Modeling and Analysis of NASA's Mission Software Development Archetypes using Petri-Net Graphs

Amanda Pavlicek and Tai-Chi Lee
Department of Computer Science & Information Systems
Saginaw Valley State University
7400 Bay Rd., University Center, MI 48710

Abstract:

The purpose of this research is to improve the validation and performance of mission safety software within the Mission Control room, as well as achieving financial objectives and fulfilling governmental regulation while utilizing the best software engineering and project management practices. The implementation of this project will be represented in an analysis paper that utilizes Petri nets to portray the enhanced relationships within the Mission Control software infrastructure and the underlying issues of the previous software infrastructure within Mission Control. The results of this research will contribute to our understanding of designing a more efficient software infrastructure. With this newly acquired knowledge, NASA organization, once the organization is given the opportunity to restart manned space missions, can push the barriers of space exploration and aeronautical science within the current market's reinforced limitations in future NASA space missions.

1. Introduction

The NASA space program has undergone a series of innovations for more than half of century since Russia launched Sputnik, the first satellite to orbit the Earth in 1957. Through the years, NASA has made great accomplishments in space exploration using advanced Mission Control software. While some mission software systems are successful in their objectives, there are other systems that need to be improved upon so the risk of failure during a NASA mission can decrease. In addition, the world economy's unpredictable state should be considered in determining how and where the NASA software team obtains or creates Mission Control and vehicle software.

Mission software is based on bettering all phases of a NASA space mission, in which examples of such are ground and flight data systems and on-orbit performance management. This category of software, which is composed of workflow and data-processing applications, is accountable for the scientific progress of a mission and the safety of NASA astronauts, space shuttle, and other aeronautical equipment. Research will revolve around not only NASA's current state as a governmental agency, but also the installation and performance of NASA mission software systems and applications within the Mission Control Room prototype, individually as independent applications, and collectively as a functional, vital component of a NASA mission.

2. Modifications in Software Methodologies

Norman Kluksdahl, a Systems Engineer at Johnson Space Center, noted that the most monumental change with NASA's Mission
Control software infrastructure and enhancement is their software development methodology, the work-related outline executed to plan and implement software applications, in creating software for both space vehicles and the Mission Control consoles [6]. All areas of the software development process, including programming, management, and application users, will be affected by this prevalent modification as it allows the development team to produce validated, well-received software before the deadline and with minimal cost.

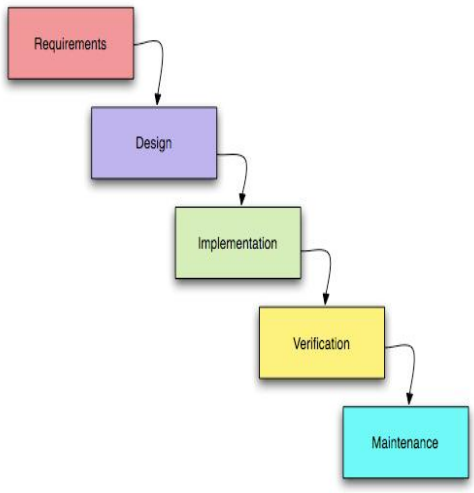

Figure 1. A pictorial representation of the Waterfall Development Methodology that the Mission Control Systems team relied on for their software-engineering framework in employing Mission Control software (The Smart Method). [12]

In the past, the Mission Control Systems team followed a very straightforward process that is known as the Waterfall development, a sequential software-engineering process [10]. According to my research in project management, the programming team converses with management and other stakeholders to obtain details about the application being

developed. After the application's first version is fully advanced, an abundance of Quality Assurance and Quality Control procedures occur in the application development pre-mission phase known as the Baseline Release Process [6]. Succession with module tests allows the Simulation team to proceed with integration testing, which provides a set of tools to test subsystems [6]. The "Test Rig" or "Simulation" practices occurs when changes suggested by programming team are made, which gives the team an opportunity to watch the system and observe how it interacts with environment [6]. To be truly efficient, it is necessary to conduct lifecycle analysis upon all pieces of software to ensure that each piece doesn't have tight interdependency within one another. The team loads the new software into the infrastructure after the selected applications pass the validation testing, but the implementation process varies depends on how long simulation and runtime should take [6]

