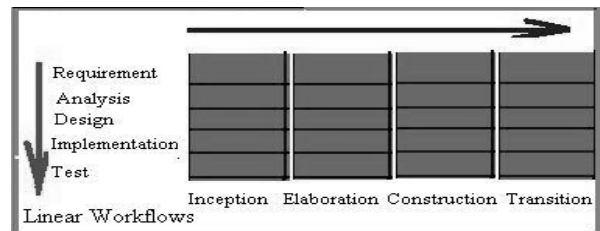
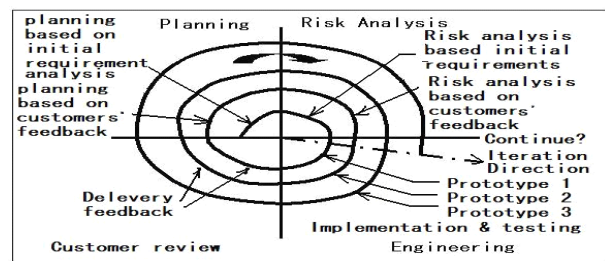
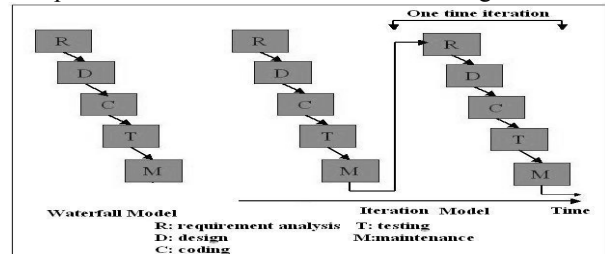


Fig. 1 NSE Software development method

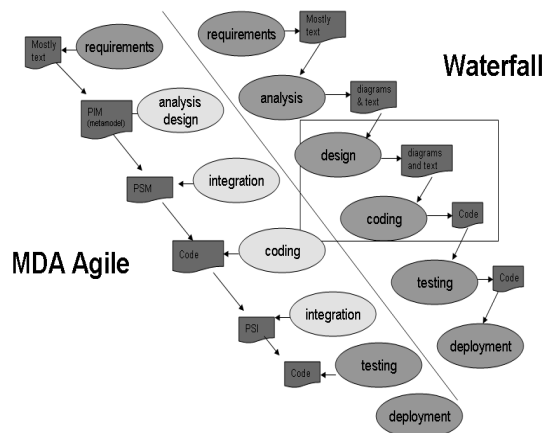
2.4 Software Engineering Process Models

A. The old-established software engineering paradigm

With the old-established software engineering paradigm all software engineering process models are linear with no upstream movement at all as shown in Fig 2.



Software Development Methodologies



(Source: "An Analysis of Model Driven Architecture (MDA) and Executable UML (xUML)", http://www.powershow.com/view/30871-MjU5N/An_Analysis_of_Model_Driven_Architecture_MDA_and_Executable_UML_xUML_flash_ppt_presentation)

Fig. 2 Various Linear Software Engineering Process Models with No Upstream Movement at All

B. NSE

With NSE a nonlinear software engineering process model is offered as shown in Fig. 3.

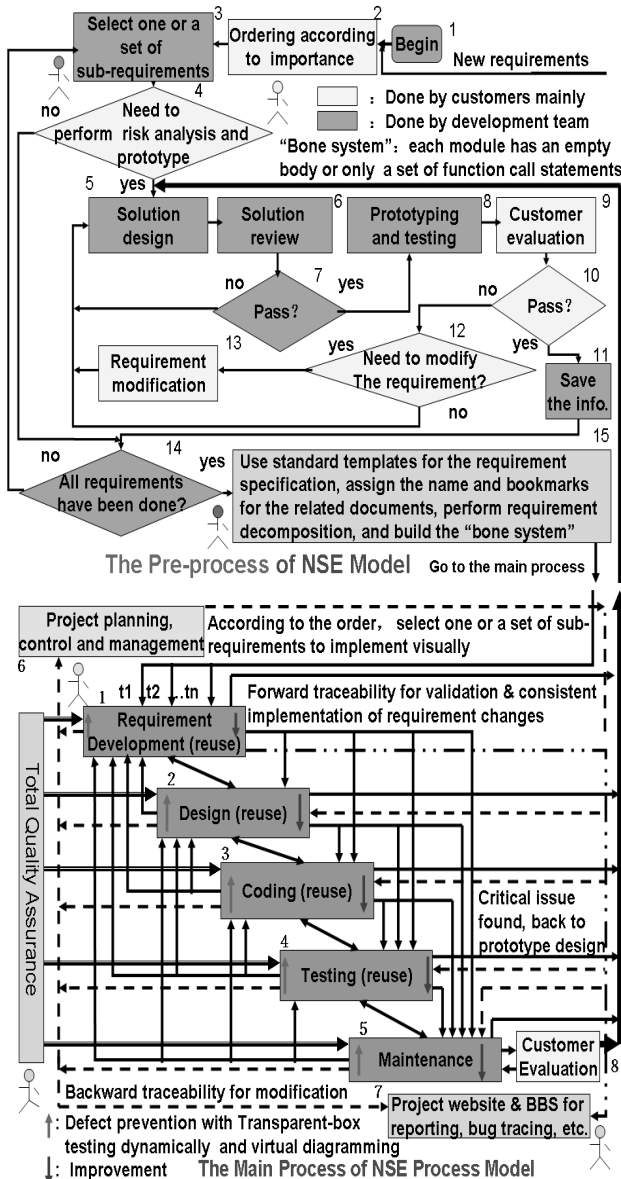


Fig. 3 NSE software engineering process model

The major features of NSE process model include:

- * **Dual-process:** NSE model consists of the pre-process and the main process. They are different but also closely linked together.
- * **Nonlinear:** The NSE model is nonlinear, complying with the Nonlinearity principle and the Holism principle.
- * **Parallel with Multiple tracks:** "Much of software architecture, implementation, and realization can proceed in parallel." [4]. For reducing waiting time and speeding up software development processes, the

NSE process model supports tasks being performed in parallel with multiple tracks through bidirectional traceability.

- * **Real time:** "Timely updating is of critical importance." [4]

- * **Incremental development with two-way iteration:** The NSE process model supports incremental development with two-way iteration, including refactoring to handle highly complex modules and performance bottlenecks with side-effect prevention. When a critical issue is found in the main process, the work flow may go back to the preprocess for selecting a better solution method, and so on.

- * **The software development process and software maintenance process are combined together seamlessly:** With the NSE process model, there is no big difference between the software development process and the software maintenance process – both support requirement changes through side-effect prevention.

- * **The software development process and the project management process are combined together closely:** all documents including the project management documents such as the project development plan, the schedule chart, and the cost estimation report are traceable with the requirement implementation and the source code for better control of the product development. NSE process model also supports the critical requirements and most important requirements being implemented early with the assigned priority to avoid budget overuse – if necessary, some optional requirements and not so important requirements can be ignored temporarily.

- * **Adaptation focused rather than predictability focused:** the entire world is always changing, so the NSE process model is adaptation focused rather than predictability focused – it supports requirement changes, code modifications, data modifications, and document modifications to make them consistent and updated with side-effect prevention.

- * **Defect prevention driven**

- * **People are considered as the first order driver for software development** - When people consider "people as the first-order" to software development, they focus on how to trust and support people better for their jobs, but ignore the other side of people's effect on software development – almost all defects introduced into software products are made by people, the developers and the customers. So NSE supports people in two ways: one is to support them with better methodology, technology, and tools; another one is to prevent the possible defects to be introduced into the software products by people - it is done mainly through various automated and bidirectional traceabilities.

2.5 Software Modeling Approaches

A. The old-established software engineering paradigm

Offering linear, partial, and local software modeling approaches through two kinds of sources with one in graphics drawn by hands or using a graphic editor for human understanding of a software product, and another one in test format for computer understanding of the software product – there is a big gap between the two kinds of sources as shown in Fig. 4.

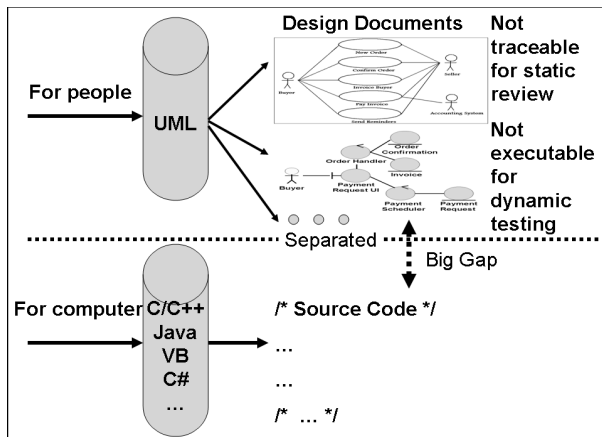


Fig. 4 Existing software modeling approaches

B. NSE

Offering nonlinear, holistic, and global software modeling approach (called NSM – Nonlinear Software Modeling approach) using one kind of source (the source code of a stub program using dummy modules having an empty body or only some function call statements without detailed program logic, or the source code of a regular in forward engineering or reverse engineering) for both human understanding of a software system using better graphics automatically generated from the source code for high-level system abstraction, and computer understanding of the software system using the source code directly as shown in Fig. 5.

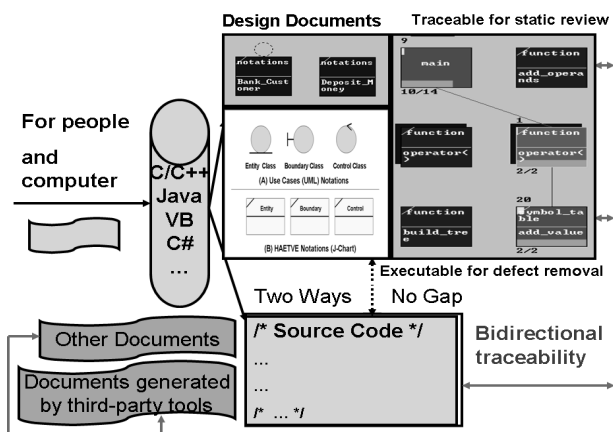


Fig. 5 NSE software modeling approach

The major features of NSE software modeling approach include:

- (1) NSM is based on complexity science, complying with the essential principles of complexity science, particularly the Nonlinearity Principle, the Dynamic Principle, and the Holism Principle, so that with NSM almost all software modeling and engineering activities are performed nonlinearly, holistically, and globally, rather than linearly, partially, and locally.
- (2) NSM uses one kind source (the source code of a platform-independent programming language (such as Java/Java-DSL) or even a platform-dependent programming language) for human understanding of a complex software product through the colorful and meaningful Models/Diagrams automatically generated from the source code for high-level abstraction, and computer understanding of the complex software product using the source code or the transformed source code, so that with NSM the models/diagrams are always consistent with the source code.
- (3) NSM makes design become pre-coding, and coding become further design – offering Top-Down plus Bottom-Up software development approach.
- (4) NSM offers dynamic software modeling approach rather than static one: (a) with NSM, the generated models/diagrams are existing dynamically - when a chart or a diagram is shown, the corresponding generator is always working for users' commands to operate to meet users' needs through the interface – using the chart/diagram itself; (b) with NSM, the generated models/diagrams are dynamically executable through the corresponding source code; (c) with NSM, the generated models/diagrams are dynamically traceable to the requirements and the source code.
- (5) NSM completely solves the inconsistency issues between the generated models/diagrams and the source code.
- (6) NSM brings revolutionary changes to software modeling quality by making the generated models/diagrams traceable for static defect removal, and executable through the corresponding source code for dynamic defect removal.
- (7) The models/diagrams generated with NSM are accurate and precise to the source code.
- (8) With NSM a software product developed through nonlinear software modeling and engineering is much easier to understand, review, change, test, and maintain.

2.6 Software Testing Paradigm

A. The old-established software engineering paradigm

Functional testing and structural testing are separated, performed after coding, can not be used to find defects in

requirement development phase and design phase dynamically..

B. NSE

Functional testing and structural testing are combined together seamlessly using the innovated Transparent-box testing method which not only checks whether the output (if any, can be none) from a software product being developed is the same as what is expected, but also helps users to check whether the real program execution path covers the expected program execution path, then automatically establishes bi-directional traceability among related documents and test cases and the source code according to the description of the test cases through Time Tags (when a test case is executed) and some special keywords (see Fig. 6 and Fig. 7)

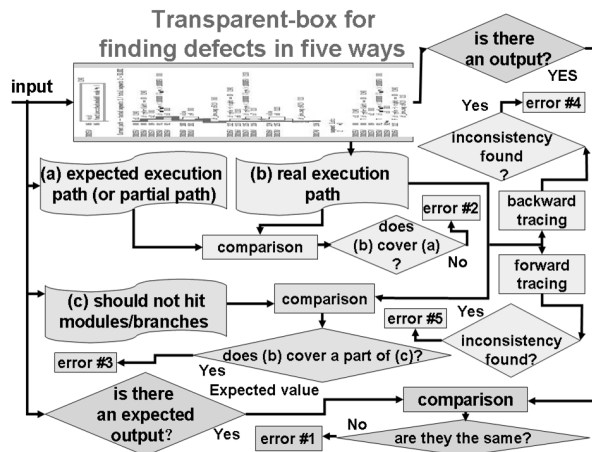
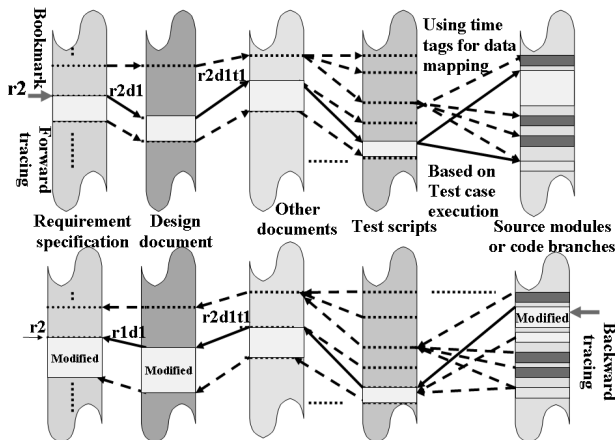


Fig. 6 The innovated Transparent-box testing method



The facility for bidirectional traceability among all artifacts and the source code 1

Fig. 7 Automated and self-maintainable traceability

2.7 Software Quality Assurance Paradigm

A. The old-established software engineering paradigm

Software quality assurance is mainly based on inspection and software testing after coding, violates W. Edwards Deming’s product quality assurance that “*Cease dependence on inspection to achieve quality. Eliminate the need for inspection on a mass basis by building quality into the product in the first place.*” [5].

