Resource Dependencies in Parallel Development of Military Systems: A Comparison of Waterfall and Agile Development Methodologies

By

Erik Roberto Garcia

S.B. Aerospace Engineering with Information Technology (2006)
Massachusetts Institute of Technology

SUBMITTED TO THE SYSTEM DESIGN AND MANAGEMENT PROGRAM
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

MASTERS OF SCIENCE IN ENGINEERING AND MANAGEMENT

AT THE

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2014
[June 2014]

Signature redacted

Signature of Author: _____
Erik R. Garcia
System Design and Management Program
January 31, 2014

Signature redacted

Certified by: _____
Nelson P. Repenning
Professor of System Dynamics and Organization Studies at MIT Sloan School of Management
Faculty Director of MIT Executive MBA Program

Signature redacted

Accepted by: _____
Patrick Hale
Director
System Design and Management Program

Table of Contents

1 Introduction 1
1.1 Motivation 2
1.2 Research Question 4
1.3 Thesis Organization 4
2 Background 5
2.1 Department of Defense Development Life Cycle 5
2.1.1 Materiel Solution Analysis 6
2.1.2 Technology Development 7
2.1.3 Engineering & Manufacturing Development 7
2.1.4 Role of System Engineers 7
2.2 Waterfall and Agile Development 7
2.2.1 Waterfall Development 8
2.2.2 Agile Development 9
2.3 Previous Relevant Models 10
2.3.1 Dynamics of Concurrent Software Development 10
2.3.2 Simulating Kanban and Scrum vs Waterfall with System Dynamics 11
2.3.3 Agile Project Dynamics 12
3 Model Development 13
3.1 Development and Testing Timelines 14
3.1.1 Materiel Solution Analysis 17
3.1.2 Technology Development 17
3.1.3 Engineering & Manufacturing Development – System Design 18
3.1.4 Engineering & Manufacturing Development – System Testing 25
3.1.5 Human Resourcing 31
3.1.6 Table Constant 32
4 Analysis of Waterfall and Agile development Models 39
4.1 Overview of Waterfall and Agile Model results 39
4.1.1 Requirements Completion 39
4.1.2 Types of Technologies Developed 40
4.2 Analysis of Waterfall and Agile model development tendencies 41
4.2.1 Requirements Development in each Build 41
4.2.2 Types of Technologies Developed 48
4.3 Discussion 51
4.3.1 Reducing conflict between cost, schedule, and capability constraints 51
4.3.2 Reducing Firefighting 51
4.3.3 Impact on Development Tasks 53
5 Conclusion 55
6 Future Work 57
7 Appendix 58
7.1 System Design during Agile Development 58
7.2 System Design TRL Coflow during Agile Development 59
7.3 System Testing for both models 60
7.4 System Design Team 61
7.5 Software Design Team 62
8 Works Cited 63

Table of Figures

Figure 1 - System Acquisition Framework (Defense Acquisition Guidebook, 2013) 5
Figure 2 - Implementation flow during development of a concept/software system (Royce, 1970) 9
Figure 3 - Implementation flow during development of a increments of a concept based on Figure 2 10
Figure 4 - Agile development framework of Scrum (Cocco, Mannaro, Concas, & Marchesi, 2011) 12
Figure 5 - List of sub-tasks for each development phase 14
Figure 6 - Overview of Model Structure 15
Figure 7 - Timeline within Development Cycle 16
Figure 8 - Timeline of parallel Build Development and Testing 16
Figure 9 - System Dynamics representation of Materiel Solution Analysis development 17
Figure 10 - System Dynamics representation of Technology Development 18
Figure 11 - System Dynamics representation of Concept Planning 19
Figure 12 - Interaction between Concept Planning and Concept Development 20
Figure 13 - System Dynamics representation of Waterfall development 21
Figure 14 - System Dynamics representation of product planning and tasking for Agile development 22
Figure 15 - System Dynamics representation of Agile development 24
Figure 16 - Coflow for managing TRL during Agile development 25
Figure 17 – Overview of System Testing 26
Figure 18 - System Dynamics representation of Integration/Verification Testing 27
Figure 19 - System Dynamics representation of Contractor Testing 28
Figure 20 - System Dynamics representation of Government Testing 29
Figure 21 - System Dynamics representation of Operational Testing 30
Figure 22 - Priority for allocating System and Software Engineers 32
Figure 23 - Design Defect Rate Curve 33
Figure 24 - Code Defect Rate Curve 34
Figure 25 - Concept Defect Rate Curve 35
Figure 26 - System Test Defect Rate Curve 36
Figure 27 - Defect Rate Factor over Time 36
Figure 28 - Productivity Rate over Time 37
Figure 29 - System Engineering Involvement in Test Period 38
Figure 30 - Requirements Completed in Build 1 41
Figure 31 - System Engineers supporting System Design in Build 1 42
Figure 32 - Software Engineers supporting System Design in Build 1 42
Figure 33 - Requirements in Progress in Build 1 43
Figure 34 - Requirements Completed in Build 2 44
Figure 35 - System Engineers supporting System Design in Build 2 44
Figure 36 - Software Engineers supporting System Design in Build 2 45
Figure 37 - Requirements in Progress in Build 2 45
Figure 38 –Requirements Completed in Build 3 46
Figure 39 - System Engineers supporting System Design in Build 3 47
Figure 40 - Software Engineers supporting System Design in Build 3 47
Figure 41 - Requirements in Progress in Build 3 48
Figure 42 - Engineers supporting Technology Development 49
Figure 43 - Engineers supporting Materiel Solution Analysis 50
Figure 44 – Technologies introduced during each Build's test phase of Waterfall 52
Figure 45 – Technologies introduced during each Build's Test Phase of Agile 52

Table of Tables

Table 1 - Comparison of Product Development Processes (Ulrich & Eppinger, 2008) 6
Table 2 - Productivity metrics for System Design Tasks 33
Table 3 - Technologies considered for Build Development 39
Table 4 - Size of Development Team 39
Table 5 – Associated System Testing phase for completed requirements 39
Table 6 – Number of Requirements introduced during each Build's Testing phase 40
Table 7 – Average TRL for Requirements added to each Build 40
Table 8 – Number of Technologies Adopted in a Build 51
Table 9 - Requirements introduced in each Build's Test Phase 53
Table 10 - TRL of Requirements introduced in each Test Phase of Build 1 53
Table 11 - Comparison of technologies developed based on average TRL 54

Resource Dependencies in Parallel Development of Military Systems: A Comparison of Waterfall and Agile Development Methodologies

By

Erik Roberto Garcia

Submitted to the System Design and Management Program
In Partial Fulfillment of the Requirements for the Degree of
Master of Science in Engineering and Management

Abstract

The United States Department of Defense has been plagued with failing programs that are over budget, behind schedule, and exhibit poor performance during testing. Once a program has cost, schedule, or capability issues, follow-on development efforts adopt the underlying issues only to reinforce poor performance. To address these issues that lead to firefighting, one option is to use an Agile software development process to introduce capabilities earlier in the development process for effective testing. Two System Dynamics models compare Agile with the traditional Waterfall development approach to determine: if Agile development reduces the conflict between cost, schedule, and capability constraints; if Agile development reduces firefighting; and will Agile development impact other development tasks. Based on the simulation of each model, Agile did improve the dynamics of parallel development cycles by maximizing the productivity of the entire development team. Under the same System and Software Engineering team size and development release schedule, Agile development increases the quantity of requirements introduced within a development cycle. However, Agile development emphasized less on maturing new technologies leading to considerably less innovative builds.

Thesis Supervisor: Nelson P. Repenning
Title: Professor of System Dynamics and Organization Studies
Faculty Director, MIT Executive MBA Program

Acknowledgements

I gratefully acknowledge the helpful support and valuable resources provided by MIT's System Design and Management (SDM) Program. The SDM staff and faculty made my experience at MIT a truly enjoyable and rewarding one. In addition, I would like to thank all of my classmates, especially those in the Military, for providing their insight and recommendations on the models.