The greatest flaw with this waterfall process, according to Kluksdahl, is that the progress of the project is compromised if a stakeholder wishes for a change to be made in the new application [6] [10]. Therefore, all of the completed work, whether it was a sliver of a program functionality to an entirely new version, frequently has to be scrapped and the developers have to start all over again. There is no limit to how many times this could occur, and this major disadvantage has misused precious time, money, and solid relations within the NASA organization. Unfortunately, a stakeholder's influence on the project's final result will always remain a constant in any substantial endeavor because not everyone will know what they exactly desire, especially if it revolves around a concept that was never physically created before [10]. NASA, along with other corporations, have witnessed what the waterfall development cycle can do for an software development-driven project, where 70 percent of objectives in an average project are not met [11]. Furthermore, the waterfall development approach has demonstrated that it is not the most fiscally-conservative methodology as project costs are, on average, much higher than the estimated costs approximated during project initiation [11].

In addition, space operations and management officials rarely observe concrete, effective software development in their project preparation stage or incorporate room for flexibility within the finished product for future software-engineering trends [6]. Incorporating future trends within obtaining the project's end objectives allows advancing technology and strategies to take root. However, today's software development tends to revolve around the latest programming fad, such as object-oriented design, as a cure-all for efficient development. Often, the effect is to reduce productivity because engineers are required to learn a new methodology. As exemplified in several information technology-related projects, industry experience shows that the first use of a new methodology is often very inefficient, and thus more costly, than the methodology it replaced. This causes issues with maintainability and adaptability.

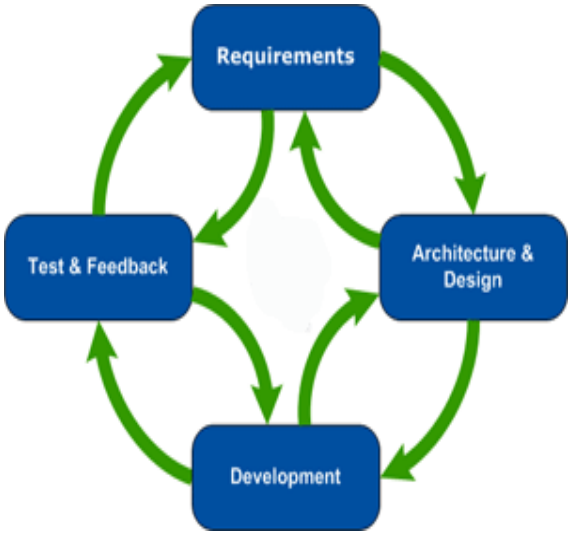

Figure 2. A pictorial representation of the Agile Software Development Cycle that Dr. Kluksdahl and other software system engineers are incorporating into their work methodology to ensure stress-free software development. [1]

What the Mission Control System group needs to build a better project framework for NASA developers and management to follow is one that embraces compliance in the sense of team communication, project objectives, and resources. During this time of mission inactivity, Kluksdahl expressed that this lack of productivity gives the team an opportunity, enough time and resources to "decide on a methodology and adapt to its standard practices"[6]. The Mission Control System group selected the Agile Software Methodology to integrate into their project objectives. Created by seventeen prominent software developers within a document called the "Agile Manifesto" in 2001, the Agile Software Methodology embraces adaption, cooperation, swift delivery, and client goal-orientation as the crucial elements of successful software development [3]. As stated in the Manifesto, programmers that function under an Agile programming settings value "individuals and interactions over processes and tools, working software over comprehensive documentation, customer collaboration over contract negotiation [and] responding to change over following a plan", which allows teams to actually provide a product on time and within the flexible requirements set by the client [3].