B. NSE

Software quality assurance is based on defect prevention and defect propagation prevention performed in the entire software development process from requirement development down to maintenance through program execution using the innovated Transparent-box testing method dynamically, plus inspection using traceable documents and traceable source code automatically diagrammed.

2.8 Software Documentation Paradigm

A. The old-established software engineering paradigm

Software documents are separated from the source code without bi-directional traceability between them, and often inconsistent with source code after code is modified again and again..

B. NSE

Software documents are combined with source code through various traceability, and consistent with source code after code modification through backward traceability to update the related documents directly or updating the corresponding database for re-generating most documents from source code – see Fig. 8.

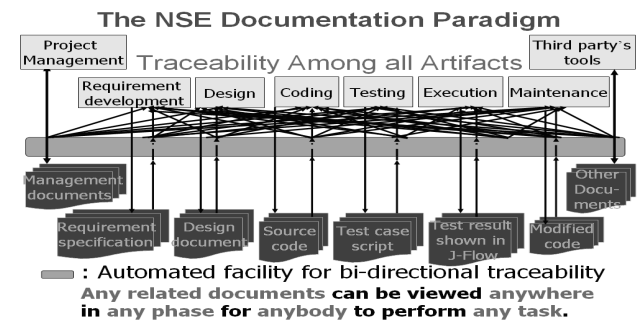


Fig. 8 NSE documentation paradigm

2.9 Software Visualization Paradigm

A. The old-established software engineering paradigm

Partially supported in the modeling process using UML and the support tools.

B. NSE

With the NSE process model and the support platforms, the entire software development process is visible from the first step to the maintenance phase using integrative and traceable 3J graphics (J-Chart, J-Diagram, and J-Flow innovated by me) and the corresponding diagramming tools, which generate all charts and diagrams globally and holistically with various kinds of traceabilities to make the software product being developed much easier to understand, test, and maintain - see Fig. 8.

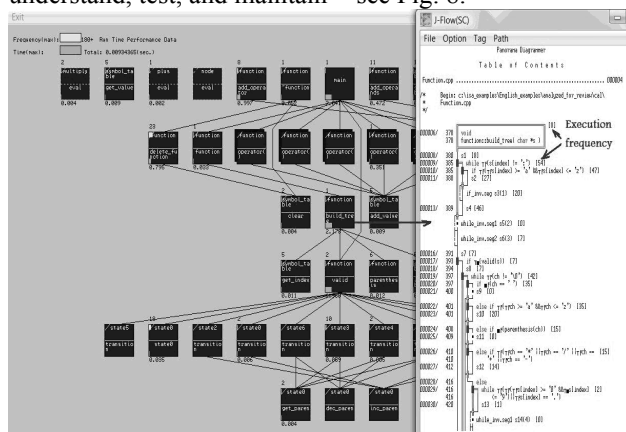


Fig. 9 An application example of NSE software visualization paradigm

2.10 Software Maintenance Paradigm

A. The old-established software engineering paradigm

Based on linear process models without facilities for various bidirectional traceabilities, or very limited traceability made manually; software maintenance is performed locally and partially with no way to prevent the side-effects for the implementation of requirement changes or code modifications, so that often when a bug is fixed, there is a 20% to 50% chance to introduce a new one to the software product. Often the regression testing is performed by reusing all test cases - it is time consuming and costly. It is why software maintenance takes more than 75% of the total cost and total effort in a software system development.

B. NSE

Based on the NSE nonlinear process model with the support of facilities for various bidirectional traceabilities

that are automatically established, software maintenance is performed globally and holistically with side-effect prevention. There is no big difference between the software development process and the maintenance process, because with NSE requirement changes are welcome at any time to support the customer's market competition strategy, and responded to in real time where the side-effects for the implementation of requirement changes or code modifications can be prevented to assure the quality through various bidirectional traceabilities. The regression testing after code modification can be performed with minimized test cases to greatly save the cost and time. In the case that only a few code branches are modified, only some related test cases will be selected for regression testing through backward tracing from the modified branches to the test case scripts. The regression testing will use the Transparent-box method which combines functional testing and structural testing together seamlessly with the capability to establish the new bidirectional traceabilities, and the capability to perform performance measurement, memory leak and usage violation check, and MC/DC (Modified condition/Decision Coverage) test coverage measurement. If something wrong is found after the code modification, a global and holistic version comparison will be performed for helping users to find and fix the problem quickly.

2.11 Software Project Management

A. The old-established software engineering paradigm

The project management processes are separated from the product development processes - the project plan/schedule information and the cost information are not traceable with the requirement implementation, so that often a software becomes a monster of missed schedules and blown budgets.

B. NSE

The project management processes and the product development processes are combined together, making the project plan/schedule information and the cost information traceable with the requirement implementation and the source code, assigning implementation priorities to the requirements according to the importance and market needs, so that the schedules and budgets can be controlled better. Particularly, the NSE nonlinear process model is used with defect prevention for the implementation of requirement changes or code modification to greatly reduce the cost spent in the software development process and the software maintenance process, and ensure the quality from the first step to the end of a software development project.

An application example is shown in Fig. 10.

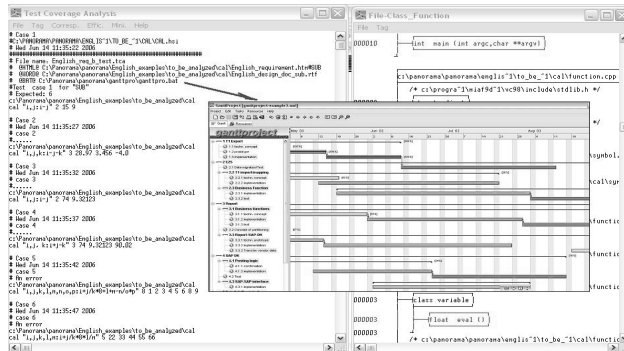


Fig. 10 An application example of making project management documents traceable with requirements and source code

3. The Essential Differences between the Old-Established Software Engineering Paradigm and NSE

The Essential Differences between the Old-Established Software Engineering Paradigm and NSE is how to handle the relationship between the whole of a nonlinear system and its parts – the old-established software engineering paradigm is based on reductionism and the superposition principle that the whole of a nonlinear system is the sum of its components, so that with it almost all software engineering activities are performed linearly, partially, and locally; but NSE is based on complexity science by complying with the essential principles of complexity science, particularly the Nonlinearity principle and the Holism principle that the whole of a nonlinear system is greater than the sum of its parts, the characteristics and behaviors of the whole of a nonlinear system emerge from the iteration of its parts, can not be inferred simply from the behavior of its individual components, so that with NSE almost all software engineering activities are performed non linearly, holistically, and globally.

4. Conclusion

Why have the critical issues (low quality and productivity, and high cost and risk) existed for more than 40 years? The main reason is that software is a nonlinear system where a small change may bring big impact to the entire system. Each one of the critical issue, such as the low quality issue, is related to the all parts of software engineering, including the software development method, and software modeling approach, the software engineering modeling approach, the software testing paradigm, the documentation paradigm, the visualization paradigm, and the maintenance paradigm, so that only improve the quality assurance method and tools without improving the other parts of an entire software engineering paradigm will not be able to efficiently solve the software quality issue – for efficiently solving the all

critical issues in software development, we need a complete revolution in software engineering based on complexity science as described in this paper.

References

- [1] Jay Xiong, 2009, Tutorial, *A Complete Revolution in Software Engineering Based on Complexity Science*, WORLDCOMP'09 - , Las Vegas, July 13-17, 2009 (http://www.world-academy-of-science.org/worldcomp09/ws/tutorials/tutorial_xiong)
- [2] Jay Xiong, 2011, *New Software Engineering Paradigm Based on Complexity Science*, Springer (<http://www.springer.com/physics/complexity/book/978-1-4419-7325-2>)
- [3] Pressman R. S. *Software Engineering: A Practitioner's Approach*, McGraw-Hill, 2005
- [4] Brooks, Frederick P. Jr., "The Mythical Man-Month", Addison Wesley, 1995
- [5] Deming W E. *Out of the Crisis*. MIT Press, 1986

Silver Bullet: Slaying Software Werewolves Efficiently

Jay Xiong

NSEsoftware, LLC., USA

jayxiong@yeah.net, jay@nsesoftware.com

Abstract

This paper introduces a Silver Bullet for slaying software werewolves efficiently by shifting software engineering foundation from reductionism to complexity science. The Silver Bullet complies with the essential principles of complexity science, including the Nonlinearity Principle, the Holism Principle, the Dynamics Principle, the Self-Organization Principle, the Self-Adaptation Principle, the Openness Principle, and more, so that with the Silver Bullet almost all software engineering tasks are performed nonlinearly, holistically, globally, and quantitatively to bring revolutionary changes to almost all areas of software engineering. The Silver Bullet has been fully implemented and supported by Panorama++ platform. Theoretical comparisons and preliminary applications show that compared with the old-established software engineering paradigm, it is possible for the Silver Bullet to help software development organizations double their productivity and project success rate, halve their cost, improve the quality of their products in several orders of magnitude, and slay software werewolves ("a monster of missed schedules, blown budgets, and flawed products") efficiently.

Keywords: Silver Bullet, software engineering paradigm, modeling, testing, quality assurance, maintenance

1. Introduction

Software "Werewolves" is defined by Brooks in his paper "No Silver Bullet: Essence and Accidents of Software Engineering" published in 1984 [1] as that "Of all the monsters who fill nightmares of our folklore, none terrify more than werewolves, because they transform unexpectedly from the familiar into horrors", "The familiar software project has something of this character (at least as seen by the nontechnical manager), usually innocent and straightforward, but capable of becoming a monster of missed schedules, blown budgets, and flawed products."

"No Silver Bullet" - Brooks also pointed out that "There is no single development, in either technology or management technique, which by itself promises even one order-of-magnitude improvement within a decade in productivity, in reliability, in simplicity."

Here it is clear that, the "werewolves" is a monster of missed schedules, blown budgets, and flawed products" – these issues relate to the entire software engineering paradigm, including the process models, the software development methodology, the quality assurance paradigm, the software testing paradigm, the project management paradigm, the software documentation paradigm, the software maintenance paradigm, the self-organization capability, the Capability Maturity of the organization and the team, and more. But the "Silver Bullet" defined by Brooks is a "single development, in either technology or management technique, which by itself promises even one order-of-magnitude improvement within a decade in productivity, in reliability, in simplicity." – how can a single technology or management technique solve the issues of **missed schedules, blown budgets, and flawed products** which are not only technology or technique issues but strongly related to people (the customers and the developers) and the project management?

The answer is that the "Silver Bullet" defined by Brooks can not slay the "werewolves" defined by him:

(1) In theory, it is impossible

According to complexity science, the whole of a complex system is greater than the sum of its parts, the characteristics and behaviors of the whole of a complex system emerge from the interaction of its components, can not be inferred simply from the behavior of its individual components. It means

a single development, in either technology or management technique, the individual characteristics and behaviors can not be inferred simply by the whole of the software engineering paradigm, so that it is impossible for the single development, in either technology or management technique to slay the software monster of missed schedules, blown budgets, and flawed products - those problems come from the whole of the old-established software engineering paradigm.

(2) From practices, it is impossible

After analyzing more than 12,000 software projects, Capers Jones pointed out in his article titled "Social and Technical Reasons for Software Project Failures" that "Major software projects have been troubling business activities for more than 50 years. Of any known business activity, software projects have the highest probability of being cancelled or delayed. Once delivered, these projects display excessive error quantities and low levels of reliability. Both technical and social issues are associated with software project failures. Among the social issues that contribute to project failures are the rejections of accurate estimates and the forcing of projects to adhere to schedules that are essentially impossible. Among the technical issues that contribute to project failures are the lack of modern estimating approaches and the failure to plan for requirements growth during development. However, it is not a law of nature that software projects will run late, be cancelled, or be unreliable after deployment. A careful program of risk analysis and risk abatement can lower the probability of a major software disaster." [2] – it means that the issues of missed schedules, blown budgets, and flawed products are not only technology issues, but also social issues, can never be solved by a single development, in either technology or management technique.

With the same reasons, CMMI (*Capability Maturity Model Integration*, focusing on Software Process Improvement and project management improvement only) or SEMAT (Software Engineering Method and Theory, mainly focusing on the improvement of software development methodology) without bringing revolutionary changes to the entire software engineering paradigm will not be able to efficiently slay software "werewolves" too.