I owe a debt of gratitude to my thesis advisor, Nelson Repenning, for his considerable support during the development of the System Dynamics models and insights on fire fighting and resource dependencies.

I also thank Professor Edward Crawley and Dr. Bruce Cameron for the opportunity to be a teaching assistant for System Architecture.

I also am gratefully to my Raytheon product leads, Chibl Nahas and Jaime Wiley, for making my attendance to MIT possible while working, especially with classes during business hours and when the program's need was high.

Finally, my deepest thanks goes to my wife, and fellow SDM classmate, Kathleen Voelbel. She has given me the strength and support to complete this degree.

1 Introduction

Developing a new United States military system requires leadership, management, and execution. To realize and maintain a large-complex system, it is necessary to manage competing project constraints of scope, quality, schedule, budget, resources, and risk (Project Management Institute, 2013). Whether the system is managed as part of a project, program, or a portfolio, decisions to influence one constraint affect the others because they are interrelated. Therefore, it is important to have a business strategy and determine which constraints are most important when faced with development challenges.

Regarding the phrase "Good. Fast. Cheap. Pick any two.", a successful business must ensure "good" products and should determine flexibility in increasing schedule or cost (Frand, 1980). The value of a military system is determined by its capability to support the warfighter. It is the responsibility of the Department of Defense acquisition enterprise to oversee the procurement of systems and ensure reasonable system requirements are set. If a military system faces challenges due to development risk, the business strategy should determine which of the following actions the government project/program office should select: inject the needed funding to maintain schedule, inject a portion of funding to reduce schedule delay, or inject minimal funding required to support schedule delay.

However, allowing schedule delay or providing additional funding contradicts recent direction from the Department of Defense. Due to fiscal hardship, the United States Congress requires significant reduction in procurement costs for military systems. On the other hand, the Secretary of Defense Chuck Hagel indicates that the military needs to develop more technologically advanced equipment and weapons systems to support evolving military threats (Parrish, 2013). Depending on the size of the system, the acquisition process can take many years for a system to be developed, tested, manufactured, and deployed.

Unfortunately, the acquisition process for large systems has been failing. A 2010 Army Acquisition Review showed Acquisition Category 1[1] programs in development had an average schedule slippage of

[1] An Acquisition Category I program is either designated by the Under Secretary of Defense for Acquisition, Technology and Logistics as a Major Defense Acquisition Program; or requires more than $365 million for initial development and testing, or requires more than $2.19 billion to support development through production (based on fiscal year 2000 US dollars). (ACQuipedia, 2013)

2.1 years and that between 1990-2010, 22 Major Defense Acquisitions Programs were terminated (Decker & Wagner JR., 2011). In a separate assessment charted by the Under Secretary of Defense for Acquisition, Technology and Logistics, 67 major programs experiencing schedule delays were examined in response to acquisition community concerns that testing drove undue scope, schedule delays, and excessive costs. The author of this assessment states that "programs are most often delayed because of the results of testing, not the testing itself" (Gilmore, 2011). Both studies agree that poor performance experienced in testing is due to development efforts citing reasons of "unconstrained requirements, weak trade studies, and erosion of the requirements and acquisition workforce" (Decker & Wagner JR., 2011).

The reasons cited for poor performance are also cited as indicators of "firefighting" experienced in Integration & Testing and Product Deployment phases of aerospace product development (McQuarrie Jr., 2004). For programs that require multiple development cycles, the development team will focus vital resources on addressing the performance problems during current testing instead of devoting resources to early development that ensures quality performance of the next cycle's test phase. The dynamics of firefighting in parallel development, as is present in programs with multiple development cycles, is self-reinforcing (Repenning, 2001) and leads to an increasing overall schedule delay if left unaddressed.

Similar to the idea behind Technical Debt (Cunningham, 2011), Test Debt, as the author of this thesis defines it, occurs when capabilities are introduced late in the development cycle and consequently are not tested early. As Test Debt increases in the system, the risk of defect detection during final testing or, even worse, defect detections during deployed operation increases. The greater the Test Debt, the more likely firefighting will ensue and the customer's confidence in the system's performance will decrease to the point that additional "builds" are needed to "clean-up" defects discovered late in the lifecycle.

1.1 Motivation

Developing new military systems challenges all six project management constraints: scope, quality, schedule, budget, resources, and risk (Project Management Institute, 2013). However, when three particular constraints do not go as planned, i.e., the combination of cost overruns, schedule delays, and poor system performance, a burning-platform is created for the DoD to "shed excessive requirements"

and identify "trade-offs between cost, schedule, and performance" (Carter, 2011). In these situations, the government acquisition entity can reduce the initial set of requirements to a fundamental prototype that supports future upgrades and provides the minimum capability needed to assess early system performance, thereby mitigating potential cost, schedule, or performance issues. If the initial development cycle is successful in meeting objectives, the government acquisition entity can exercise the option for hardware and/or software upgrades in subsequent parallel development cycles. This process is similar to engineering real options that reduce risk and financial loss (De Neufville, 2011).

Although having reasonable and mature requirements does help reduce risk of failure during testing, the prototype may still exhibit poor performance that would lead to fire-fighting during parallel development cycles. Instead of de-scoping, increasing budget, or allowing schedule delay, one option to balance fire-fighting would be to alter the development approach. If decreasing Technical Debt reduces fire-fighting, one option could be to alter the software development approach from the traditional sequential Waterfall software development and encourage the rapid incremental Agile software development. Agile software development approaches like Scrum and Extreme Programming were developed to increase immediate development results, improve software quality, and adapt to changing requirements and are widely used in the commercial sector. Although Waterfall software development compliments the Defense Acquisition process and milestones, Agile software development can succeed in software development efforts of military embedded systems both small and extremely large complex systems (Tavassoli, 2007), (O'Connell, 2011).

The first two Laws of Program Evolution state that a system in use is changing continually and only increases in complexity (Lehman, 1980). Agile development addresses the evolution and complexity of a system by strong team interaction and incremental design and implementation of each new feature. However, Agile development's reliance on strong team interaction can have a negative effect since it increases the complexity of coordination, especially for large teams developing complex and complicated systems. In addition, the process for designing and implementing a fraction of a capability lacks the holistic view of the full solution which challenges experienced reviewers in ensuring proper execution of system-level requirements and preventing future re-design and re-implementation.

Although Agile development is successful in commercial software systems, the application of Agile development in the defense sector should be compared with traditional Waterfall development to determine the impact on a parallel product development environment that requires extensive testing.

1.2 Research Question

To determine whether Agile development should be used during parallel software development cycles of military systems, the following research questions are posed:

- Does Agile development reduce the conflict between cost, schedule, and capability constraints?
- Does Agile development create an environment that reduces firefighting?
- Will Agile development impact other development tasks?

1.3 Thesis Organization

To address these research questions, this thesis is organized into the following chapters:

- Chapter 1, Introduction, describes the background, motivation, and research questions of this thesis.
- Chapter 2, Background, defines the development process for military systems and relevant models.
- Chapter 3, Model Development, describes the modeling process that created the System Dynamics models used for analysis.
- Chapter 4, Analysis of Waterfall and Agile development Models, compares the results of both development processes during parallel development life cycles.
- Chapter 5, Conclusion, evaluates how the research objectives were met, suggests potential future work, and presents the key contributions of this thesis.

2 Background

Government and Defense Contractors adhere to the Defense Acquisition System for managing United States defense technology projects and acquisition programs (DoD Instruction 5000.2, 2013). The Defense Acquisition System along with the Defense Acquisition Guidebook provides insight on the development process to achieve cost, schedule, and performance goals (Defense Acquisition Guidebook, 2013). In comparing Waterfall and Agile development, previous work provides insight in modeling each development approach and the dynamics between development processes within the development life cycle.