The Agile Methodology thrives on four basic stages of software development: Project Requirements, Software Design, Development or Implementation, and Testing and Debugging, the four stages that are present in the waterfall model [2]. One of the most notable discrepancies between the agile and waterfall development approaches is that when project requirements change, resources are modified, and other factors previously analyzed confront a major alteration, the entire project doesn't start over. Instead, the project

leaders and key stakeholders integrate the changes into the project plan accordingly as the developers interact with each phase of the cycle until the current issue is resolved [2]. How various developers and other stakeholders are able accomplish distinctive tasks from different stages of the software development process is through "iterations", which allows software developers to complete a full software cycle for numerous tasks within a short amount of time [2]. A fully operative software development team is engaged in the planning, development, testing, and execution included in every iteration, where a deliverable or a segment of the finalized deliverable is produced [2]. Each project member must be heavily involved in selected Agile practices, such as face-to-face conversations, intricate team meetings involving daily reports, and comprehensive documentation [2]. In addition, users and developers alike must review and approve changes before these project modifications are merged into the baseline, developers must review each other's code and programming skill sets, and each piece of code must undergo unit tests. Customer representatives are introduced into the project team to act upon the stakeholder's behalf, allowing the project team sufficient access to the client's needs. The iteration approach contrasts with the waterfall development approach, which focuses on employing distinct phases with checkpoints and deliverables at every stage, a nonflexible, risky method of project planning [2] [11]. Kluksdahl described that through these phases allow developers and project leaders to acquire the knowledge necessary to fully implement an unfamiliar system, in which the waterfall methodology greatly lacks, by completing easier tasks in the earlier phases and focusing on the challenging assignments later [6] [11]. Also, verification of project requirements occurs much earlier in the development process than it would with the waterfall development approach, which permits stakeholders to adjust requirements while they are still relatively painless to modify [11].

In his discussion concerning the transferring of software methodologies, Kluksdahl stressed that the Mission Control Systems team is only restructuring the way the team operates within a software development environment and recycling the indispensable components (software development phases, requirements, communication means, etc) to fit the Agile structure [6]. Currently, Kluksdahl and other NASA system engineers are fundamentally pushing a development system into operation that does not employ all of the most modern software practices or is without any of the elements of the preceding software methodology. According to Kluksdahl, "What we are doing is we are making software development better, more sufficient for everyone involved in future continuing, and complex projects. Everything still works, so why should we get rid of it" [6]. The factors that needed to be improved upon, such as flexibility, team collaboration, and project integrity were the only portions that needed to be emphasized to build an advanced, enduring development methodology [6] [11].

3. Mathematical Approach of Petri Net Models

A Petri net is exemplified by these mathematical structures provided by James Peterson, in his book: *Petri Net Theory and the Modeling of Systems* [9]:

DEFINITION 3.1: A Petri net structure *C*, is a four-tuple, C = {P, T, I, O}, where P = {P1, P2…Pn) is a finite set of places, $n \geq 0$. T = {t1, t2… tm) is a finite set of transitions, $m \geq 0$. The set of places and the set of transitions are disjoint, $P \cap T = \emptyset$. I: $T \rightarrow P^{\infty}$ is the input function, a mapping from transitions to bags of places. O: $T \rightarrow P^{\infty}$ is the output function, a mapping from transitions to bags of places.

The cardinality of the set P is n, and the cardinality of the set T is m. We denote an arbitrary element of P by pi, i = 1… n, and an arbitrary element of T by tj, j = 1… m.
I(ti) is a bag of input places for the transition ti.

O(ti) is a bag of output places for the transition ti [9].