This paper describes Silver Bullet which is, in fact, a complete revolutionary software engineering paradigm based on complexity science.

2. What Does a Qualified Silver Bullet Mean?

Before answering this question, let us consider what make the software “werewolves” exist:

- (a) The existing process models (no matter if they are waterfall models, incremental development models which is “a series of Waterfalls”[3], or iterative development models on which each iteration is a waterfall) which are based on reductionism and superposition principle that the whole of a complex system is the sum of its components, so that with them almost all software process tasks and activities are performed linearly, partially, and locally without upstream movement at all, making the defect introduced into a software product in upstream easy to propagate down to the maintenance phase and the final defect removal cost increase tenfold many times.
- (b) The software development methodologies based on linear process, reductionism, superposition, and constructive holism principle, so that with them almost all software development tasks and activities are performed linearly, partially, and locally for the components of a software product first, then the components are “assembled” (CMMI) to form the whole of the software product, making the quality of the software product very hard to ensured, and software maintenance much hard to perform.
- (c) The top-down software modeling approaches including MDA, MDD, and MDE based on UML, with which the obtained models/diagrams are not traceable for static defect removal, not executable for debugging, and not dynamically testable for dynamic defect removal, so that nobody knows whether they are complete, correct, and consistent with each other - they are not qualified as the road map for project implementation.
- (d) The software testing paradigm which ignores the fact that most critical software defects are introduced to a software product in the requirement development phase and the product design phase, can only be dynamically used after coding, so that NIST (National Institute of Standards and Technology) concluded that “Briefly, experience in testing software and systems has shown that testing to high degrees of security and reliability is from a practical perspective not possible. Thus, one needs to build security, reliability, and other aspects into the system design itself and perform a security fault analysis on the implementation of the design.” (“Requiring Software Independence in VVSG 2007: STS Recommendations for the TGDC,” November 2006, <http://vote.nist.gov/DraftWhitePaperOnSIinVVSG2007-20061120.pdf>). Even if a defect has been found through dynamic software testing after coding, the defect removal cost will increase tenfold several times.
- (e) The quality assurance paradigm base on inspection and software testing after production, which violates W. Edwards Deming’s product quality principle that “*Cease dependence on inspection to achieve quality. Eliminate the need for inspection on a mass basis by building quality into the product in the first place.*” [4], making software quality hard to ensure.
- (f) The software visualization paradigm which mainly supports visual modeling only, does not make the entire software development and maintenance process and the work products visible, so that software engineers and maintainers need to spend much more time to understand and maintain a software product.
- (g) The software documentation paradigm with which the documents are not traceable with the source code, and often do

not consistent with the source code, making a software hard to understand and hard to maintain.

- (h) The software maintenance paradigm with which the implementation of requirement changes and code modifications are performed blindly, partially, and locally, so that fixing a defect has a substantial (20-50 percent) chance of introducing another[1], making a software product unstable day by day.
- (i) The project management paradigm with which software project management process and the software development process are separated, the software management documents are not traceable to the implementation of requirements and the source code, making the schedule is hard to meet, and the budget is hard to control.
- (j) The corresponding software development techniques and tools which are designed to work with linear process models, hard to be used to handle a complex software which is nonlinear complex system.
- (k) The entire software engineering paradigm which is based on reductionism and superposition principle, hard to efficiently handle a nonlinear software system where a small change may bring big impact to the entire system - Butterfly-Effects.

It means that almost all parts of the old-established software engineering paradigm are making the possibility for the software werewolves to exist.

Now it is the time we can answer the question: only such a Silver Bullet can be used to slay software werewolves:

- (1) it is based on complexity science, complying with the essential principles of complexity science, particularly the Nonlinear principle and the Holism principle, so that with it almost all software development tasks and activities are performed holistically, globally, and quantitatively;
- (2) it not only can bring revolutionary changes to the all parts of the software engineering paradigm, but also can make the required characteristics and behaviors of the whole emerge from the iteration of its all parts.

In fact, **a qualified “Silver Bullet” being able to slay software “werewolves” means a complete revolution in software engineering through paradigm-shift from the old one based on reductionism and superposition principle to a new one based on complexity science.**

3. A Silver Bullet for Slaying Software werewolves Efficiently

A Silver Bullet[5][6] with the support platform, Panorama++, consisting of more than 10,000 function points and one million lines of source code) for slaying software werewolves efficiently has been innovated through the “Five-Dimensional Structure Synthesis Method” (FDS) framework (Fig.1) and implemented by me and my colleagues. Silver Bullet is based on complexity science by complying with the essential principles of complexity science, particularly the Nonlinearity principle and the Holism principle so that with Silver Bullet almost all software engineering tasks are performed nonlinearly, holistically, globally, and quantitatively to bring revolutionary changes to almost all areas of software engineering, including

- **The foundation**

From: that based on reductionism and superposition principle that the whole is the sum of its parts, so that nearly all software development tasks/activities are performed linearly, partially, and locally, such as the implementation of requirement changes.

To: that based on complexity science - to comply with the essential principles of complexity science, particularly the Nonlinear Principle and the Holism Principle that the whole of a complex system is greater than the sum of its parts - the characters and the behavior of a complex system is an emergent property of the interactions of its components (agents), so that with Silver Bullet nearly all software development tasks/activities are performed nonlinearly, holistically, and globally to prevent defects in the entire software life-cycle – for instance, if there is a need to change a requirement, with Silver Bullet and the support platform Panorama++ the implementation of the change will be performed nonlinearly, holistically, and globally through various bidirectional traceabilities: (1) Performs forward tracing for the requirement change (through the corresponding test cases) to determine what modules should be modified. (2) Performs backward tracing to check related requirements of the modules to be modified for preventing requirement conflicts. (3) Checks what other modules may also need to be changed with the modification by tracing the modules to find all related modules on the corresponding call graph shown in J-Chart innovated by me. (4) Checks where the global variables and static variables may be affected by the modification. (5) After modification, checks all related statements calling the modified module for preventing inconsistency defects between them using the diagrammed source code with traceability shown in J-Diagram notations innovated by me. (6) Performs efficient regression testing through backward tracing from the modified module or statement to find the related test cases. (7) Performs backward tracing to find and modify inconsistent documents after code modification.

- **The process model(s)**

From: linear ones based on reductionism principle and superposition principle, including the waterfall models, the incremental development models, the iterative development models, with which there is only one track in one direction - no upstream movement at all, always going forward from the upper phases to the lower phases, so that defects introduced in the upper phases will easily propagate to the lower phases to make the defect removal cost greatly increase.

To: a nonlinear one (the Silver Bullet process model, see Fig. 2 and Fig. 3) based on complexity science with this model there are multiple tracks in two directions through various traceabilities to prevent defects and defect propagation, so that experience and ideas from each downstream part of the construction process may leap upstream, sometimes more than one stage, and affect the upstream activity. With Silver Bullet, the software development process and software maintenance process are combined together closely, the software development process and the project management process are also combined together closely so that the project management documents are traceable with the implementations of software requirements and the source code. With the Silver Bullet process model, requirement validation and verification can be done easily through forward traceability in parallel, and code modification can be done with side-effect prevention through backward traceability in parallel too.

- **the software development methodologies**

From: the software development methods based on Constitutive holism - “**building**” a software system with its components – the components are developed first, then the system of a software product is built through the integration of the

components developed. From the point of view of quality assurance, those methodologies are test-driven but the functional testing is performed after coding; it is too late. These methodologies handle a software product as a machine rather than a logical product created by human beings. They all comply with the reductionism principle and superposition principle.

To: the software development method (Silver Bullet software development method, see Fig. 4) based on Generative Holism of complexity science - having the whole executable dummy system first, then “**growing up**” with its components. From the point of view of quality assurance, it is defect-prevention driven to ensure the quality of a software product.

- **The software modeling approaches**

From: that based on reductionism, offering linear, partial, local, and static modeling approaches, the obtained models/diagram from which are not traceable for static defect removal, not executable for debugging, and not dynamically testable for dynamic defect removal, so that it is very hard to ensure the quality of the modeling results.

To: that based complexity science, offering nonlinear, holistic, global, and dynamic modeling approach, the obtained models/diagram from which are traceable for static defect removal, executable for debugging through the corresponding source code of a stub program using dummy modules or regular programs in reverse engineering, and dynamically testable for dynamic defect removal to ensure the quality of the obtained modeling results.

- **The software testing paradigm**

From: that mainly based on functional testing using the Black-Box testing method being applied after the entire product is produced, plus structural testing using White-Box testing method being applied after each software unit is coded. Both testing methods are applied separately without internal logic connections.

To: that mainly based on the Transparent-box method (Fig.5) innovated by me to combine functional testing and structural testing together seamlessly: to each set of inputs, it not only verifies whether the output (if any, can be none) is the same as the expected value, but also helps users check whether the execution path covers the expected path, with the capability to automatically establish bidirectional traceability among all of the related documents and test cases and the source code for helping users remove inconsistency defects.

- **The quality assurance paradigm**

From: a test-driven approach, mainly using black-box testing method plus structural testing method and code inspection after coding.

To: NSE-SQA – defect prevention-driven approach innovated by me, mainly using the Transparent-box testing method in all phases of a software development life-cycle from the first step to the end because having an output is no longer a condition to use the Transparent-box testing method dynamically. The priority of NSE-SQA for ensuring the quality of a software being developed is ordered as (1) defect prevention; (2) defect propagation prevention; (3) Refactoring applied to highly complex modules and module(s) that are performance bottlenecks; (4) Deep and broad testing.

- **The software visualization paradigm**

From: drawing the diagrams manually or using graphic editors or using a tool to generate partial charts/diagrams which are neither interactive nor traceable in most cases. Even if some charts/diagrams for an entire software system can be generated, they are still not useful because there are too many connection lines to make the charts/diagrams hard to view and

hard to understand without a capability to trace an element with all the related elements.

To: holistic, interactive, traceable, and virtual software visualization paradigm innovated by me to make an entire software development life-cycle visible. The charts/diagrams are dynamically generated from several Hash tables from the database and the source code through stub programming or reverse engineering virtually without storing the hard copies in hard disk or memory to greatly reduce the space. The generated charts/diagrams are interactive and traceable between related elements – users can highlight an element with all of the related elements easily.

- **The documentation paradigm**

From: (a) separated from the source code without bi-directional traceability; (b) inconsistent with the source code after code modifications; (c) requiring huge disk space and memory space to store the graphical documents; (d) the display and operation speed is very slow; (e) hard to update; (f) not very useful for software product understanding, testing, and maintenance.

To: (a) managed together with the source code based on bidirectional traceability; (b) consistent with the source code after code modification; (c) most documents are dynamically generated from several Hash tables and exist virtually without huge storage space; (d) the display and operation speed is very fast; (e) most documents can be updated automatically; (f) very useful for software product understanding, testing, and maintenance.

- **The software maintenance paradigm**

From: that based on reductionism, with which software maintenance is performed blindly, partially, and locally without the capability to prevent the side-effects for the implementation of requirement changes or code modifications, takes about 75% of the total effort and cost in the software system development in most software organizations.

To: that based on complexity science with which software maintenance is performed visually, holistically, and globally using a systematic, disciplined, quantifiable approach innovated by me to prevent the side-effects for the implementation of requirement changes or code modifications through various automated traceabilities; takes only about 25% of the total effort and cost in software system development, because with Silver Bullet there is no big difference between the software development process and the software maintenance process – both support requirement changes or code modification with side-effect prevention.

- **The software project management paradigm**

From: that based on reductionism with which software project management is performed separately from the software product development process, often makes the necessary actions being done too late.

To: that based on complexity science with which software project management is performed closely with the software development process, makes the project management documents such as the product development schedule, the cost reports, and the progress reports traceable with the requirement implementation and the corresponding test cases and the source code, making the necessary actions being done in time.

4. The Major Feature and Characteristics of NSE (Silver Bullet)

The Major Feature and Characteristics of NSE(Silver Bullet) are listed as follows:

- **It is based on a solid foundation - complexity science:** the entire NSE paradigm is established by complying with the essential principles of complexity science, particularly the Nonlinearity principle and the Holism principle.
- **It is complete** – NSE itself is complete, including its own process model, software development methodology, dynamic modeling approach, visualization paradigm, testing paradigm, QA paradigm, documentation paradigm, maintenance paradigm, management paradigm, support techniques and tools and platform.
- **It brings revolutionary changes to almost all aspects of software engineering** – it makes them changed from the old one based on linear processes and the superposition principle to the new one based on complexity science.
- **It offers both “what to do” and “how to do”** – different from some popular models which only offer “what to do” but ignore “how to do”, NSE offers both.
- **With it almost all software engineering tasks/activities are performed holistically and globally** – with NSE, from requirement development down to maintenance, all tasks/activities are performed holistically and globally with defect prevention including side-effect prevention for the implementation of requirement changes and code modification.
- **It combines the software development process and software maintenance process together closely** – with NSE, requirement changes are welcome at any stage and implemented with side-effect prevention through various bidirectional traceabilities .
- **It combines the software development process and software management process together closely** – it makes all documents including the management documents such the schedule chart and the cost reports traceable to the implementation of requirements and the source code to control a software project better and to find and fix the related issues in time.
- **It ensues software product quality from the first step to the final step through defect prevention and dynamic testing using the Transparent-box testing method** – NSE offers many means to prevent defects introduced into a software product by people (the customers and the developers) with dynamic testing using the Transparent-box testing method which combines functional testing and structural testing seamlessly, can be dynamically used in the cases where there is no real output from the software system such as a dummy system with dummy modules only without detailed program logic.
- **With NSE the design becomes pre-coding (top-down), and the coding becomes further design (bottom-up)** – with NSE, in most cases the design through dummy programming using dummy modules becomes pre-coding, and the coding becomes further design through reverse engineering. **It makes software documents traceable to and from source code** – with NSE all related documents and test cases and the source code are traceable forwards or backwards though automated and self-maintainable

traceabilities.