2.1 Department of Defense Development Life Cycle

The objective of the Defense Acquisition System is "to rapidly acquire quality products that satisfy user needs with measurable improvements to mission capability at a fair and reasonable price" (Defense Acquisition Guidebook, 2013). To do so, Government project/program offices and Defense Contractor/Suppliers follow the System Acquisition Framework shown in Figure 1.

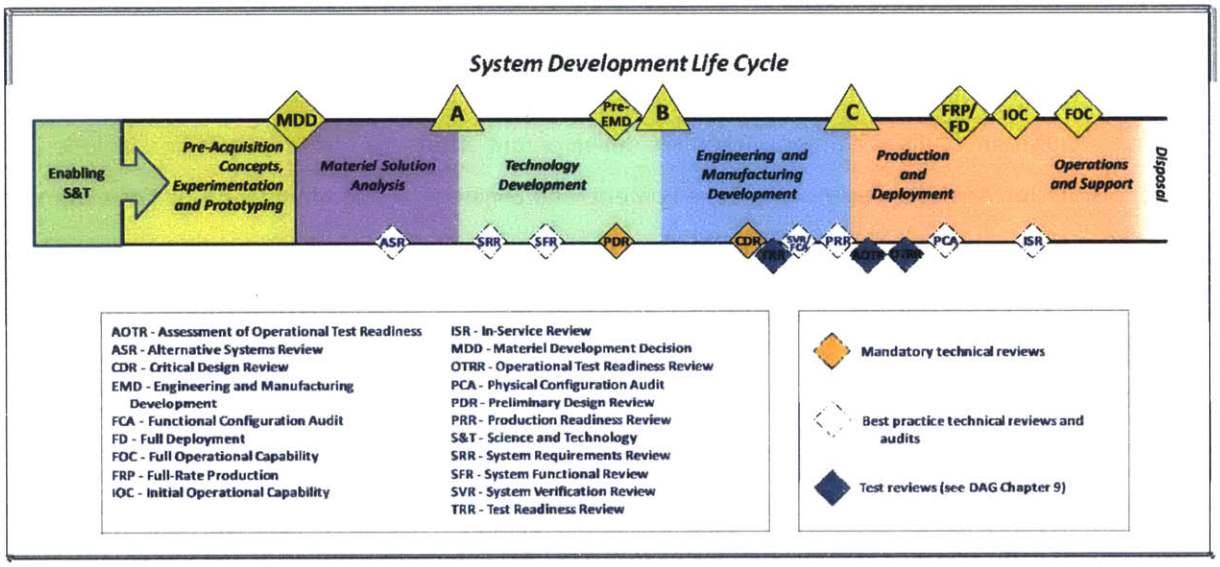


Figure 1 - System Acquisition Framework (Defense Acquisition Guidebook, 2013)

The Defense Acquisition Framework for Research, Development, Test & Evaluation programs is similar to commercial New Product Development. Table 1 compares the System Acquisition Framework phases with the Product Development and Development process described by Ulrich and Eppinger. The two major differences are: 1) the Ulrich & Eppinger processes couples design and implementation while the DoD System Acquisition Framework couples implementation with test and demonstration, and 2) the

DoD System Acquisition Framework includes post-production operations and support services as part of the Development process.

Table 1 - Comparison of Product Development Processes (Ulrich & Eppinger, 2008)

Ulrich & Eppinger	DoD System Acquisition Framework	Similarities:
Planning	Pre-Acquisition Concepts, Experimentation and Prototyping	Determining market/military need, assessing existing systems and alternative technologies, etc.
Concept Development	Materiel Solution Analysis	Identifying users and user needs, developing experimental prototypes, etc.
System-Level Design	Technology Development	Develop architecture and interfaces
Detail Design	Engineering Development - Integrated System Design	Develop and document final design of product/system
Testing & Refinement	Engineering Development – System Capability & Manufacturing Process Demonstration	Testing the product/system design and implementation, and validate product/system meets needs
Production Ramp-Up	Production & Deployment Phase	Production and distribution of the product

For each System Acquisition Framework phase, a system passes each stage of pre-systems acquisition, systems acquisition, and sustainment. The core development efforts for system acquisition include: Materiel Solution Analysis, Technology Development, and Engineering and Manufacturing Development. Each effort increases the maturity of a technology for future capability introduction. Currently, the DoD measures the maturity of a technology using the Technology Readiness Level (TRL) metric.

The TRL scale ranges from 1 to 9 and is a metrics-based indicator associated with the maturity and risk of a technology. TRL 1-3 represent Research & Development technologies that have positive laboratory results. TRL 4-6 represent technologies that have been successfully modeled or prototyped in a relevant environment. TRL 7-9 represent technologies that have been integrated into an actual system and have successfully completed mission operation (Technology Readiness Assessment (TRA) Guidance, 2011).

2.1.1 Materiel Solution Analysis

Materiel Solution Analysis is the first element of the Defense Acquisition Framework. The goal of Materiel Solution Analysis is to identify and define performance requirements and

technology/technologies for a system. This effort includes further developing the maturity of a technology; identifying implementation and manufacturing risk; and possible prototype/demonstration.

2.1.2 Technology Development

Technology Development is the second element of the Defense Acquisition Framework. The goal of Technology Development is to reduce technology risk for future system adoption. This effort may include further technology development and/or demonstration of application. A Preliminary Design Review (PDR) occurs during this phase.

2.1.3 Engineering & Manufacturing Development

The Engineering & Manufacturing Development phase is the third element of the Defense Acquisition Framework. The Engineering & Manufacturing Development is comprised of Integrated System Design and System Capability & Manufacturing Process Demonstration with each effort occurring before and after the Critical Design Review (CDR), respectively. Unlike Ulrich and Eppinger's approach to coupling the design and implementation during the Detailed Design phase, the Integrated System Design phase refines and documents the system, performance, and/or functional requirements. Implementation occurs in the System Capability & Manufacturing Process Demonstration effort along with the integration and demonstration of capabilities. In addition, this phase verifies the manufacturability, affordability, and producibility of the system.

2.1.4 Role of System Engineers

System Engineering is the "holistic, integrative discipline, whereby the contributions across engineering disciplines such as Structural Engineers, Electrical Engineers, Mechanical Engineers, Software Engineers, Human Factors Engineers, and Reliability Engineers are evaluated and balanced to produce a coherent capability - the system" (Defense Acquisition Guidebook, 2013). The role of a Systems Engineer spans across the entire life cycle and includes the realization of the architecture; modeling and prototyping of designs; developing system, performance, and functional requirements and "build-to" documentation; supporting development testing; and preparation for production.

2.2 Waterfall and Agile Development

As described by Barry Boehm in his 2006 paper titled "A View of 20th and 21st Century Software Engineering", Software Engineering processes have evolved every decade, cycling between formal and

informal development. During the 1950's, Software Engineering followed the same exhaustive reviews that hardware engineers performed before committing to a design. The use of a strict development approach was deemed not necessary in the 1960's because software defects are easy to fix, leading to a "code and fix" model. Unfortunately, the "code and fix" model led to "spaghetti code" and relied on "cowboy programmers" to address issues; this led to the 1970's effort in promoting a formal structured, sequential Waterfall development approach. Later in the 1980's, the U.S. DoD coupled the Waterfall development structure to contractual standards and commissioned the Carnegie Mellon University – Software Engineering Institute to develop the Software - Capability Maturity Model. Concurrently in the 1980's, Software Engineering initiatives to improve productivity and scalability through distributed efforts, such as early testing, and promoting development tools, software processes, and software re-use.