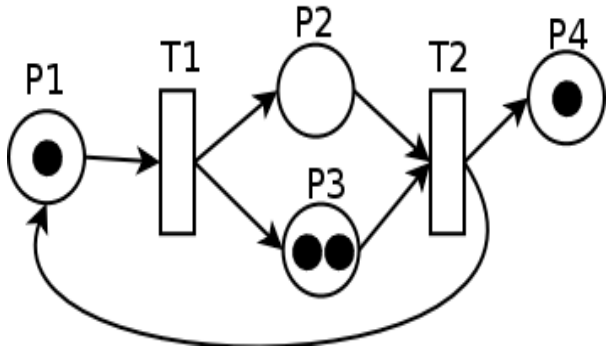

Figure 3. An example of a Petri net graph with four places, two enabled transitions, and four markings, such that I(p1) = {t2}, O(p1)={t1}, I(p2)={t1}, O(p2)={t2}, I(p3)={t1}, O(p3)={t2}, I(p4) = {t2}, and O(p4)={}.[4]

This research endeavor will specifically concentrate on a branch of the Petri net theory that is called Applied Petri net Theory, where a Petri net is represented by special graphs called Petri net graphs [9]. Petri net graphs are also described in mathematical terms below, establishing Petri net graphs as bipartite directed multigraphs, where the multigraph allows various arcs or functions from one node to another [9]:

DEFINTION 3.2: A Petri net graph *G* is a bipartite directed multigraph, G=(V, A), where V ={v1, v2, …, v5} is a set of vertices and A = {a1, a2, …, a,} is a bag of directed arcs, ai = (vj, vk), with Vj, Vk is a subset of V. The set V can be partitioned into two disjoing sets P and T such that V = P U T, $P \cap T = \emptyset$, and for each directed arc, a is a subset of A, if ai = (vj, vk), then either Vj is a subset of P and vk is a subset of T or Vj is a subset of T and vk is a subset of P [9].

A Petri Net is composed of five elements: a set of *places* (represented by the letter 'P' in formulas and by circles in graphical representations), a set of *transitions* (represented by the letter 'T' and by bars in graphical representations), an *input* function (represented by the letter 'I' and by incoming arrows or arcs in graphical representations), an *output* function (represented by the letter 'O' and by outgoing arrows or arcs in graphical representations), and markings located inside place (are represented by the marking *u* and by dots in graphical representation) [8] [13]. The input and output functions creates relations between transitions, which controls the arrival and departure of resources, and places, which is a component that holds resources [8] [13]. A mapping from a specific transition (tj) to a specific place (pi), within a collection of input places I(tj) is an input function *I* [9] [13]. If transition (tj) is mapped to a specific place (po), within a collection of output places, O(tj), then the result is an output function [8] [9].

Markings are an essential component of a Petri net graph because they represent resources used to simulate the overall functionality of the replicated system. Described below is the mathematical description of a marking:

DEFINITION 3.3: A marking μ of a Petri net *C = {P,T,I,O}* is a function from the set of places *P* to the nonnegative integers *N*. $\mu: P \rightarrow N$.
The marking μ can also be defined as an n-vector, $\mu = (\mu 1, \mu 2, \ldots \mu n)$, where n = |P| and each μi is a subset *N*, i =1, . . ., n. The vector μ gives for each place pj in a Petri net the number of markings in that place. The number of markings in place pi is μi, i = 1…n. The definitions of a marking as a function and as a vector are obviously reflected by $\mu(pi) = \mu i$. The functional notation is somewhat more general and so is more commonly used [9] [13].

In other words, markings are used to define the execution of a Petri net, where the number and positions of markings may alter during the execution and controls the execution's duration [9] [13]. In order for execution to occur within a Petri net graph, a transition must first be enabled for firing, which requires each of the transition's input places to have at least as many markings in it as arcs from the places to the transition [8] [9]. Next, the transition is fired by removing all of its enabling markings from its input places and then depositing into each of its input places and then depositing into each of its output places one marking for each outgoing arc from the transition to the place. Thus, multiple markings are produced for multiple output arcs and transition firings can continue as long as there is at least one transition enabled. Otherwise, the execution is terminated and the system is presumed to be idle [8] [9].