- **It supports real time communication through traceable web pages and traceable technical forum** – with NSE, the bidirectional traceability is extended to include web pages and BBS for real time communication.
- **It makes the entire software development process visible from first step down to the final step** – the NSE visualization paradigm is capable of making the entire software development process visible through dummy programming and reverse engineering.
- **It makes a software product much easier to read, understand, test, and maintain** – with NSE a software is represented graphically and shown in both the overall structure of the entire product and the detailed logic diagram and control flow diagram with various traceabilities and that the untested conditions and branches are highlighted.
- **It can be applied at any time in any stage for a software product development using any other method originally** – NSE can be added onto a software product being developed using any other approach by adding bookmarks in the related documents and modifying the test cases to use some keywords to indicate the format of a document and the file path plus the bookmark, then other work can be performed by the NSE support platform automatically.
- **It requires much less time, resources, and manpower to apply, compared with other existing approaches** – one just needs to re-organize the document hierarchy using bookmarks and modifying the test case description using some simple rules; all of the other work can be performed automatically by the NSE support platform with many automated and intelligent tools integrated together, including the creation of huge amount of traceable and virtual documents based on static and dynamic measurement of the software, the diagramming of the entire software product to generate holistic and detailed system call graphs and class inheritance charts, the holistic and detailed test coverage measurement results shown in J-Chart and J-Diagram or J-Flow diagram with untested conditions and branches highlighted, the holistic and detailed quality measurement results shown in Kiviati diagram for the entire software product and each class or function, the holistic and detailed performance measurement results shown in J-Chart and bar chart with branch execution frequency measurement result shown in J-Diagram or J-Flow Diagram to locate the performance bottleneck better, the software logic analysis results shown in J-Diagram with various kinds of traceability for semi-automated code inspection and walk through, the software control flow analysis results shown in J-Flow with untested conditions and branches highlighted, the GUI test operation capture and selective playback for regression testing after code modification, the test case efficiency analysis and test case minimization to form a minimized set of test cases to replace the all test cases to speed up the regression testing process and greatly save the required time and resources, the establishment of bidirectional traceability among all related documents and the test cases and the source code, the generation of more than 100 reports based on

the static and dynamic measurement of the software – the reports can be stored in HTML format for being used on the internet, the Cyclomatic complexity measurement results shown in J-Chart and J-Flow diagram for performing refactoring on the over complicated modules to reduce possible defects, and more.

- **It is possible for NSE to help software organizations double their productivity, halve their cost, and reduce 99.99% defects in their software products** – with NSE the quality of a software product is ensured from the first step through defect prevention and defect propagation prevention rather than testing after coding, so that the amount of defects introduced into a software product is greatly reduced, and that the defects propagating to the maintenance phase are also greatly reduced; the software maintenance is performed holistically and globally with side-effect prevention; the regression testing after software modification is performed using a minimized test case set and some test cases selected through backward traceability from the modified modules and branches; software testing is performed in the entire software development process dynamically using the Transparent-method which combines functional testing and structural testing together seamlessly, and can be dynamically used in the case that there is no a real output in running some test cases, when it is used in the requirement development phase and the software design phase.

5. Applications

Theory comparison and preliminary applications show that compared with the old one it is possible for Silver Bullet to help users double their productivity and project success rate, halve their cost, and remove 99.99% of the defects in their software products.

(a) Efficiently Solving the Issue of Missed Schedules

- (1) Helping the project development team and the customer work together closely to assign priority to requirements according to the importance (see the preprocess part shown in Fig. 1), so that the important requirements will be implemented early to meet the market needs. If necessary some optional requirements can be temporally ignored .
- (2) Making the project plan, the schedule chart and other related documents traceable with the implementations of requirements and the source code as shown in Fig. 6, so that the management team can find and solve the schedule issue in time.
- (3) Helping the software development team set a project web site and technical forum, and making the web pages and the topic pages of the technical forum traceable to the implementations of requirements and the source code, so that any schedule delay will be known by the members of the team, and each member may make his/her contribution to solve the issue quickly – see Fig. 7 an application example.
- (4) See section (c) “Efficiently Solving the Issue of Flawed Products – Removing More Than 99.99% of the Defects” – through greatly reducing the amount of

defects to help the development team much easy to meet the project development schedule.

- (5) See section (d) “How Is It Possible for NSE to Help Users Double Their Productivity” - through defect prevention and defect propagation prevention in upstream to greatly reduce the defects propagated into the downstream, and side-effect prevention in the implementation of requirement changes and code modifications to make it possible to reduce 2/3 of the total effort spent in software changes and maintenance to help the development team to meet the project development schedule better.

(b) Efficiently Solving the Issue of Blown Budgets

- (1) Assigning priority to the requirements according to the importance ((a) must have, (b) should have, (c) better to have, (d) may have or optional...) to make the critical and important requirements be implemented early to form an essential working version (about 20% of the requirements) first, then making the working product grow up incrementally according to the assigned priority (see Fig. 8 and Fig. 9), to avoid the issue of blown budgets – if necessary some optional requirements can be ignored or implemented in the future.
- (2) Complying with the Generative Holism principle of complexity science, helping users to form the whole of a software product first through dummy programming as an embryo through the use of HAETVE (Holistic, Actor-Action and Event-Response drive, Traceable, Visual, and Executable) technique for requirement development, and the **Synthesis Design and Incremental growing up (Implementation and Integration)** Technique for product design, to help users estimate the cost/budget better.
- (3) Making the cost estimation chart, the budget plan, and other related documents traceable with the requirement implementation and the source code, so that the management team can know the situation in time and control the budget better.
- (4) Making the web pages or topic pages of the technical forum traceable to the implementations of requirements and the source code, so that any budget issue can be known by the members of the team early, and each member may make his/her contribution to solve the issue quickly.
- (5) Helping users to make the product grow up incrementally, according to the requirement priority.
- (6) See section **(c)** “Efficiently Solving the Issue of Flawed Products – Removing More Than 99.99% of the Defects” – through greatly reducing the amount of defects to help the development team much easy to develop the product within the budget.
- (7) See section (d) “How Is It Possible for NSE to Help Users Double Their Productivity” - through defect prevention and defect propagation prevention in upstream to greatly reduce the defects propagated into the downstream, and side-effect prevention in the implementation of requirement changes and code modifications to make it possible to reduce 2/3 of the total effort spent in software maintenance to help the

development team develop the product within the budget better.

- (8) See section (e) “How Is It Possible for NSE to Help Users Halve Their Cost” – through greatly reducing the cost to further ensuring the product being developed under the budget.

(c) Efficiently Solving the Issue of Flawed Products – Removing More Than 99.99% of the Defects mainly through Defect Prevention and Defect Propagation Prevention

- (1) Helping users efficiently remove defects particularly upstream defects through
- * defect prevention by (a) providing some templates such as requirement specification template (see appendix A) to prevent something missing; (b) helping users apply the HAETVE technique for requirement development though dummy programming and making the dummy program executable through dynamical testing using the Transparent-box method combining functional and structural testing together seamlessly, can be used dynamically in the entire software development lifecycle; (c) supporting incremental coding to prevent inconsistency between the interfaces;
 - * defect propagation prevention mainly through dynamic testing using the Transparent-box testing with capability to perform MC/DC (Modified Condition/Decision Coverage) test coverage measurement, memory leak and usage violation check, performance analysis, and the capability to automatically establish bidirectional traceability to help users check and remove the inconsistency defects among the related documents and the source code, plus inspection using traceable documents and source code.
 - * refactoring for those modules with higher Cyclomatic complexity (the number of decision statements) and performance bottleneck modules with side-effect prevention – often 20% higher complex modules have about 80% of the defects.
- (2) supporting quality assurance from the first step to the end through dynamic testing using the Transparent-box method;
- (3) providing techniques and tools for quality measurement to the entire software product and each component for finding and solving the quality problems in time.
- (4) helping users perform software maintenance holistically and globally with side-effects prevention though various bidirectional traceability.
- (5) see section (f) “How Is It Possible for NSE to Help Users Reduce the Risk” and section (g) “Efficiently Handling the Issue of Changeability” for more information about quality assurance with NSE.

(d) How Is It Possible for NSE to Help Users Double Their Productivity

- (1) With the old-established software engineering paradigm, linear process models are used and dynamic testing is performed after coding, so that defects are easy introduced into a software product in upstream, and the defects are easy to propagate to the

maintenance phase in which the implementation of requirement changes and code modifications are performed partially and locally, so that software maintenance is very difficult to perform – usually takes 75% or more of the total efforts in a software development; But with NSE, nonlinear NSE process model is used which combines software development process and maintenance process together, ensures software quality from the first step down to the final step through defect prevention, defect propagation prevention, refactoring, and software testing dynamically using the Transparent-box method in the entire software system development lifecycle, so that the defects propagated into maintenance phase are greatly reduced, plus that the implementation of requirement changes and code modifications are performed holistically and globally with side-effect prevention – the result is that the effort spent in software maintenance will be almost the same as that spent in the software development process, it means about 2/3 efforts originally spent in software maintenance can be saved – about half of the total effort can be saved (equal to double the productivity).

- (2) As described in section (c), with NSE about 99.99% of the defects can be removed. So that as Capers Jones pointed, “Focus on quality, and productivity will follow”[Jon94].
- (3) NSE also supports the reuse of qualified components to increase software productivity.
- (4) With NSE software documentation paradigm and NSE software visualization paradigm, software document and source code are traceable, making a software product much easy to read, understand, test, and maintain to increase the productivity.
- (5) With NSE there are more means to help users increase their productivity:
 - Provides techniques and automated tools to help users manage and control their software projects better
 - Provides automated tools and templates for helping users execute their project development plan easily
 - Provides techniques and visual tools to help users perform requirement development, product design, and bug fixing quickly
 - Supports reverse engineering to generate a lot of design documents automatically
 - Supports incremental and visual coding
 - Provides techniques and automated complexity analysis tools to help users design their test plan quickly
 - Provides techniques and tools to help users perform test case design efficiently through un-executed path analysis
 - Provides techniques and tools for capturing GUI operation and playing back automatically
 - Provides techniques and automated tools for test case efficiency analysis and test case minimization, to help users perform regression test quickly (at least 5 times fast)
 - Provides techniques and automated tools for incremental data base management, so that unchanged source files do not need to analyze twice to speed up the regression process (10 times faster than other tools without incremental data base management capability).

- Provides techniques and automated tools to analyze the system structure, data usage, logic flow of a users’ software product to help them manage the product better
- Provides intelligent version comparison tools to help users maintain their product versions easier.

(e) How Is It Possible for NSE to Help Users Halve Their Cost

- (1) All of the techniques and tools used for helping users double their productivity are also useful for reducing the software development cost.
- (2) All techniques and tools provided for reduce 99.99% of the bugs are also useful for reducing the software development cost.
- (3) With the old-established software engineering paradigm, software maintenance takes 75% or more of the total cost in a software development; But with NSE, nonlinear NSE process model is used which combines software development process and maintenance process together, ensures software quality from the first step down to the final step through defect prevention, defect propagation prevention, refactoring, and software testing dynamically using the Transparent-box method in the entire software system development lifecycle, so that the defects propagated into maintenance phase are greatly reduced, plus that the implementation of requirement changes and code modifications are performed holistically and globally with side-effect prevention – the result is that the effort spent in software maintenance will be almost the same as that spent in the software development process, it means about 2/3 cost originally spent in software maintenance can be saved – about half of the total cost can be saved as shown in Fig. 10.
- (4) Provides techniques and tools to diagram the entire system of a users’ product, links the related parts each other, making code inspection and walkthrough much easier to perform.
- (5) Supports efficient regression testing using minimized test cases.
- (6) Provides techniques and tools to capture users’ GUI operations, and play them back to reduce regression test cost, plus
 - Provides techniques and visual tools to help users quickly perform requirement development, functional decomposition, and bug fixing
 - Supports reverse engineering to automatically generate design documents
 - Supports incremental and visual coding
 - Provides automated tools for complexity analysis to help users design their test plan rapidly
 - Provides tools to help users perform efficient test-case design
 - Provides techniques and tools for capturing GUI operations and playing them back
 - Provides techniques and automated tools for test-case efficiency analysis and test case minimization
 - Provides techniques and tools to diagram the entire system of a user’s software product for immediate product comprehension and understanding
 - Provides techniques and automated tools to analyze the system structure, data usage, and

- logic flow of users' software products for better product management
- Provides intelligent version comparison tools to help users maintain their product versions effortlessly
 - Provides forward and backward traceability among requirement specifications, design documents, test cases, source code, and test cases, making the software product easier to understand, test, and maintain

(f) How Is It Possible for NSE to Help Users Reduce the Risk

- (1) Helping users work with the customer to assign priority order to requirements according to the importance for making the important requirements be implemented early.
- (2) Helping users perform prototype design and test for important and unfamiliar requirements to prevent unrealized requirements.
- (3) Helping users estimate the cost better using the designed dummy system through dummy programming.
- (4) Making it possible to help users remove 99.99% of the defects in the designed product, double their productivity, halve their cost – further reducing the risk.

6. Conclusion

Silver Bullet is a qualified solution for slaying software Werewolves efficiently.