Starting in the 1990's, the market shifted to developing quick solutions to meet customer needs. This emphasis on "Time-To-Market led to risk-driven spiral development. Subsequently, in the 2001, rapid development approaches integrated into the Manifesto for Agile Software Development (The Agile Manifesto, 2001). The Manifesto for Agile Software Development values are:

- "Individuals and interactions over processes and tools"
- "Working software over comprehensive documentation"
- "Customer collaboration over contact negotiation"
- "Responding to change over following a plan"

2.2.1 Waterfall Development

The Waterfall development approach consists of sequential development phases that are often separated by a formal review. The approach in developing large software systems was introduced in 1970 by Dr. Winston Royce, as shown in Figure 2 (Royce, 1970). In 1976, "waterfall" was coined in an article discussing development of software systems for Ballistic Missile Defense (Bell & Thaer, 1976).

The purpose of sequential development is to reduce the number of defects identified in later development stages. For early system requirements and design tasks, Defense Acquisition Framework promotes Preliminary and Critical Design Reviews to ensure the design embodies the capability needed and to reduce risk of rework during implementation of the design. By approaching the concept as a whole, the design team can determine whether the solution is optimal and is aware of the architectural

impact of the solution on other concept solutions. This strategy is important since a requirement's rework can cost 50 to 200 times to fix during implementation or deployment of the software (McConnell, 1996).

Another aspect of Waterfall is that the design of a technology is managed in multiple forms of documentation to ensure the intent is conveyed during design reviews and to other engineers involved in the design, implementation, and test efforts.

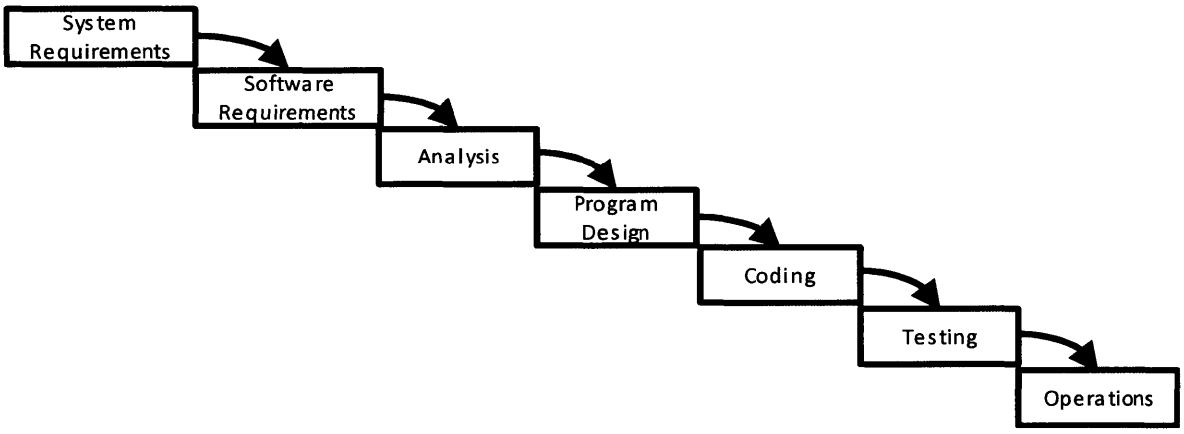


Figure 2 - Implementation flow during development of a concept/software system (Royce, 1970)

2.2.2 Agile Development

Unlike the sequential effort of a concept during Waterfall development, Agile development approach consists of incremental parallel development efforts. Figure 3 illustrates the concurrent development of increments of multiple concepts being developed simultaneously. The development of each increment can occur between one to six weeks, which may include the requirements development, design, implementation, testing, documentation, and deployment (MITRE, 2010). This development approach revolves around principles that promote interaction between customers and developers and leads to flexibility with changing requirements and early releases of software delivery (The Agile Manifesto, 2001).

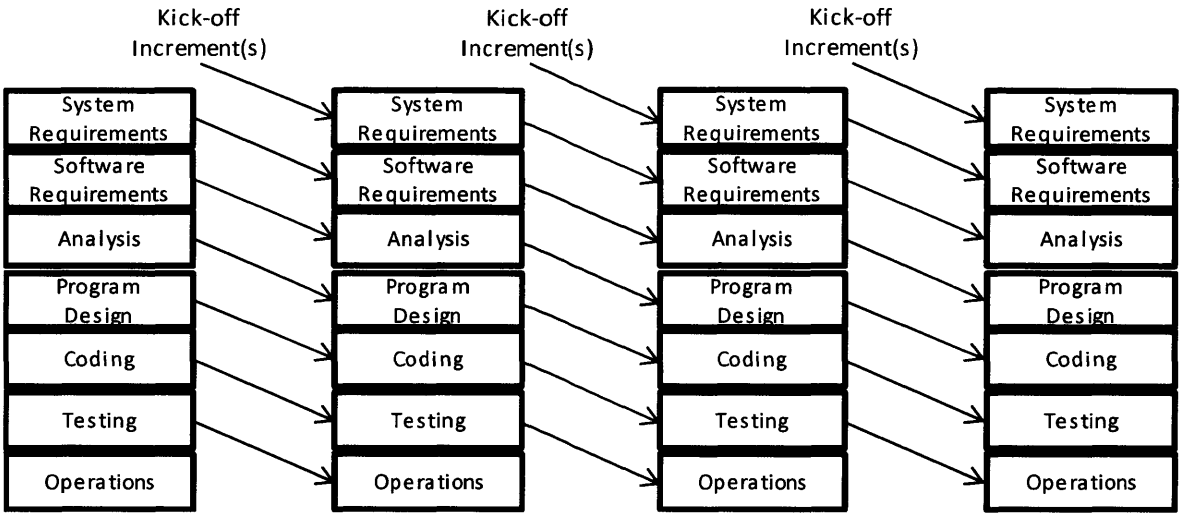


Figure 3 - Implementation flow during development of a increments of a concept based on Figure 2

There are different methods of Agile development such as: Extreme Programming, Scrum, Dynamic Systems Development Method (DSDM), Adaptive Software Development, Crystal, Feature-Driven Development, and Pragmatic Programming. Of the different methods, Scrum (50%) and hybrids of Scrum (25%) account for 75% of Agile development methods used in industry (Version One, 2009).

2.3 Previous Relevant Models

There are three System Dynamics models that influenced the research in the dynamics of parallel product development using Agile software development. The first model provided insight on modeling parallel product development and the dynamics of managing current and future software development efforts (Rahmandad & Weiss, 2009). The second and third models provide framework on modeling the Agile software development process (Luisanna Cocco, 2011) (Glaiel, Moulton, & Madnick, 2013).

2.3.1 Dynamics of Concurrent Software Development

Using System Dynamics to model the dynamics of parallel software development, (Rahmandad & Weiss, 2009) investigated two Software teams (alpha and beta) within the same organization to identify why only one team was successful at providing a quality product within cost and schedule constraints.

Through interviews conducted by Rahmandad and Weiss, it was apparent that the management styles were different between alpha and beta. Both alpha and beta had high priority in ensuring customer satisfaction by developing features and improving development capability. However, alpha often used

current development resources to quickly address defects in the field. In contrast, Beta gave higher priority to current development and capability building.

Using System Dynamics, the authors modeled firefighting in concurrent software development and identified that the combination of scope, work pressure, work quality, and resource allocation policies can reinforce firefighting. Although the intentions of customer satisfaction were the same between teams, the policies employed by the alpha team led to excessive refactoring, yielding ineffective and inefficient team performance. The policies employed by the beta team yielded successful development productivity and led to long-term customer satisfaction.

2.3.2 Simulating Kanban and Scrum vs Waterfall with System Dynamics

Using a single System Dynamics model, the dynamic behavior of Agile development styles of Kanban and Scrum are compared with the traditional waterfall approach (Cocco, Mannaro, Concas, & Marchesi, 2011). By using different input parameters, their data suggest that Lean-Kanban approach is more efficient than Scrum and Waterfall.