The most straightforward view of a Petri net representation of a particular system focuses on two concepts: events and conditions. Actions that occur within the system are events, in which the instigation of an event is influenced by the current state of the system. The varying state of the system is derived by a set of predetermined conditions, "predicate[s] or logical description[s]" to the system's status [9] [13]. In order for an event to occur, the event's preconditions must be completed. The repercussions of an event, or post conditions, may cause the preconditions of other events to end or commence [9] [13]. Conditions are modeled by places in a Petri net while events are represented by transitions [9] [13]. The inputs of a transition are considered preconditions of the consequent event, where an enabled transition indicates that the preconditions are completed [9] [13]. Thus, the outputs of a transition are the post conditions of an event [9] [13].

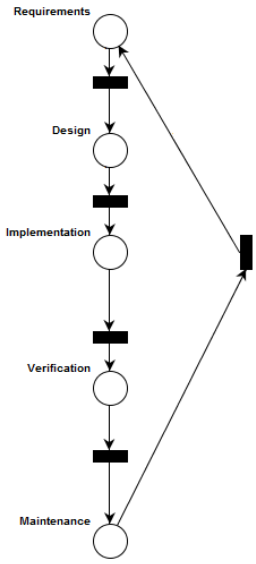

Figure 4. A Petri-net graph that demonstrates how the NASA Mission Control Systems team applied the basic outline of Waterfall Methodology in their software development process.

4. Petri Net Graphical Comparisons of Software Methodologies

Four Petri net graphs were created using an open-source application that creates Petri net graphs called Pipe 2. The resulting Petri net graphs fully resemble how the NASA Mission Control Systems team conducted and will conduct their software development utilizing the precedent approach: Waterfall Development Methodology and the impending approach: Agile Software Development. Figure 5 displays the progression of an archetypal project where the project team must endure the requirements phase, design phase, implementation phase, verification phase, and maintenance phase (which is an ongoing process until the software is eliminated from the system). So, when the project starts, the resources of the first version (which is symbolized by the marking) will undergo the requirements phase and when it is completed, a transition will be enabled, fired, and the marking will proceed to the design phase. This will continue until the software is successfully released.

Nevertheless, when project requirements change or a major project issue arises within the software development or the deliverable is being produced, another marking will be added to the "Project Hindrance" place, as exemplified in Figure 4. This particular graph is built in a manner in which if there are two markers next to each other in separate places (which represents the encounter or realization of a significant issue), a specific transition will be enabled (representing the event in which the team realizes that there is an issue at hand), firing

one of the markings to the beginning of the project process and firing the other to one of the "Resources Cache" place. The project team must repeat the project process again with new resources, a revised project-related blue print, while the dilapidated resources and wasted time will be accumulated "Resource Cache" places until the deliverable is released into the system. Evidently, this Petri net graph (Figure 5) illustrates how the Waterfall development methodology exhibits an uncomplicated organization, but this limits the amount of resources and adaptability of the project team. The following Petri net graph (Figure 5) epitomizes the Mission Control System team's current development process, which employs Agile Software Development practices and six delivery iterations. The marking (current state of the specific project) begins at Iteration 1 at the beginning of the project and the project team undergoes its own software development cycle (represented in Figure 7) to manufacture a deliverable. When that iteration's particular deliverable is generated, the precondition of the next event is fulfilled and the nearest transition in Figure 6 (T0) will become enabled, transporting the ticket from the preceding iteration to the subsequent iteration, which is called Iteration 2. This process continues until the marking reaches the maintenance place, where the marking will always be repositioned by a transition that will permanently be enabled (T7).