References

- [1] Brooks, Fredrick P., "No Silver Bullet: Essence and Accidents of Software Engineering," *Computer*, Vol 20, No 4 (April 1987), pp. 10-19
- [2] Jones, Capers, Social and Technical Reasons for Software Project Failures, *CrossTalk*, Jun 2006 Issue
- [3] Condensed GSAM Handbook, chapter 2, *CrossTalk*, 2003
- [4] Deming W E. *Out of the Crisis*. MIT Press, 1986.
- [5] Jay Xiong, Tutorial, *A Complete Revolution in Software Engineering Based on Complexity Science*, WORLDCOMP'09 -, Las Vegas, July 13-17, 2009.
- [6] Jay Xiong, Jonathan Xiong, *A Complete Revolution in Software Engineering Based on Complexity Science*, WORLDCOMP'09 – SERP (*Software Engineering Research and Practice 2009*), 109-115.

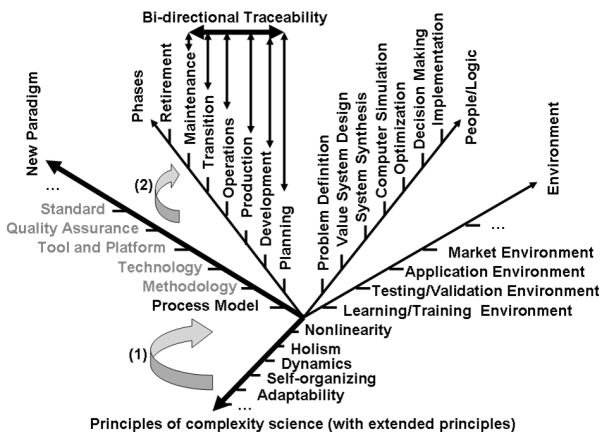
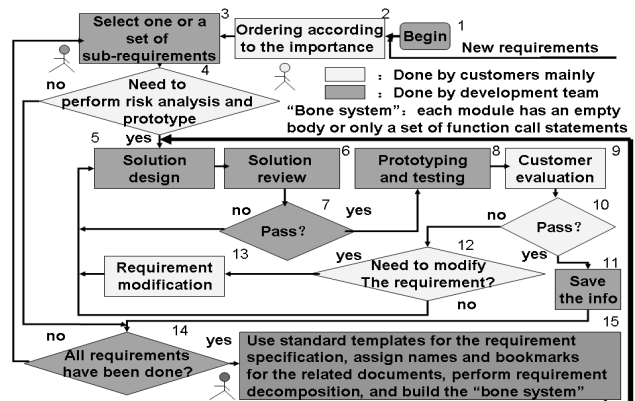
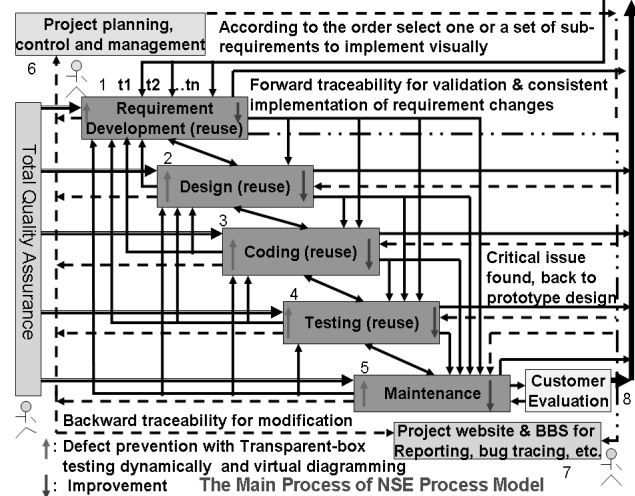


Fig. 1 The paradigm-shift framework



The Pre-process of the NSE Model



The Main Process of NSE Process Model

Fig. 2 The Silver Bullet process model

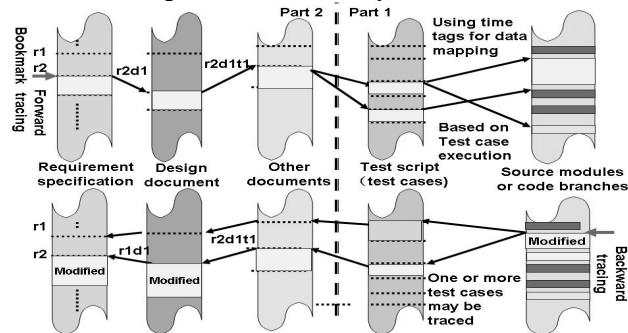
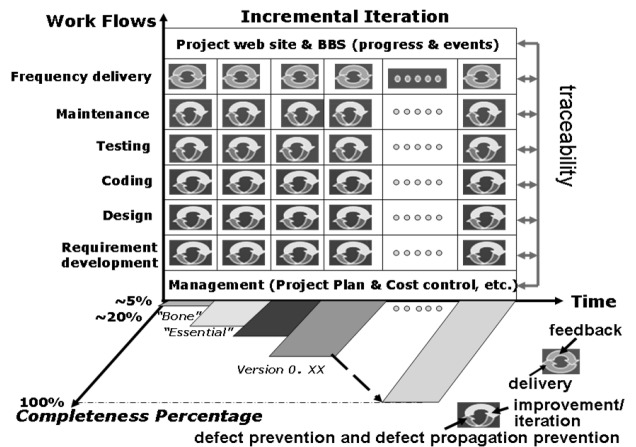


Fig. 3 The automated and self-maintainable traceability The NSE Software Development Methodology



A Software Engineering Tool for Pedagogy: Reporting Performance Test Results

J. Crunk, M.M. North, and S.M. North
Visualization & Simulation Research Center
Southern Polytechnic State University, Georgia, United States of America

Abstract—As an innovative and useful tool of the software engineering pedagogy, performance testing may be conducted to show how a system will perform in a post production environment. It is common to report the results of the test as a quantitative result; this often leaves the reader of the results with their own interpretation which can be subjective. The purpose of this paper is to discover if there is a way to give the same information as a result of the performance test to the people that need it, but in a mixture of qualitative and quantitative methodologies. It is believed that by providing this information, the results of the performance test activities will be better supported and less likely to be misinterpreted. In turn, it provides a useful tool for software engineering pedagogy.

Index Terms— Performance Test Reports, Software Engineering, Pedagogy,

I. INTRODUCTION

It is common to use varieties of tools to teach software engineering concepts, of most interesting and effective tools are the innovative and useful ones. For instance, a mixture of qualitative and quantitative methodologies integrated in reporting software performance testing, is described in this paper.

A performance test is run and the results are reported giving the users information about the test, the results reported show numbers that range from less than a second to a minute for certain actions (Mar 2007, Pully 2007, Tyagi 2007). One user of the system will determine that the results are too slow and that something is wrong, while another user will accept the results not knowing that there is a problem. It is proposed that part of the reason for this is that the results are quantitative in nature and do not give the users information about what is really seen in the application when the final report is complete. Allowing the users to see qualitative information in the results will allow them to know exactly what is happening during the application and actually be able to see them doing a specific action. Users need to be able to see the

action performed to make better decisions about what is going on; qualitative reporting of results is how they can see this.

To accomplish this, information about how people are currently gathering results and presenting them will be collected from several experts in the field. These results are expected to be presented in a quantitative way since performance testing is primarily about the time that it takes to complete an action.

Once the current method that experts are using is established, a new methodology will be determined from this that integrates qualitative reporting methods as a part of the presentation into the final results. It is expected that this will be more informative to the experts in the field. Finally, this information will be given back to these experts for consideration and possible further modification.

II. REPORTING METHODS

Reporting methods have been adopted to help in making clear different ways of presenting material. There are three types of reporting methods that will be discussed here; they are qualitative, quantitative, and mixed methods.

A. Quantitative Method

The quantitative method of reporting is one that involves numeric description of trends, attitudes, or opinions of a population by studying a sample of that population (Creswell 2003). We see this when reporting results from performance testing too. During a performance test run, we find that numbers are captured and logged, in the end; these numbers are collected into a presentation of charts and graphs that are presented to people with business interest in the project for review.

B. Qualitative Method

Qualitative procedures employ different claims, strategies of inquiry, and methods than qualitative procedures rely on. The processes are similar, but they have unique steps in data analysis, and

draw on diverse strategies of inquiry (Creswell 2003). Qualitative research takes on some of these characteristics according to Creswell (2003).

- 1) Research takes place in a natural setting
- 2) Research uses multiple methods that are interactive and humanistic
- 3) Research is emergent rather than tightly prefigured
- 4) Research is fundamentally interpretive
- 5) The researcher views social phenomena holistically
- 6) The researcher systematically reflects on who he or she is in the inquiry and is sensitive to his or her personal biography and how it shapes the study.

As can be seen here, many of these aspects of qualitative procedures are also the goals of reporting the results when a performance test is run. For this reason, it only makes sense to include this information in the performance test.

III. REPORTS CURRENTLY

Reports that are currently presented after a performance test has been run typically concentrate on numerical results as the chart in figure 1. Results like this one simply collect numbers and then present them in this form at the end.

Test results like these are subjective and leave much to interpretation. Names can mislead the users to thinking that results are one thing when they are something else and numbers reflected can indicate different information to different users. Solutions to this problem are easy, but take some work to develop.

Other problems that confuse the readers of these reports are the presentation of too much information at one time. When too much information is presented in a report, it only serves as garbage as is depicted in figure 2. Graphs like this one depict too much information to be processed, and even though there are colors for the separate information that is portrayed, it is not enough. This graph does not provide useful information to the audience.

IV. SOLUTIONS IN REPORTING

There are some solutions that can make the information that is presented during testing more relevant and meaningful. These solutions are found in quantitative reporting methods. The question is, how can we integrate this reporting methodology into something that is typically qualitative?

We can find answers to this by looking to the definition of qualitative. Qualitative has some shared aspects about it that is the goal of performance testing. Those shared components are Creswell (2003):

- 1) Testing takes place in a natural setting
- 2) Testing uses multiple methods that are interactive and humanistic
- 3) Testing is emergent rather than tightly prefigured
- 4) Testing is fundamentally interpretive
- 5) The tester views social phenomena holistically
- 6) The tester systematically reflects on who he or she is in the inquiry and is sensitive to his or her personal biography and how it shapes the study.

Notice that we have replaced the word research with test and come up with a meaning that seems to fit both. Reporting on this is the next logical step. We must not create a report that demonstrates the ability to show what is going on during our test.

One such way of introducing qualitative information into a report of this nature is a transaction traversal table (Tyagi 2007) as can be seen in figure 3. This table outlines exactly what is occurring in the application at each step. Follow this table the user should be given the information as presented above so that they are able to clearly understand what occurred in the application and what they are being told. Figure 4 (Badi 2007) shows the quantitative information that coincides with the transaction traversal table.

As can be clearly seen, the readers of this report can see exactly what is happening in the application while they are looking at the numbers. Improvements in the system are reported and bottlenecks were identified. It is also clear where further improvements are needed. The Homepage is still a problem and needs further evaluation based on clear identification of that page from figure 3.

Is this enough though to show users of a given system exactly what will be going on in their application during performance testing? It is proposed that this is probably not sufficient to give the audience of the test results what they are looking for. Numbers are great, telling them what is happening while the test run is better, but in the end; showing them what happened is best.

A current project that I was a part of had contracted a company to perform the performance test for us. This application was a VRU (Voice Response Unit) where they would dial numbers for us under load. We could log into a web site while the test was running and not only see the results of the system under load, but we could listen to the actual calls that were taking place. We could actually hear what happened on the calls and even drill down on where problems occurred so that we could fix the problems.

This is real results; a test that gives the user the ability to see the results as they occurred is what the readers want to see. In the case of the application mentioned earlier, they want to view the screens that displayed for each user as they occurred. If they error,

they don't want to see a log of the error, they want to see the error message on the screen when it happened. This gives weight to the expression, "seeing is believing". If they can't see the conditions as they would as a user of the system, then it cast doubt on whether it really occurred or not.

V. DISCUSSION AND CONCLUSION

We have evaluated qualitative and quantitative reporting methods of performance test results. It has been proposed that there is a large gap in the way that reports have typically been done that serve to confuse the reader of reports.

As a result, it is proposed that the reports include information that shows the audience those reports are directed toward more about what is going on in the application and not just the numbers that were captured. Flows of operation through transaction traversal tables and actual interface captures of the application are both helpful in giving valuable information to the readers.

Some problems may arise in attempting to give some of this information through the reports. Current reporting methods will need to be researched and modified to allow an interface that the audience can navigate through and view as the desire. Some of this is available on a limited scale, but other methods will need to be developed for further abilities.

ACKNOWLEDGMENT

This effort was partially supported by a grant from the DoD/National Security Agency. The content of this work does not reflect the position or policy of the DoD and no official endorsement should be inferred.