However, the assumption to consolidate all phases of planning, design, coding, and testing cannot be applied directly to the development process of military systems. The Defense Acquisition Process has a considerable amount of formal and informal reviews to reduce risk and ensure that the design and implementation of capabilities integrate within large complex, and sometimes extremely complicated, systems. For this reason, the Agile development framework of Scrum, shown in Figure 4, will be used to model Agile development within the Defense Acquisition Framework.

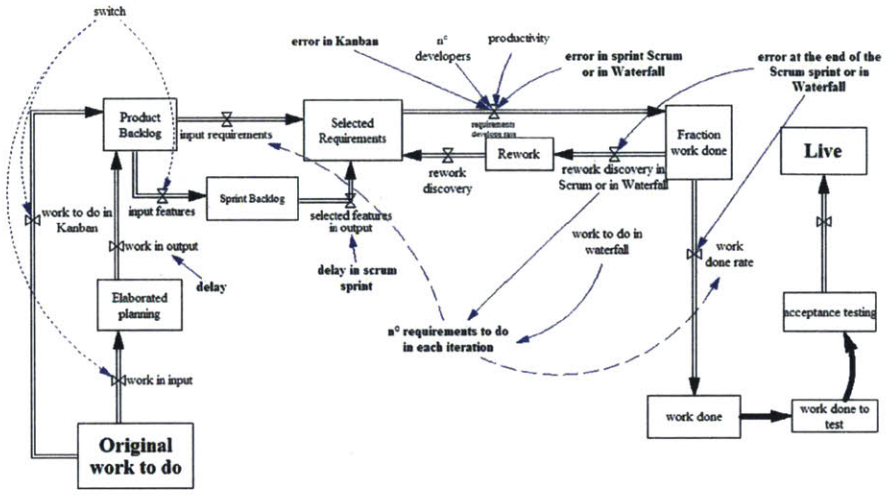


Figure 4 - Agile development framework of Scrum (Cocco, Mannaro, Concas, & Marchesi, 2011)

2.3.3 Agile Project Dynamics

Similar to the prior modeling effort of Kanban, Scrum, and Waterfall, the Agile Project Dynamics (APD) model compares Agile development with Waterfall (Glaiel, Moulton, & Madnick, 2013). However, APD incorporates a set of Agile practice and management policy attributes, called "genes", that capture the dynamics of system development. Through experimentation, their analysis concludes that there are opportunities to improve cost, schedule, and quality through certain combination of Agile practice and management policy.

In addition to identifying and experimenting with Agile practice and management policies, they identify several difficulties for applying Agile software development methodologies to the Defense Acquisitions process. Several of these reasons include: conflicting stakeholder interests, domain of the contractor, program control processes, and level of interaction between government and defense contractors during development.

3 Model Development

This chapter describes the process for modeling the parallel product development of military systems for comparing Waterfall and Agile software development processes. The basis for both System Dynamics models stem from the defense acquisition process as discussed in Section 2.1. Of the six phases in the Defense Acquisition System, the three phases that capture the engineering effort in developing follow-on software builds are: Materiel Solution Analysis, Technology Development, and Engineering & Manufacturing Development.

To facilitate the modeling effort, the Materiel Solution Analysis and Technology development phases were simplified to a set of 3 sub-tasks each, as listed in Figure 5. The Engineering & Manufacturing Development phase consists of the Integrated System Design and System Capability & Manufacturing Process Demonstration, which are separated by the Critical Design Review. However, for the purpose of modeling separate development and testing of a system, the Engineering Development effort is split between the System Design and System Testing efforts, which is consistent the Detailed Design and Testing & Refinement effort specified by Ulrich & Eppinger.

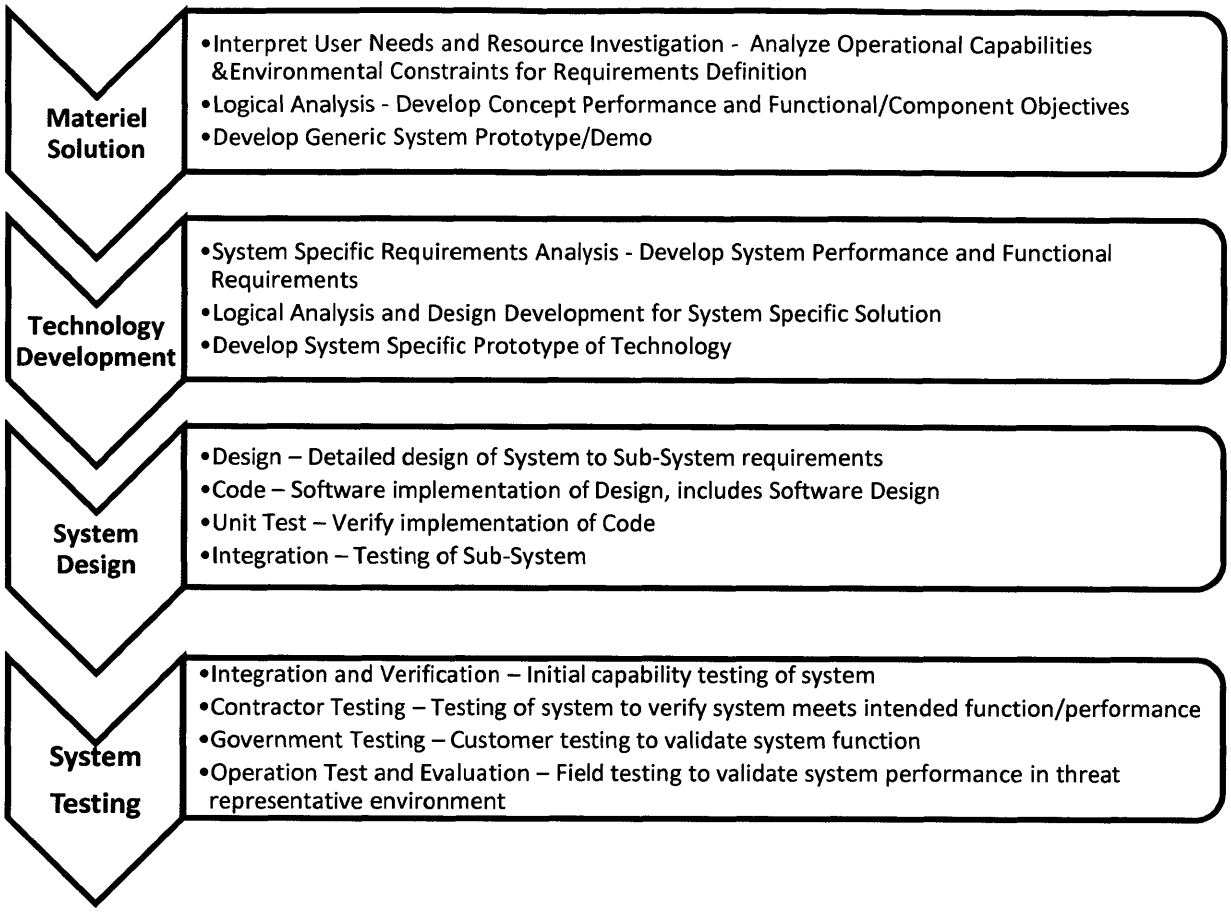


Figure 5 - List of sub-tasks for each development phase

The System Design phase includes the requirements design, software implementation, and sub-system testing of a technology. The System Testing phase represents the different stages of testing that ensure reliable mission critical performance. The first three sub-tasks of System Testing associate to "alpha" testing, internal "beta" testing, and customer "beta" testing that represent Testing & Refinement effort specified by Ulrich & Eppinger.

3.1 Development and Testing Timelines

Since the architecture of newly developed systems should anticipate obsolete hardware, both System Dynamics models are based on the introduction of new software-based features. These features are developed by joint System and Software Engineering efforts as discussed in Section 2.1.4. The scope of work modeled will include the test phase associated to the system's inception (Build 0) and three follow-on development cycles (Builds 1-3) as shown in Figure 6.