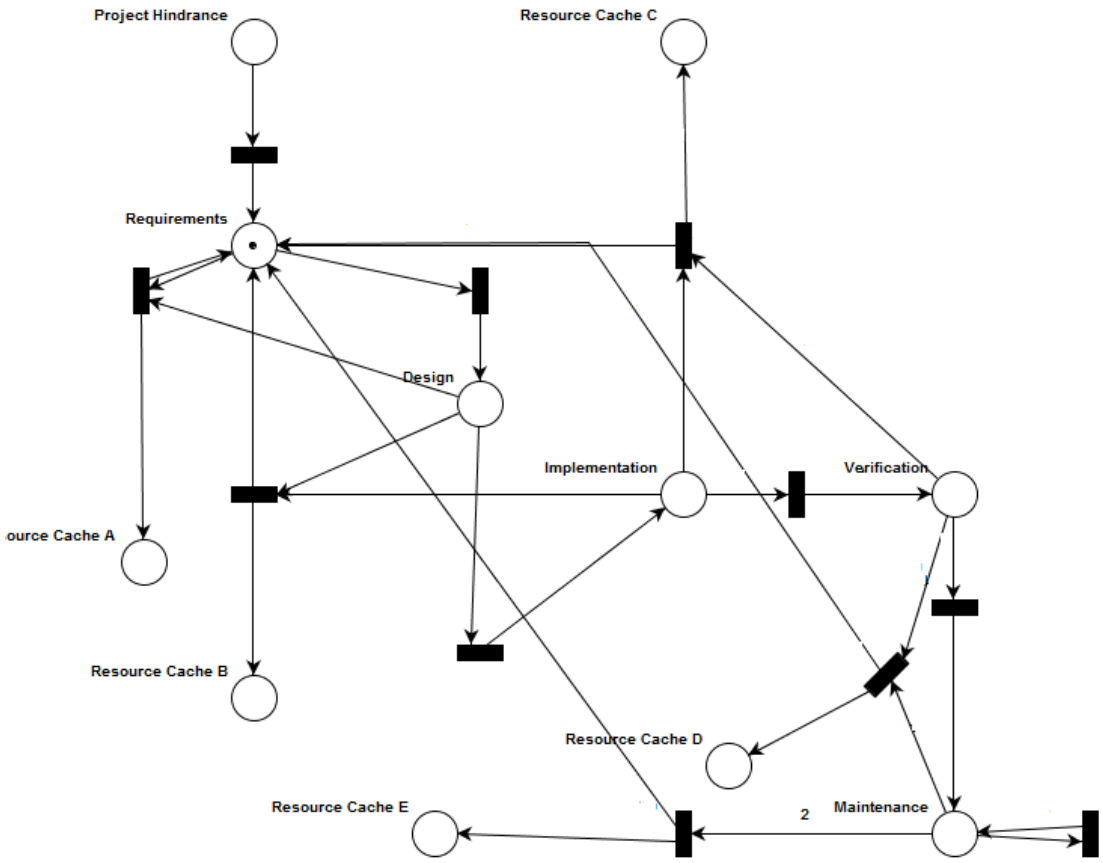

Figure 5. The constructed Petri Net that represents how the Waterfall Methodology was used within the NASA software development environment and the multiple problems that the methodology possessed. The Project Hindrance place represents any kind of condition or problem which may delay the project, such as management disagreement, misallocated resources, and drastic change in project requirements. When a resource (project resource) is found in a Resource Cache place, it is considered a misplaced resource (a condition to an imperfect project) that increases the final cost of the project. In addition, if there is more than one marking within the Maintenance place, then the number of markings represents the number of versions in which the delivery has been through.