REFERENCES

- Creswell, J. W. (2003). *Research design: Qualitative, quantitative, and mixed methods approaches* (2nd ed.). Thousand Oaks, CA: Sage Publications. ISBN: 0761924426.
- Trochim, W. M. K. (2006). *Introduction to validity*. Retrieved August 15, 2007, from <http://www.socialresearchmethods.net/kb/introval.php>
- Trochim, W. M. K. (2006). *Qualitative validity*. Retrieved August 15, 2007, from <http://www.socialresearchmethods.net/kb/qualval.htm>
- California.com (2003), Performance Testing, Retrieved 11/10/07 from <http://www.bestpractices.cahwnet.gov/.%5Cdownloads%5CTopic%20-%20performance%20testing.pdf>
- Loadrunner User Group, (2007), Loadrunner User Group, Retrieved 11/10/07 from <http://tech.groups.yahoo.com/group/LoadRunner/?yguid=165069635>
- Pully, J. (2005), Handicaped Results, Sent via Email from James Pully
- Mar, W. (2007), Performance Test Results, Retrieved 11/10/07 from <http://www.wilsonmar.com/1loadrun.htm>
- Mar, W. 2007, Correspondence with Wilson Mar via Email
- Tyagi, A., (2007) Correspondence with Ashish Tvagi via Email
- Podelko, A., 2007, Multiple Dimensions of Performance Requirements, Sent from Alexander Podelko via Email.
- AgileTesting, (Feb 2005), Blog Spot, Performance Testing, Retrieved 11/23/07 from <http://agiletesting.blogspot.com/2005/02/performance-vs-load-vs-stress-testing.html>
- Badi, K. (2007), Email correspondence with Kiran along with documents sent to me from him.

Color	Scale	Measurement	Minimum	Average	Maximum	Std. Deviation
Green	1	QAS_1_LaunchQAS	0.461	3.542	70.782	4.727
Yellow	1	QAS_2_PostalCodeSearch	0.17	1.669	71.553	3.272
Purple	1	QAS_3_SelectPostSearchResultSet	0.2	1.894	64.282	3.452
Grey	1	QAS_4_StreetSearch	0.17	1.466	69.81	2.645
Cyan	1	QAS_5_SelectStreetSearchResultSet	0.25	1.975	62.73	3.564
Black	1	QAS_6_BuildingNumberSearch	0.18	1.488	59.956	2.476
Red	1	QAS_7_SelectBuildingSearchResultSet	0.17	1.572	66.536	2.51

Figure 1. Test Results in numerical form

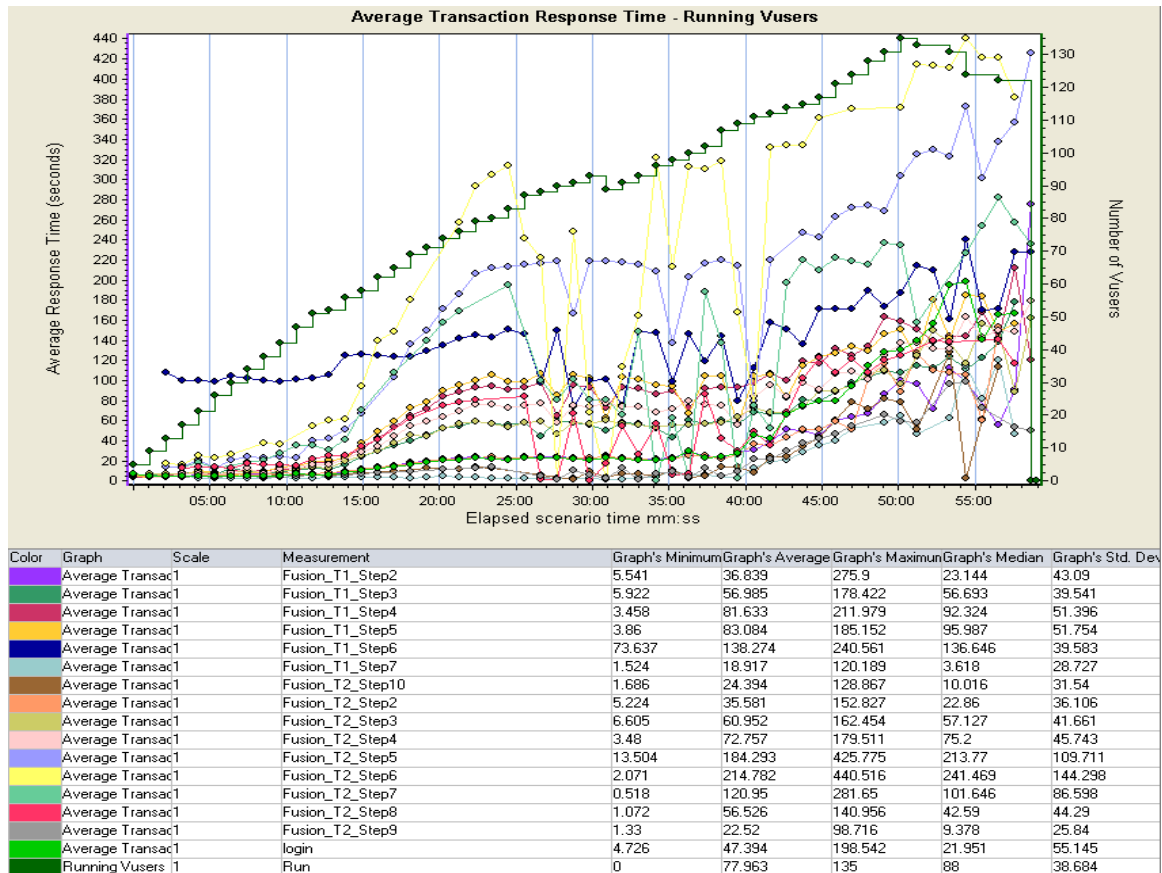


Figure 2. A graph with overwhelming presentation of information

View Current Development Plan	
Homepage	Open the home page using student link
Log On	Log on to the system using student id
Open Course	Launch AA course
Navigate to page 2	Click on next button on course
Close course first time	Close course window
Open Course second time	Launch course second time
Navigate to last page	Click on next button on all pages
Click Completed Button	Click on Completed Radio Button to view completed course
Logout	Logout of the system

Figure 3. Transaction Traversal Table

Transaction Name	Initial run		After tuning		Improvement %	
	Average	90%	Average	90%	Average	90%
Homepage	13.93	14.54	4.888	5.278	65%	64%
Logon	6.904	7.637	4	4.16	42%	46%
OpenCourse	5.702	7.745	2.995	3.285	49%	58%
NavigateToPage2	10.387	13.122	5.051	5.392	51%	59%
CloseCourseFirstTime	11.511	12.368	4.822	5.033	58%	59%
OpenCourseSecondTime	20.344	21.654	5.073	5.394	75%	75%
NavigateToLastPage	17.386	17.858	0.252	0.275	98%	98%
ClickCompletedButton	8.734	8.903	0.166	0.188	98%	98%

Figure 4. Transaction Results for Transaction Traversal Table

Using Neural Network for Security Analysis in Software Design

A. Adebisi, Johnnes Arreymbi and Chris Imafidon
 School of Architecture, Computing and Engineering,
 University of East London,
 London, UK

adetunjib@hotmail.com, J.Arreymbi@uel.ac.uk, C.O.Imafidon@uel.ac.uk

Abstract: Security is often considered as afterthought for many developers whose primary aim is to produce software quickly and release it into the market under tight deadlines. However Integrating security into software applications after deployment or at a later stage of software development lifecycle (SDLC) has been found to be more costly than when it is integrated during the early stages of SDLC. Previous research shows that design flaws still outnumber all other types of security flaws. Therefore designing software with security in mind will go a long way in developing secure software. In this paper, a new method of analysing security in software design by using neural network trained to identify attack patterns is presented. This research found out that the neural network was able to match attack patterns to software design scenarios presented to it. The result of performance of the neural network is presented in this paper.

Keywords- *Neural Networks, Software security, Attack Patterns*

I. INTRODUCTION

Software security flaws have been attributed to defects unintentionally introduced during SDLC especially during the design and the implementation phase. Therefore, it is now an on-going challenge for the software industries to look into ways through which software defects can be reduced during SDLC in order to produce more secured software. As reliance on network security and other host based security measures to protect software no longer provide adequate security, a new research field called software security emerged in the last decade with the aim of building security into software application during development. In this approach, security is viewed as an emergent property of the software application under development. With this in mind, security is considered all through SDLC

One of the critical areas in this approach is the area of software design and security which proactively deals with attacking security problems at the design phase of SDLC. Reportedly, 50% of security problems in software products today have been found to be design flaws [17]. Design-level vulnerability has been described as the hardest category of software defect to contend with. Moreover, it requires great expertise to ascertain whether or not a software application has design-level flaws which makes it difficult to find and automate [9]. Many authors also argue that it is much better to find and fix flaws during the early phase of software development because it is more costly to fix the problem at a

late stage of development and much more costly when the software has been deployed [6][29][30]. Therefore, taking security into consideration at the design phase of SDLC will help greatly in producing secured software applications.

There is much advancement in the tools currently used for integrating security at the implementation phase of SDLC. However, software design security tools and technologies for automated security analysis at the design phase have been slow in coming. This is still an area where many researches are currently being undertaken. Neural Networks has been one of the technologies used during software implementation and testing phase of SDLC for software defect detection in order to intensify software reliability and it has also been used in area of application security and network security in technologies such as cryptography and intrusion detection systems (IDS). This research takes a further step by using neural networks as a tool for analysing security of software design at the design phase of SDLC.

II. CURRENT APPROACHES IN INTEGRATING SECURITY INTO SOFTWARE DESIGN

Different approaches are currently being used in the software industry to integrate security into software design. Some of these approaches are discussed below.

Architectural risk analysis is used to identify vulnerabilities and threats at the design phase of SDLC which may be malicious or non-malicious in nature due to a software system. It examines the preconditions that must be present for the vulnerabilities to be exploited by various threats and assess the states the system may enter after a successful attack on the system. One of the advantages of architectural risk analysis is that it enables developers to analyse software system from its component level to its environmental level in order to evaluate the vulnerabilities, threats and impacts at each level [17].

Threat modelling is another important activity carried out at the design phase to describe threats to the software application in order to provide a more accurate sense of its security [1]. Threat modelling is a technique that can be used to identify vulnerabilities, threats, attacks and countermeasures which could influence a software system [18]. This allows for the anticipation of attacks by understanding how a malicious attacker chooses targets, locates entry points and conducts attacks [24]. Threat modelling addresses threats that have the ability to cause maximum damage to a software application.

Attack trees is another approach used to characterize system security by modelling the decision making process of attackers. In this technique, attack against a system is represented in a tree structure in which the root of the tree

represents the goal of an attacker. The nodes in the tree represent the different types of actions the attacker can take to accomplish his goal on the software system or outside the software system which may be in the form of bribe or threat [6],[23]. "Attack trees are used for risk analysis, to answer questions about the system's security, to capture security knowledge in a reusable way, and to design, implement, and test countermeasures to attacks" [24].

Attack nets is a similar approach which include "places" analogous to the nodes in an attack tree to indicate the state of an attack. Events required to move from one place to the other are captured in the transitions and arcs connecting places and transitions indicate the path an attacker takes. Therefore just as attack trees, attack nets also show possible attack scenarios to a software system and they are used for vulnerability assessment in software designs [6].

Another related approach is the vulnerability tree which is a hierarchy tree constructed based on how one vulnerability relates to another and the steps an attacker has to take to reach the top of the tree [23]. Vulnerability trees also help in the analysis of different possible attack scenarios that an attacker can undertake to exploit a vulnerability.

Mouratidis and Giorgini [19] also propose a scenario based approach called Security Attack Testing (SAT) for testing the security of a software system during design time. To achieve this, two sets of scenarios (dependency and security attack) are identified and constructed. Security test cases are then defined from the scenarios to test the software design against potential attacks to the software system. Essentially SAT is used to identify the goals and intention of possible attackers based on possible attack scenarios to the system. Software engineers are able to evaluate their software design when the attack scenarios identified are applied to investigate how the system developed will behave when under such attacks. From this, software engineers better understand how the system can be attacked and also why an attacker may want to attack the system. Armed with this knowledge, necessary steps can be taken to secure the software with capabilities that will help in mitigating such attacks

For most of the approaches discussed above, the need to involve security experts is required in order to help in identifying the threats to the software technology, review the software for any security issues, investigate how easy it is to compromise the software's security, analyse the impact on assets and business goals should the security of the software be compromised and recommend mitigating measures to either eliminate the risk identified or reduce it to a minimum. The need for security experts arises because there is an existing gap between security professionals and software developers. The disconnection between this two has led to software development efforts lacking critical understanding of current technical security risks [22].

In a different approach, Kim T. et.al [12] introduced the notion of dynamic software architecture slicing (DSAS) through which software architecture can be analysed. "A dynamic software architecture slice represents the run-time behaviour of those parts of the software architecture that are selected according to a particular slicing criterion such as a set

of resources and events" [12] DSAS is used to decompose software architecture based on a slicing criterion. "A slicing criterion provides the basic information such as the initial values and conditions for the ADL executable, an event to be observed, and occurrence counter of the event" [12] While software engineers are able to examine the behaviour of parts of their software architecture during run time using the DSAS approach, the trade-off is that it requires the software to be implemented first. The events examined to compute the architecture slice dynamically are generated when the Forward Dynamic Slicer executes the ADL executable. This is a drawback because fixing the vulnerability after implementation can be more costly [6].

Howe [10] also argues that the industry needs to invest in solutions that apply formal methods in analysing software specification and design in order to reduce the number of defects before implementation starts. "Formal methods are mathematically based techniques for the specification development and verification of software and hardware systems" [7] Recent advances in formal methods have also made verification of memory safety of concurrent systems possible [7]. As a result, formal methods are being used to detect design errors relating to concurrency [10]. A software development process incorporating formal methods into the overall process of early verification and defects removal through all SDLC is Correct by Construction (CbyC) [24]. CbyC has proved to be very cost effective in developing software because errors are eliminated early during SDLC or not introduced in the first place. This subsequently reduces the amount of rework that would be needed later during software development. However, many software development organizations have been reluctant in using formal methods because they are not used to its rigorous mathematical approach in resolving security issues in software design. Model checkers also come with their own modelling language which makes no provision for automatically translating informal requirements to this language. Therefore, the translation has to be done manually and it may be difficult to check whether the model represent the target system [21]

III. THE NEURAL NETWORK APPROACH

Our proposed Neural Network approach in analysing software design for security flaws is based on the abstract and match technique through which software flaws in a software design can be identified when an attack pattern is matched to the design. Using the regularly expressed attack patterns proposed by Williams and Gegick [6], the actors and software components in each attack pattern are identified. To generate the attack scenarios linking the software components and actors identified in the attack pattern, online vulnerability databases were used to identify attack scenarios corresponding to the attack pattern. Data of attack scenarios from online vulnerability databases such as CVE Details, Security Tracker, Secunia, Security Focus and The Open Source Vulnerability Database were used.