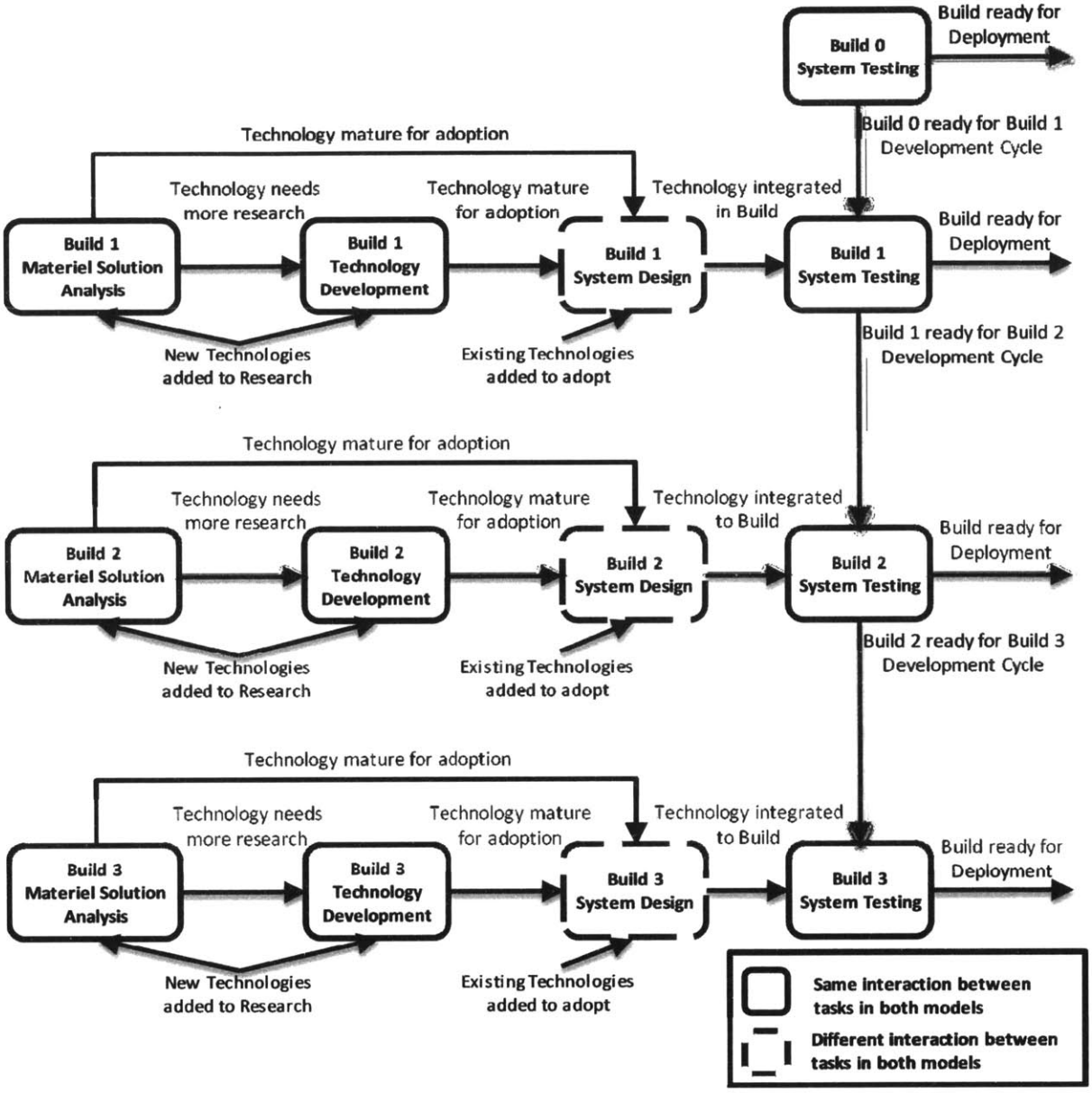


Figure 6 - Overview of Model Structure

The development cycle for each build can be conceptually categorized by two phases: Build Development and System Testing. The Build Development phase includes efforts pertaining to Materiel Solution Analysis, Technology Development, and Engineering Development – System Design. The System Testing phase includes the Engineering Development – System Testing phase. Since the introduction of capabilities is important, the Build Development phase overlaps with the System Testing phase as shown Figure 7.

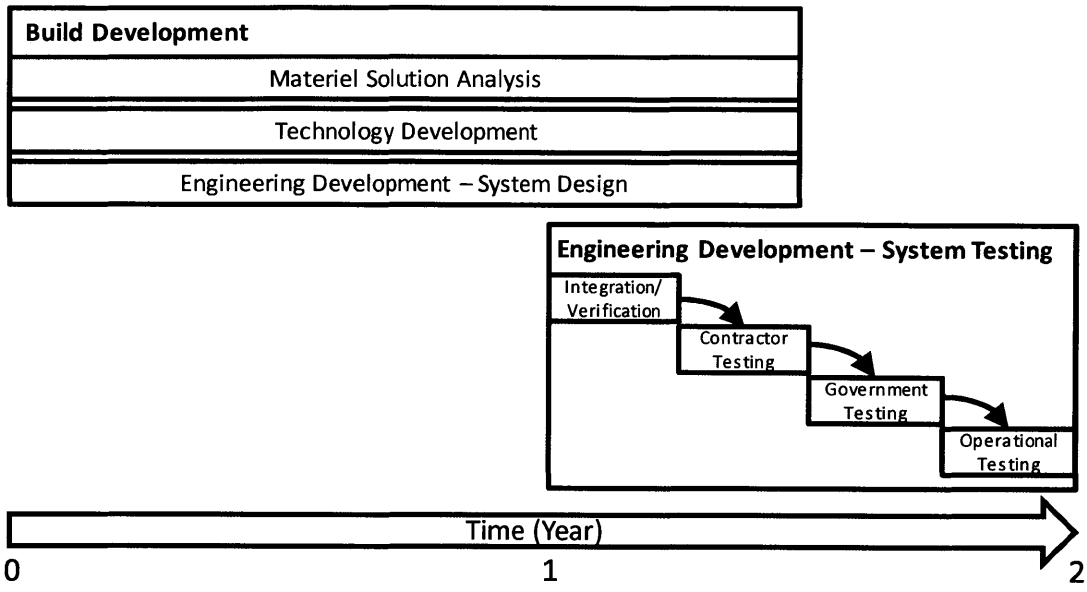


Figure 7 - Timeline within Development Cycle

The Build Development phase spans for 1.5 year and the Test Phase begins 1 year after the Build Development phase starts. Each build will overlap with the prior build for 1 year and with the subsequent build for the second year as shown in Figure 8.