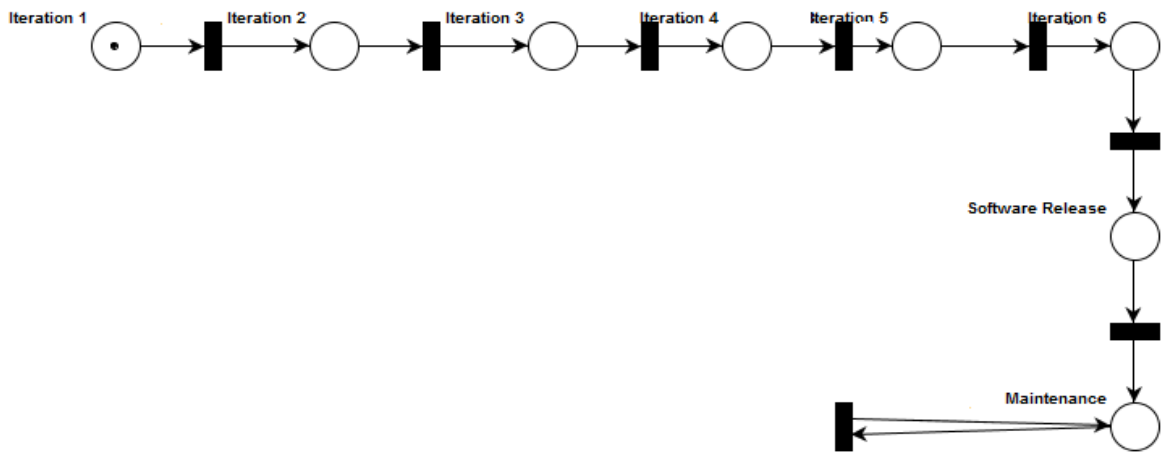

Figure 6. A Petri net graph that represents how Kluksdahl's Mission Control System team is going to integrate the basic concept of Agile Development Methodology. Each place represents an iteration or project phase and each marking represents allocated project resources. If an iteration possesses a marking after a transition is fired, then it resembles the current state of the specific project. If a problem occurs in an iteration, it is taken care of within that iteration right away.

The Mission Control System team must consider if any problematic difficulties will arise during the development of an iteration's deliverable. Within the world of Petri nets and graphical mathematical models, there is no way to logically relocate the marking to the first place of the Petri net due to

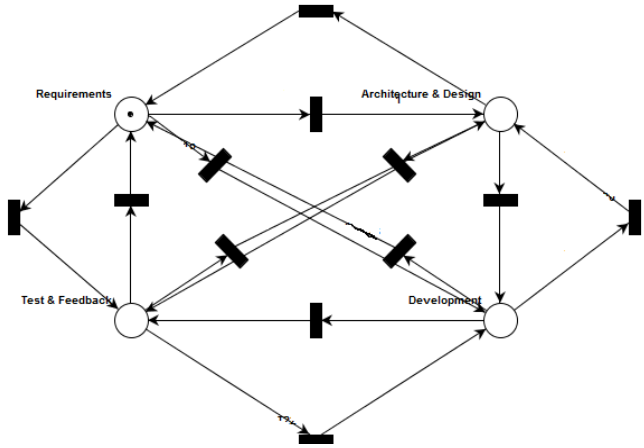

Figure 7. This Petri net graph represents the Agile software development cycle that is assigned to each project iteration. This closely resembles the flowchart example found in Figure 7 where the present location of the marking represents the current phase in which the project team is undergoing. However, the team is able to move to any other phase if need be.

the fact that the advancement of the project is almost linear, from beginning to end. If an issue occurs that would characteristically cause the project team to retrace back to Iteration 1, or the establishment of the project, the issue will be resolved within that current iteration before the next transition enables. Within the iteration, the project team can move to any phase of the software development cycle to disentangle the issue (whether it be using more face-to-face communication with clients, or revising the requirements of the iterations, or conduct more conscientious coding during the implementation phase) at hand. After the issue is resolved, the project team can go forth in completing the originally-planned objectives within the iteration and the team can make an endeavor to prevent a similar issue from happening again.