From the online vulnerability databases a total of 715 attack scenarios relating to 51 regularly expressed attack patterns by Williams and Gegick's were analysed. This consisted of 260 attack scenarios which were unique in terms of their impact,

mode of attack, software component and actors involved in the attack and 455 attack scenarios which are repetition of the same type of exploit in different applications they have been reported in the vulnerability database. The attacks were analysed to identify the actors, goals and resources under attack. Once these were identified the attack attributes below were used to abstract the data capturing the attack scenario for training the neural network. The attack attributes includes the following.

1. **The Attacker:** This attribute captures the capability of the attacker. It examines what level of access possessed when carrying out the attack.
2. **Source of attack:** This attributes captures the location of the attack during the attack.
3. **Target of the attack:** This captures the system component that is targeted by the attacker
4. **Attack vector:** This attributes captures the mechanism (i.e. software component) adopted by the attacker to carry out the attack
5. **Attack type:** The security property of the application being attacked is captured under this attribute. This could be confidentiality, integrity or availability.
6. **Input Validation:** This attributes examines whether any validation is done on the input passed to the targeted software application before it is processed
7. **Dependencies:** The interaction of the targeted software application with the users and other systems is analysed under this attributes.
8. **Output encoding to external applications/services:** Software design scenarios are examined under this attributes to identify attacks associated with flaws due to failure of the targeted software application in properly verifying and encoding its outputs to other software systems
9. **Authentication:** This attribute checks for failure of the targeted software application to properly handle account credentials safely or when the authentication is not enforced in the software design scenarios.
10. **Access Control:** Failure in enforcing access control by the targeted software application is examined in the design scenarios with this attribute.
11. **HTTP Security:** Attack Scenarios are examined for security flaws related to HTTP requests, headers, responses, cookies, logging and sessions with this attribute
12. **Error handling and logging:** Attack scenarios are examined under this attributes for failure of the targeted application in safely handling error and security flaws in log management.

A. The Neural Network Architecture

A three-layered feed-forward back-propagation was chosen for the architecture of neural network in this research. The back-propagation neural network is a well-known type of neural network commonly used in pattern recognition problems [25]. A back-propagation network has been used because of its simplicity and reasonable speed. The architecture of the neural network consists of the input layer, the hidden layer and the output layer. Each of the hidden nodes and output nodes apply a tan-sigmoid transfer function ($2/(1+\exp(-2*n))-1$) to the various connection weights.

The weights and parameters are computed by calculating the error between the actual and expected output data of the neural network when the training data is presented to it. The error is then used to modify the weights and parameters to enable the neural network to have better chance of giving a correct output when it is next presented with same input.

B. Data Encoding

The training data samples each consist of 12 input units for the neural network. This corresponds to the values of the attributes abstracted from the attack scenarios. The training data was generated from the attack scenarios using the attributes. For instance training data for the attack on webmail (CVE 2003-1192) was generated by looking at the online vulnerability databases to get its details on the attributes we are interested in. This attack corresponds to regularly expressed attack pattern 3. Williams and Gegick [6] describe the attack scenario in this attack pattern as a user submitting an excessively long HTTP GET request to a web server, thereby causing a buffer. This attack pattern is:

(User)(HTTPServer)(GetMethod) (GetMethodBufferWrite)(Buffer)

Table 1: Sample of Pre-processed training data from attack scenario

S\N	Attribute	Observed data
1	Attacker	No Access
2	Source	External
3	Target	Buffer
4	Attack Vector	Long Get Request
5	Attack Type	Availability
6	Input Validation	Partial Validation
7	Dependencies	Authentication & Input Validation
8	Output Encoding	None
9	Authentication	None
10	Access Control	URL Access
11	HTTP Security	Input Validation
12	Error	None

In this example, the data generated from the attack scenario using the attribute list is shown in Table I. Using the corresponding values for the attributes; the data is then encoded as shown in the Table II.

Table II: Sample of Training data after encoding

S\N	Attribute	Value
1	Attacker	0
2	Source	1
3	Target	9
4	Attack Vector	39
5	Attack Type	5
6	Input Validation	2
7	Dependencies	6
8	Output Encoding	0
9	Authentication	0
10	Access Control	2
11	HTTP Security	3
12	Error	0

The second stage of the data processing involves converting the value of the attributes in Table II into ASCII comma delimited format before it is used in training the neural network. For the expected output from the neural network, the data used in training network is derived from the attack pattern which has been identified in each of the attack scenarios. Each attack pattern is given a unique ID which the neural network is expected to produce as an output for each of the input data samples. The output data sample consists of output units corresponding to the attack pattern IDs. For instance, the above sample data on Webmail attack which corresponds to regularly expressed attack pattern 3, the neural network is trained to identify the expected attack pattern as 3.

C. The Neural Network training

To train the neural network the training data set is divided into two sets. The first set of data is the training data sets (260 samples) that were presented to the neural network during training.

TABLE I. TRAINING AND TEST DATA SETS

Number of Samples	Training Data	Test Data
Data Set 1	143	26
Data set 2	117	25
Total	260	51

The second set (51 Samples) is the data that were used to test the performance of the neural network after it had been trained. At the initial stage of the training, it was discovered that the neural network had too many categories to classify the input data into (i.e. 51 categories) because the neural network was not able to converge. To overcome the problem, the training data was further divided into two sets. The first set contained 143 samples and the second set contained 117 samples. These were then used for training two neural networks. Mat lab Neural Network tool box is used to perform the training. The training performance is measured by Mean Squared Error (MSE) and the training stops when the generalization stops improving or when the 1000th iteration is reached.

D. The Result

It took the system about one minute to complete the training for each the back-propagation neural network. For the first neural network, the training stopped when the MSE of 0.0016138 was reached at the 26th iteration. The training of the second neural network stopped when the MSE of 0.00012841 was reached at the 435th iteration.

TABLE II. COMPARISION OF ACTUAL AND EXPECTED OUTPUT FROM NEURAL NETWORK

s\N	Attack Pattern Investigated	Actual Output	Expected Output
Results from Neural Network 1			
1	Attack Pattern 1	1.0000	1
2	Attack Pattern 2	2.0000	2
3	Attack Pattern 3	2.9761	3
4	Attack Pattern 4	4.0000	4
5	Attack Pattern 5	4.9997	5
6	Attack Pattern 6	5.9998	6
7	Attack Pattern 7	7.0000	7
8	Attack Pattern 8	8.0000	8
9	Attack Pattern 9	9.0000	9
10	Attack Pattern 10	7.0000	10
11	Attack Pattern 11	11.0000	11
12	Attack Pattern 12	12.0000	12
13	Attack Pattern 13	12.9974	13
14	Attack Pattern 14	13.772	14
15	Attack Pattern 15	15.0000	15
16	Attack Pattern 16	16.0000	16
17	Attack Pattern 17	16.9999	17
18	Attack Pattern 20	19.9984	20
19	Attack Pattern 21	21.0000	21
20	Attack Pattern 22	22.0000	22
21	Attack Pattern 23	23.0000	23
22	Attack Pattern 24	23.9907	24
23	Attack Pattern 25	25.0000	25
24	Attack Pattern 26	26.0000	26
25	Attack Pattern 27	27.0000	27
26	Attack Pattern 28	28.0000	28
Results from Network 2			
27	Attack Pattern 29	28.999	29
28	Attack Pattern 30	29.9983	30
29	Attack Pattern 31	31.0000	31
30	Attack Pattern 32	31.998	32
31	Attack Pattern 33	32.8828	33
32	Attack Pattern 34	33.9984	34
33	Attack Pattern 35	32.8828	35
34	Attack Pattern 36	35.9945	36
35	Attack Pattern 37	36.6393	37
36	Attack Pattern 38	37.9999	38
37	Attack Pattern 39	37.9951	39
38	Attack Pattern 40	39.1652	40
39	Attack Pattern 41	40.9669	41
40	Attack Pattern 42	41.9998	42
41	Attack Pattern 43	42.998	43
42	Attack Pattern 44	43.9979	44
43	Attack Pattern 45	44.9991	45
44	Attack Pattern 46	45.8992	46
45	Attack Pattern 47	46.9956	47
46	Attack Pattern 48	47.9997	48
47	Attack Pattern 49	48.9999	49
48	Attack Pattern 50	49.8649	50
49	Attack Pattern 51	50.9629	51
50	Attack Pattern 52	50.6745	52
51	Attack Pattern 53	52.7173	53

To test the performance of the network, the second data sets were used to test the neural network. It was observed that the trained neural network gave an output as close as possible to the anticipated output. The actual and anticipated outputs are compared in the Table IV. The test samples in which the neural network gave a different output from the predicted output when testing the network includes tests for attack patterns 10, 35, 39, 40 and 52. While looking into the reason behind this, it was seen that the data observed for these attack patterns were not much. With more information on these attack patterns for training the neural network, it is predicted that the network will give an enhanced performance. During the study of the results from the neural networks, it was found that the first neural network had 96.51% correct results while the second neural network had 92% accuracy. The accuracy for both neural networks had an average of 94.1%. This subsequently shows that the neural nets may not correctly identify attack patterns presented to it all of the time. Nevertheless given the accuracy of the neural networks, it shows that neural networks can be used to evaluate software from its design.

IV. FUTURE WORK

To further improve the performance of the neural network system as a tool for evaluating software design, we are currently looking into the possibility of the system suggesting solutions that can help to prevent the identified attacks. Current research on solutions to software design security flaws gives a good insight in this area. Suggested solutions such as the use security patterns [11] and introduction of security capabilities into design in the SAT approach [19] are currently investigated.

The regularly expressed attack pattern used in training the neural network is a generic classification of attack patterns. Therefore; any unknown attack introduced to the neural network will be classified to the nearest regularly expressed attack pattern. Nevertheless the successfulness of the neural network in analysing software design for security flaws is largely dependent upon the input capturing the features of the software design presented to it. As this requires a human endeavour, further work is required in this area to ensure that correct input data is retrieved for analysis. In addition, the neural network needs to be thoroughly tested before it can gain acceptance as a tool for evaluating software design for security flaws.

V. CONCLUSION

Previous research works have shown that the cost of fixing security flaws in software applications when they are deployed is 4–8 times more than when they are discovered early in the SDLC and fixed [27]. For instance, it is cheaper and less disruptive to discover design-level vulnerabilities in the design, than during implementation or testing, forcing a pricey redesign of pieces of the application [3]. Therefore, integrating security into a software design will help tremendously in saving time and money during software development

Therefore, by using the proposed neural networks approach in this paper to analyse software design for security flaws the efforts of software designers in identifying areas of security weakness in their software design will be reinforced. Subsequently, this will enhance the development of secured software applications in the software industry especially as software designers often lack the required security expertise. Thus, neural networks given the right information for its training will also contribute in equipping software developers to develop software more securely especially in the area of software design.

REFERENCES

- [1] Agarwal, A. (2006), "How to integrate security into your SDLC", Available at: http://searchsoftwarequality.techtarget.com/tip/0,289483,sid92_gci1174897,00.html, (Accessed 24/10/2010)
- [2] Ahmad, I., Swati, S.U. and Mohsin, S. (2007) "Intrusion detection mechanism by resilient bpc Propagation (RPROP)", *European Journal of Scientific Research*, Vol. 17(4), pp523-530
- [3] Arkin B. (2006), "Build security into the SDLC and Keep the bad guys out", Available at: http://searchsoftwarequality.techtarget.com/qna/0,289202,sid92_gci1160406,00.html, (Accessed 24/10/2010)
- [4] Cannady, J. (1998), "Artificial neural networks for misuse detection", *Proceedings of 21st National Information Systems Security Conference*, pp368-381
- [5] Croxford, M. (2005), "The challenge of low defect, secure software- too difficult and too expensive", *Secure Software Engineering*, Available at: <http://journal.thedacs.com/issue/2/33> (Accessed 25/02/2012)
- [6] Gegick, M. and Williams, L. (2006), "On the design of more secure software-intensive systems by use of attack patterns", *Information and Software Technology*, Vol. 49, pp 381-397
- [7] Hinchey, M et al, (2008), "Software engineering and formal methods", *Communications of the ACM*, Vol.51(9), pp54-59
- [8] Ho, S. L.; Xie, M. and Goh, T. N. (2003), "A Study of the connectionist model for software reliability prediction", *Computer and Mathematics with Applications*, Vol. 46, pp1037 -1045
- [9] Hوجلung, G and McGraw G. (2004), "Exploiting software: The Achilles' heel of cyberDefense", *Citigal*, Available at: http://citigal.com/papers/download/cd-Exploiting_Software.pdf (Accessed 02/12/2011)
- [10] Howe (2005), "Crisis, What Crisis?" *IEEE Review*, Vol. 51(2), p39
- [11] Kienzle, D. M and Elder, M. C. (2002) "Final Technical Report: Security Patterns for Web Application Development", Available at <http://www.scrypt.net/~celer/securitypatterns/final%20report.pdf>, (Accessed 26/01/2012)
- [12] Kim, T., Song, Y. Chung, L and Huynh, D.T (2007) "Software architecture analysis: A dynamic slicing approach, *ACIS International Journal of Computer & Information Science*, Vol. 1 (2), p91-p103
- [13] Lindqvist, U, Cheung, S. and Valdez, R (2003) "Correlated attack Modelling (CAM)", *Air Force Research Laboratory, New York, AFRL-IF-RS-TR-2003-249*
- [14] Lyu, M. R, (2006), "Software reliability engineering: A roadmap", Available at: http://csse.usc.edu/classes/cs589_2007/Reliability.pdf (Accessed 21/09/2011)
- [15] Karunanthi, N., Whitley, D. and Malaiya Y. K, (1992), "Using neural networks in reliability prediction", *IEEE Software*, pp53-59
- [16] McAvinney, A. and Turner, B. (2005), "Building a neural network for misuse detection", *Proceedings of the Class of 2006 Senior Conference*, pp27-33
- [17] McGraw, G. (2006), "Software security: building security in", Addison-Wesley, Boston, MA