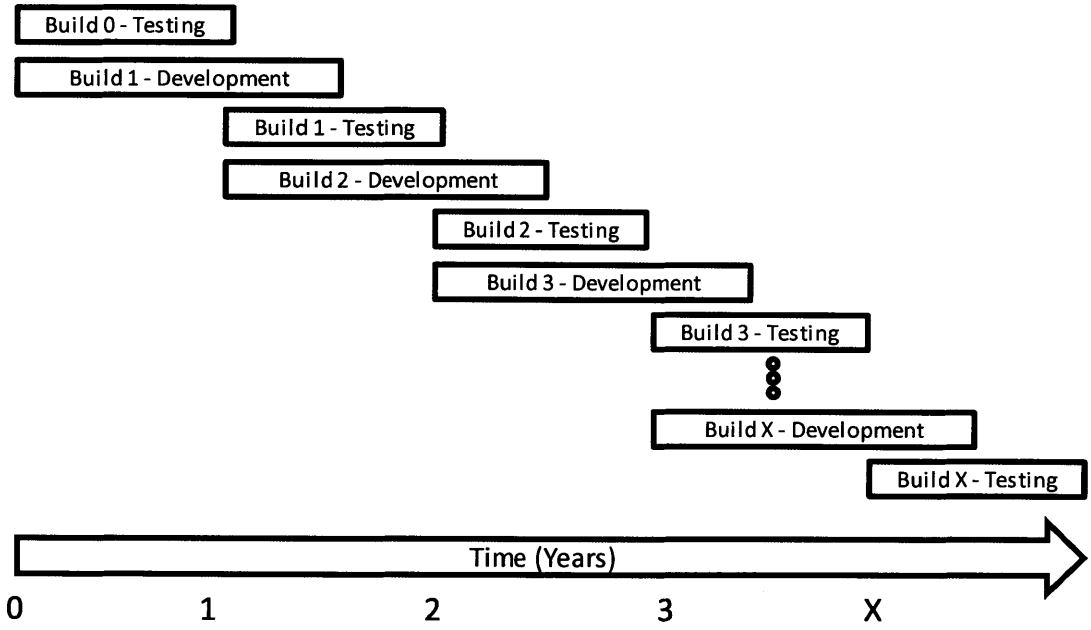


Figure 8 - Timeline of parallel Build Development and Testing

3.1.1 *Materiel Solution Analysis*

The Materiel Solution Analysis effort focuses on promoting cutting edge technologies for build introduction. Since these technologies are less developed, it is recommended that the technology undergo several iterations of design prior to being developed for build introduction. As part of the Materiel Solution Analysis process, a Technology, labeled as a "M.S. Concept", is developed by identifying the need, developing a solution, and generating some form of prototype/demo that describes the application of the technology as shown in Figure 9. The process is performed by System Engineers with an expected completion of 15 weeks. As indicated in Figure 6, the modeling of the Materiel Solution Analysis effort is identical between both models.

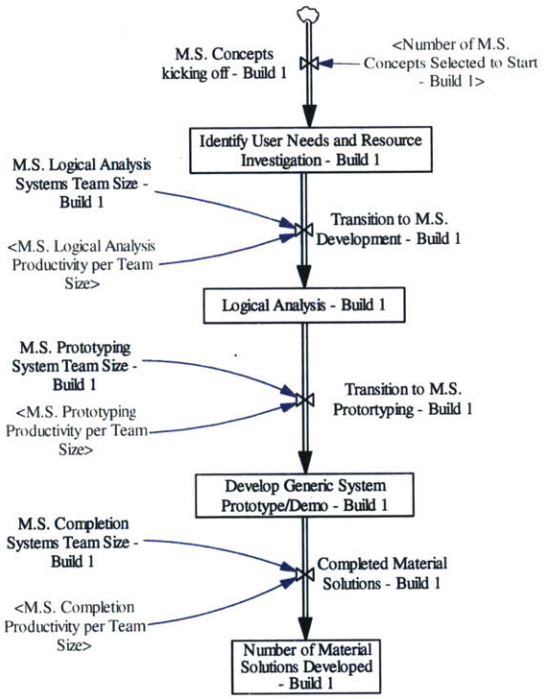


Figure 9 - System Dynamics representation of Materiel Solution Analysis development

3.1.2 *Technology Development*

The Technology Development effort focuses on promoting new technologies for build introduction. Since these technologies are lesser developed than technologies ready for immediate design and integration, it is recommended that the technology undergo an additional iteration of design prior to build introduction. As part of the Technology Development process, a technology, labeled as a Concept,

is developed by assessing impact on a specific system, developing a system-specific solution, and generating some form of prototype/demo that applies the technology to the system as shown in Figure 10. The process is performed by System Engineers with an expected completion of 15 weeks. As indicated in Figure 6, the modeling of the Technology Development effort is identical between both models.

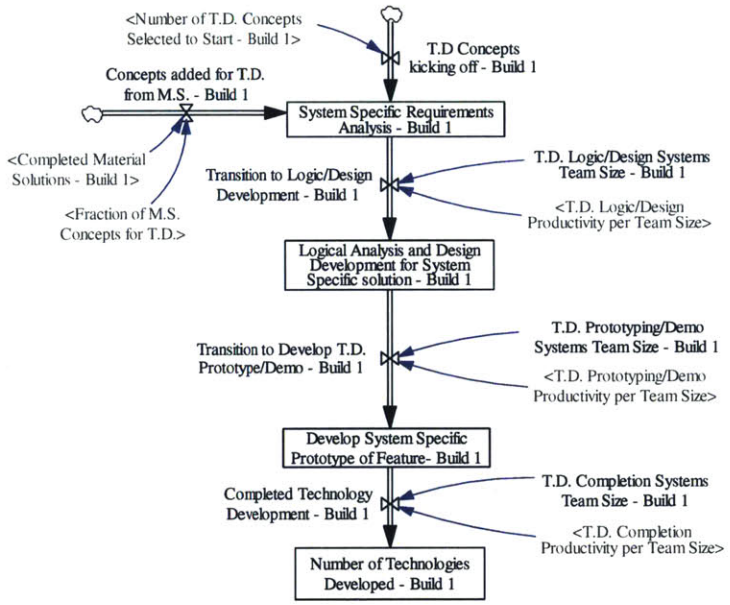


Figure 10 - System Dynamics representation of Technology Development

3.1.3 Engineering & Manufacturing Development – System Design

The System Design effort of the Engineering & Manufacturing Development phase is unique between Waterfall and Agile design processes. In the two sub-sections, two System Dynamics model are described to capture the difference between System Design processes. The interaction between Materiel Solution Analysis, Technology Development, and System Testing phases and the System Design phase is identical between both System Design models.

3.1.3.1 Waterfall Development

Waterfall Development is the standard development process for developing military systems since it best fits the emphasis on reviewing and testing required at each stage of the Defense Acquisition System. The unique aspect of Waterfall development is the focus on sequential development of a concept during each phase of the process.

3.1.3.1.1 Concept Planning

In addition to integrating new concepts developed in the Materiel Solution Analysis Development and Technology Development phases, the majority of concepts expected to be integrated in the current build are known technologies that can be applied to the system. In Figure 11, the Concepts for Development stock captures the list of concepts that are ready to be developed and integrated in the current build. Account for level of maturity of concepts in the development process, a "coflow" of the Technology Readiness Level is maintained to account for concepts being added, developed and completed in each build (Sterman, 2000). The Technology Readiness Level for new technologies added from Materiel Solution Analysis and Technology Development efforts are TRL 3 and TRL 6, respectively. The Technology Readiness Level for existing technologies added during Engineering Development is TRL 7.

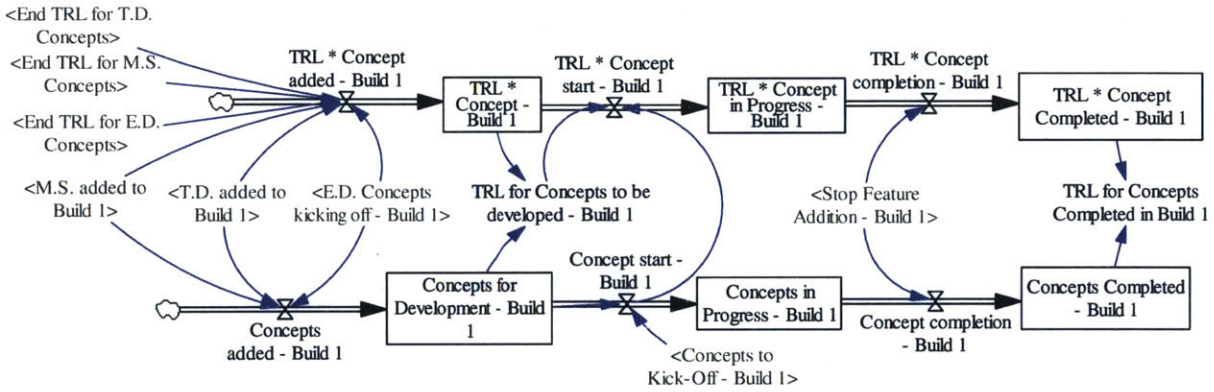


Figure 11 - System Dynamics representation of Concept Planning

When the Build Development is ongoing, new concepts are kicked-off and are categorized as "Concepts in Progress", as shown in Figure 12. Once the concept has been designed, implemented, and integrated in the build, a concept is considered complete. When the Government Test phase begins, the "Stop Feature Addition" variable is set causing all concept development tasks halts.