5. Conclusion

The Petri net investigations of both software development methodologies revealed that the Mission Control System team is making significant, beneficial progress in renovating the Mission Control Room by establishing Agile Software Development characteristics into their own infrastructure. Through the project's deliveries, six delivery phases in span of three planned years, the team expects to repeat each delivery process (update, design, work from high-level tasks to specific tasks) in order to ensure the best results. The deliveries and project's development is undergoing a "loading bar" process, or an incrementing progression towards a project's completion [6]. This allows system analysts and other researchers to measure a project's development and their ability to learn how to operate the technology using key phases. It permits the project team to approach the major issue concerning the Mission Control software infrastructure through a divide and conquer approach: learn the easy tasks first and put the most difficult and/or incompletely-specified tasks on the side to be completed later in the project [6]. Although the software production, resource verification hardware acquisition, and other tasks are still under construction, I believe that the verification of how the Agile Methodology attributes will be integrated into the infrastructure propelled the Mission Control Systems team a great length towards their ambition: the project's final delivery of a resourceful, powerful Mission Control Room.

As expressed before by Dr. Kluksdahl, along with several members of the Mission Control Systems team, experience is their main advantage in this project, a statement that I agree with whole-heartedly [6]. Gained knowledge gives the Mission Control Systems team, along with other stakeholders, the best results under an optimal testing environment that is facilitated by the combination of multiple project and development methodologies [6]. I found this component intriguing since knowledge gained from the previously-completed phases helps provide operators and testers sufficient knowledge to evaluate past error recognition and trends. When a situation is fully corrected based on the team's strict standards, the confidence of the team and crew in achieving the flight mission increases significantly, which allows the team to work harder towards a successful future. The more each team member grows mentally and personally, the more their own motivation will develop and they would finally realize the importance of their work to the entire organization, and, perhaps, the entire human race. Who would compromise the promise of a better future and the opportunity of obtaining knowledge in order to maintain the archaic environment that has failed NASA before. If the team, along with management and all other departments, tenaciously continues this endeavor in the determined "loading bar" pace and able to create resource-saving solution, NASA will be triumphant once again.

6. Acknowledgements

I would like to express my gratitude to Dr. Norman Kluksdahl, whose contributions and constant encouragement inspired me to push my own boundaries in this project and understand the significance of not only NASA's role in society, but also my contributions to the academic world. I would also like to thank the rest of the Mission Control Systems team, and my research mentor, Dr. Tai-Chi Lee, who provided extremely valuable input concerning the conducted research and my own conclusions.

References

[1] Agile Methodology, http://karthiksangi.files.wordpress.com/2009/07/agile3.gif

[2] "Agile Programming", http://agileprogramming.org/, 2012

[3] K. Beck *et al., Agile Manifesto*. http://agilemanifesto.org/, 2001

[4] Detailed Petri-Net. http://en.wikipedia.org/wiki/File:Detailed_petri_net.png

[5] T. England, "NASA's New Future." University of Michigan, Michigan Space Grant Consortium Conference, Ann Arbor, MI, November 12, 2011.

[6] N. Kluksdahl, Systems Engineer at Johnson Space Center.

[7] N. Heath, (2010). "Space Exploration: The Computers That Power Man's Conquest of the Stars." *Silicon.com.* http://www.silicon.com/management/public-sector/2010/09/25/space-exploration-the-computers-that-power-mans-conquest-of-the-stars-39746245/print/

[8] T.C. Lee, "Some Properties of Petri Nets and Their Graph Models," Proc. 27th ACM Annu. Conference of Southeast Region, April 1989, pp. 657-659. Interest

[9] J.L. Peterson, "*Petri Net Theory and the Modeling of Systems,"* Englewood Cliffs, N.J.: Prentice-Hall, 1981.

[10] W.W. Royce, "Managing the Development of Large Software Systems*,"* Proc. IEEE Wescon, 1970.

[11] Serena (2007). *"Agile in the Enterprise"* http://www.serena.com/docs/repository/products/teamtrack/agile-in-the-enterprise.pdf

[12] Waterfall Model, http://www.learnaccessvba.com

[13] M. Zhou and K. Venkatesh, "Modeling, Simulation, and Control of Flexible Manufacturing Systems—A Petri Net Approach*,"* World Scientific, vol 6, 1999.