- [18] Meier, J. D., Mackman, A. And Wastell, B. (2005), "Threat modelling web applications", Available at: <http://msdn.microsoft.com/en-us/library/ms978516.aspx> (Accessed 24/10/2010)
- [19] Mouratidis, H. and Giorgini, P (2007), "Security attack testing (SAT)-testing the security of information systems at design time", Information Systems, Vol. 32, p1166- p1183
- [20] Pan, J. (1999), "Software reliability", Available at: http://www.ece.cmu.edu/~koopman/des_s99/sw_reliability/#introduction, (Accessed 21/09/2011)
- [21] Palshikar, G. K. (2004), "An Introduction to model checking", Embedd.com, Available at http://www.embedded.com/columns/technicalinsights/17603352?_requestid=12219,(Accessed 20/02/2012)
- [22] Pemmaraju, K., Lord, E. and McGraw, G.(2000), "Software risk management. The importance of building quality and reliability into the full development lifecycle", Available at: <http://www.cigital.com/whitepapers/dl/wp-qandr.pdf>, (Accessed 07/06/2011)
- [23] Ralston, P.A.S; Graham, J.H and Hieb, J. L. (2007), "Cyber security risk assessment for SCADA and DCS networks", ISA Transaction, Vol.46(4), pp583-594
- [24] Redwine, S. T. Jr and Davis, N.; et al, (2004), "Process to produce secure software: Towards more secure software", National Cyber Security Summit, Vol. 1
- [25] Srinivasa, K.D. and Sattipalli, A. R, (2009), "Hand written character recognition using back propagation network", Journal of Theoretical and Applied Information Technology, Vol. 5(3), pp 257-269
- [26] Tamura, Y.; Yamada, S. and Kimura, M. (2003), "A software Reliability assessment method based on neural networks for distributed development environment", Electronics & Communications in Japan, Part 3: Fundamental Electronic Science, Vol. 86(11), pp13-20.
- [27] Telang, R. and Wattal, S.(2004), "Impact of software vulnerability announcement on market value of software vendors- an empirical investigation", The Third Workshop, University of Minnesota, 13-14 May, Minnesota.
- [28] Threat Risk Modelling (2010) Available at: http://www.owasp.org/index.php/Threat_Risk_Modeling, (Accessed 24/10/201)
- [29] Mockel C and Abdallah, A.E (2011) 'Threat Modelling Approaches and Tools for Securing Architectural Designs of E-Banking Application', Journal of Information Assurance and Security', Vol. 6(5), pp 346-356
- [30] Spampinato, D. G. (2008), 'SeaMonster: Providing Tool Support for Security Modelling', NISK Conference, Available at: http://www.shieldsproject.eu/files/docs/seamonster_nisk2008.pdf (Last Accessed: November 2011)
- [31] Joseph, A., Bong, D.B.L. and Mat, D.A.A (2009) 'Application of Neural Network in User Authentication for Smart Home Systems' World Academy of Science, Engineering and Technology, Vol. 53, pp1293-1300.

A proposal of New Scheduling System on Smartphone with Interactive

Eigo Ito

Dep. of Information System, Toyo University
2100 Kujirai, Kawagoe, Saitama, 350-8585,
JAPAN
e-mail@e-ito.jp

Takayuki Fujimoto

Dep. of Information System, Toyo University
2100 Kujirai, Kawagoe, Saitama, 350-8585,
JAPAN
me@fujimotokyo.com

Abstract

Today, many information systems are integrated in mobile device. One of that is a scheduling system. In the past, we make schedules with a notebook. Dates are taken the form of lists or calendars, and schedules are filled in the attendant white spaces. This system is good for checking the day has schedules and finding the day's space. A new device – smartphone – is using the same system. However, they need large display size and high-resolution: like notebooks. In addition, this system needs less-restriction input interface. Even now, scheduling system on notebooks has more benefits than that on smartphones.

This paper is written about scheduling systems that suit smartphones. One of the ways is using roll type view. This view uses a shape of 3D cylinder. This model can make us get time line with the perspective. The visual effects cause intuitive operations: to spin, to parallel shift, to change the scales. We propose the Roll Type Scheduling System.

Keywords: Video Image; Video Sharing, Social Network, Communication, Post TV

1. Introduction

Today, we live by time. Clocks make accurate running public transportation service. In addition, clocks make accurate scheduling us. We can have many appointments. Then it is the most important things for daily life that we keep to schedules. Scheduling systems support this bustling life.

Pervading systems for scheduling are 2 types (Fig. 1). One is “List Type Scheduling”. List Type scheduling is a list of appointments. Schedules are lined up widthwise or endwise. Other system is “Calendar Type Scheduling”. This system uses table of days. The rows are weeks. And columns are days of the week. Both of the two type scheduling systems use blank space with each day. If we get appointments, we write on the blank space a summary of the appointments. We can check details of a schedule by this system. And we can check that schedule is fully booked up. Notebooks are the most popular devices that use those systems because notebooks are the most popular mobile devices. But today, many systems are integrated in smartphone. Recently, scheduling systems is porting to smartphone.

Smartphone is high functionality device. However, display size and high-resolution is smaller than notebooks. Existing scheduling system aim showing total image. Smartphone's display is not match existing scheduling system. In addition, smartphone's input system is different to notebook's one. On notebook, we use pen to input and we can write anything: letters, pictures and decorations. On the other hand, smartphone's input system is touch-keyboard. Tough-keyboard allow us to input letters' data. It is not visceral. So, smartphone is not popular device to use scheduling device, yet.

In this paper, we propose a new scheduling system for smartphone, named Roll Type Scheduling System. This system aims to show schedule with import. Schedule import is variable because appointments are close to time.

We focus to that import is depended by time. An appointment, which has less time difference from the time, is important. Roll Type Scheduling System shows only important term that is shorter than existing system. In addition, this system's view uses a shape of 3D cylinder. Appointments with time look like on the surface. Center that is approximate to user is normally showing. The most important schedule is fixed on the center. Some appointments that have some time difference by the time are putting on over or under the center. The position is followed time difference. This view is look like on the cylinder; the less import appointments are shadowy background color and font color. In addition, the less import appointment's letters is shrunk lengthwise.



Fig.1 Calendar Type and List Type Scheduling System

2. Roll Type Scheduling System

2.1. View

Roll Type Scheduling System aim showing schedule with import. When we want to check schedule, we want to get closing data form schedule system. List Type Scheduling System on notebooks and Calendar Type Scheduling System on notebooks are match this goal. But those systems on smartphone are not match the goal because of smartphone's display. It is too small and too less-resolution to show total

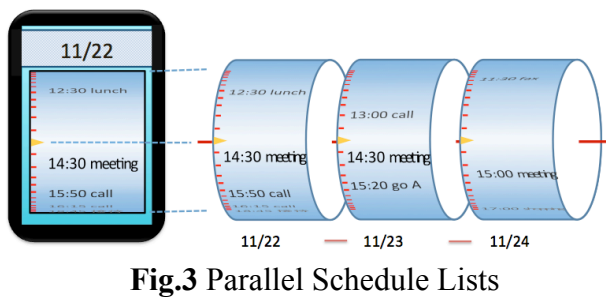
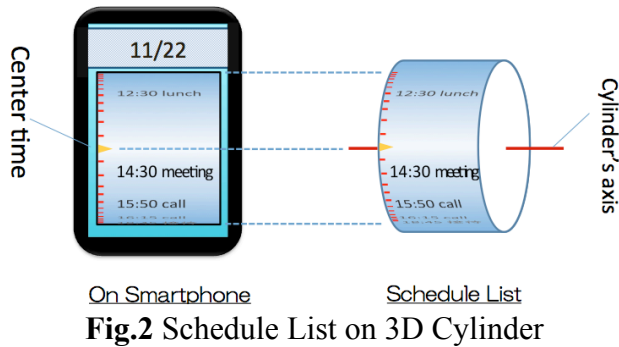
image. We must scroll to use those systems on smartphone.

Roll Type Scheduling System have a new view system. Using horizontal 3D cylinder makes us visceral cognize (Fig. 2). The most important schedule – when we want to check plans on the time, an appointment on the time is the most important – is fixed in the center of cylinder. Other appointments that have time lag from the center are putting on the over or under the center. This position is determined by time difference. This view look like List Type Scheduling System's one, but this system's schedule list is on the 3D cylinder's surface. Appointments view is shadowy and shrunk lengthwise by time differences. This system makes us visceral cognize.

This visceral view is not only to show schedule by times. Above is written about hours on a day. If it shows schedules on a month, the system checks appointments each days. But, this system can use 3D cylinder view to show days on a month. In a similar way, Roll Type Scheduling System can show months schedules on a year by 3D cylinder view. This changing is called “scale shift” in this paper.

3D cylinder allows us visceral operations. This shape connect roll. This cylinder is horizontal, so we think that this will roll vertical. This system is supported to the visceral image. Scrolling endwise cause schedule list view changing. When display is scrolled up to down, the center time get back. On the other hand, when display is scrolled down to up, the center time gaining time.

3D cylinder connects parallel cylinders (Fig. 3). Parallel cylinder has the same center hour and different date, because our life is cyclic by the days. When display is scrolled right to left, the cylinder is shift a day before. And when display is scrolled left to right, the cylinder is shift a day after. This operation is high visceral as vertical scrolling.



2.2. Schedule Data

In order to use this system, the function to input the schedule data to manage is required.

The usual schedule management system performs a simple character representation. There was no obstacle in giving visual operation guidance. However, in this system, vision expression of the time information on a schedule is carried out. Presenting of information other than a schedule narrows a display domain. Moreover, the display gives unnecessary information. Therefore, a schedule input is performed on entire another screens, and the immediately time after starting is taken as the center time.

Double-touch-upping on the 3D cylinder performs the shift to the input screen. A clock time, a title, and a detail exist as schedule data. User is inputting all these information and registers schedule data. These pieces of input can perform edit and deletion from a schedule detailed screen.

3. Conclusion

The existing scheduling system is a list type and a calendar type suitable for a handwritten input and display. However, it is not suitable for smartphones because of the resolution and screen size. These systems on smartphone cannot avoid a problem that is the recognition nature of a character.

In this paper, we seem to that the importance of a schedule was related from the time lag centering on the center time and it. A schedule with a smaller time lag from the center time has higher importance. And it displays so that it may be easier to sight a more important schedule. The schedule with low importance changed the longitudinal width of a character, the character color, and the background color so that it might be hard to sight on the contrary.

The method of this display serves as a solid display, which arranges the character on the 3D cylinder surface. Since it is as close to a user as a center, the thing and the user also with high importance recognize naturally. Moreover, the form of a 3D cylinder made scroll operation more nearly intuitive, and made recognition of the time lag easy.

4. Future Tasks

In this system, recognition of the time lag was made easy by indicating a schedule list the form that can be caught more sensuously.

However, it eventually makes the pliability in a display fall. In the present trial production system, in order to avoid crossing the new-line and display domain in a schedule list, the title is set up aside from detailed contents. By this method, the amount of information acquired on one screen will be reduced, and the opportunity of a detailed check will be increased.

Now, changes of a page will increase and the simplicity of use important as a schedule management system will be lost. On the other hand, it is difficult to make character size small for the number reservation of characters per one line, in order to change the longitudinal width of character size in this system. Therefore, it is necessary to verify the method of a time display

or the display for which it is more suitable about character size by subject experiment. Moreover, when the time of a schedule is very near, there is a problem that the display will overlap.

Visibility has also here sufficient thing, a thing which can be shifted to the upper and lower sides or right and left so that it may not overlap, and room to inquire in the present state.

By this system, since only the short schedule of a period can be checked simultaneously, a possibility of becoming unsuitable for grasping the existence of a schedule is also considered.

However, using the existing calendar type and a List style display type can solve this. By the schedule system proposed by this research, by making it the system that can display other kinds, and the system that can share schedule data, if it becomes a system suitable for more nearly actual use, it will think.

We advance examination of the mechanism as a system with higher practicality also including an interface design from now on.

5. References

[1] A.B. Smith, C.D. Jones, and E.F. Roberts JMA Management Center Inc., "Survey about how to use a Notebook for Scheduling 2011", http://www.jmam.co.jp/new/newsrelease/1260431_1362.html.

[2] Takashi Yoshino, and Takayuki Yamano, "Casual Scheduling Management and Shared System Using Avatar", *IPSJ SIG Technical Reports*, 52(3), March 2011.