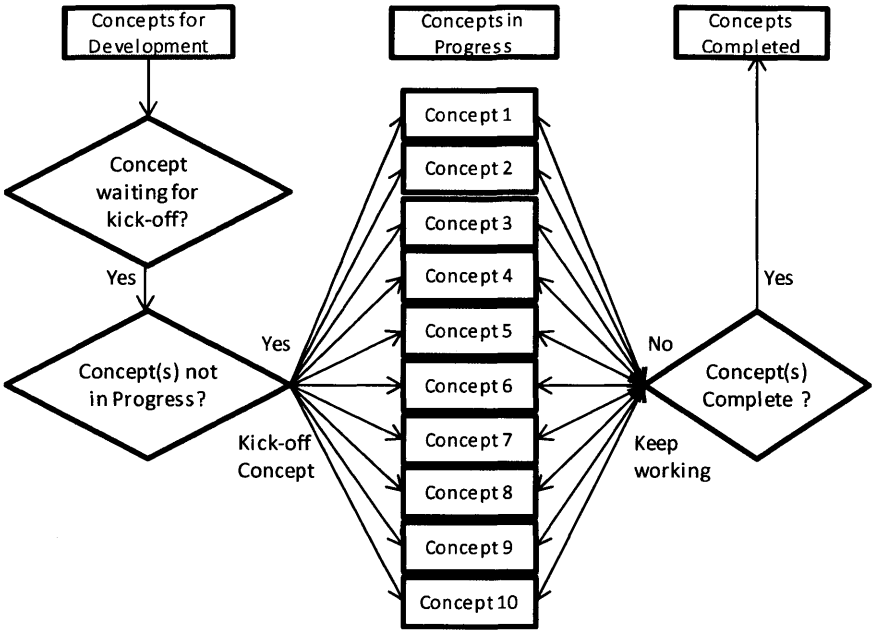


Figure 12 - Interaction between Concept Planning and Concept Development

3.1.3.1.2 Concept Development

For each concept in progress, Design, Code, Unit Test, and Integration efforts, as shown in Figure 13, are modeled independently, as shown in Figure 12. This ensures that the work performed in each phase of a concept adheres to the Waterfall development methodology.

The defense acquisition process requires includes multiple formal and informal design reviews that separate the design and implementation of a concept. Therefore, the Waterfall Concept Development process requires the completion of the Detailed Design, prior to coding the concept. Unlike the Agile Development Design effort, the Design effort includes the entire concept which shall be fully thought out and vetted through a formal review with all stakeholders. The painstaking emphasis of upfront development and reviews ensures the optimal concept solution and prevents impact of architectural changes from other competing concepts in development. Therefore, any Design Defects due to concept solution and architectural changes are encountered during the Detailed Design effort.

After the Initial Design is completed by System Engineers, the Software Engineering Code effort can begin. As the software code is developed, the code is unit tested and Code defects are returned to the Code effort to be addressed. For Code that does not require re-work, System Engineers can begin concept level Integration to validate the design and code meet the Concept intent. To capture potential

defects due to intent, Concept Defects are discovered during Integration and are allocated to the Design and Code efforts based on the Design Defect Rate.

Once the entire concept has been designed, coded, unit tested, and passed integration testing, the concept is considered complete and is ready to be added to the official build for System Testing. However, since System Testing progress is measured by System Requirements, each completed concept will be converted to five Requirements when added to System Testing.

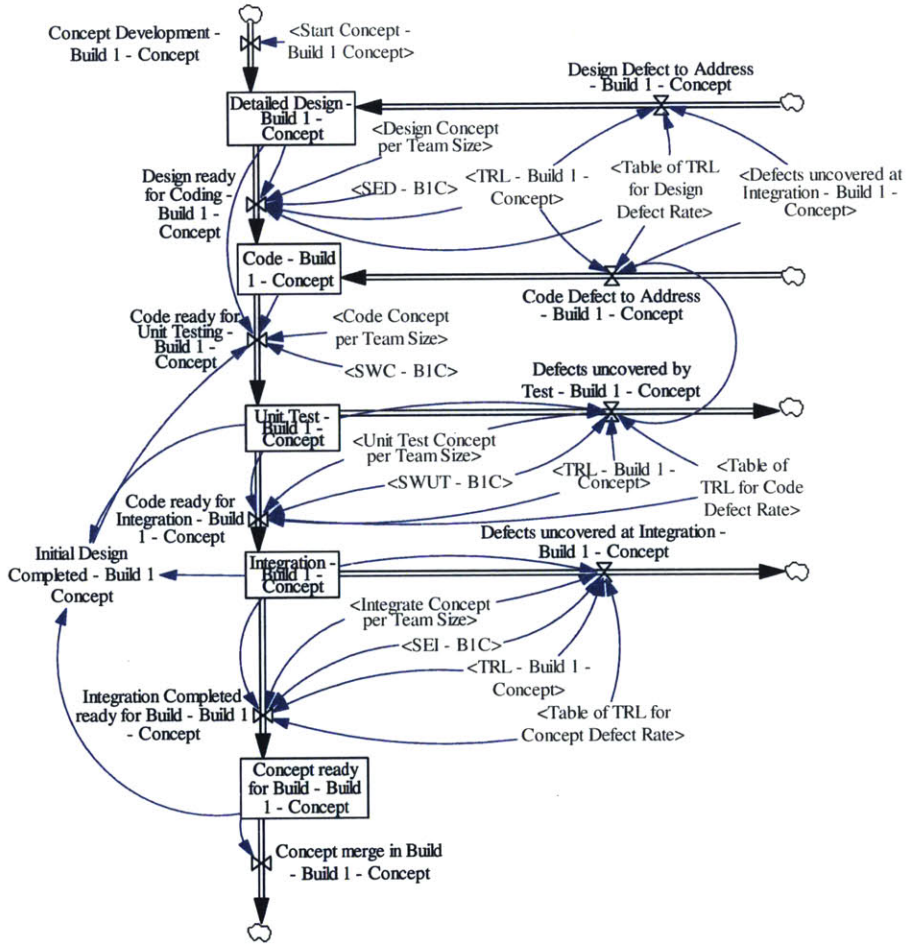


Figure 13 - System Dynamics representation of Waterfall development

3.1.3.1.3 Concept TRL and Defect Ratio

During Concept Development, each concept has an associated TRL. This TRL is assigned during Concept Kick-off and is based on the average "TRL of the Concepts for Development". The TRL of the concept is

used to determine the Design, Code, and Concept Defect Rates using the associated defect tables discussed in Section 3.1.6.2.

3.1.3.2 Agile Development

Agile Development is an incremental version of Waterfall Development. Instead of a full Concept being developed, an element of a Concept is developed at a time. To capture this incremental effort, the Agile model develops Concepts based on individual requirements. The requirements metric is consistent with the Waterfall conversion of 1 concept to 5 requirements.

3.1.3.2.1 Agile Planning

Similar to the Concept Planning representation of the Waterfall model, the Agile development model keeps track of each Concept but at the Requirements level. When a concept is added for the current build, five requirements are added to the Product Backlog as shown in Figure 14. The Product Backlog manages all requirements that are waiting for or are in development. For requirements being added or in process, the associated TRL is managed via coflow similar to Waterfall development. The Technology Readiness Level for new technologies added from Materiel Solution Analysis and Technology Development efforts are TRL 3 and TRL 6, respectively. The Technology Readiness Level for existing technologies added during Engineering Development is TRL 7.

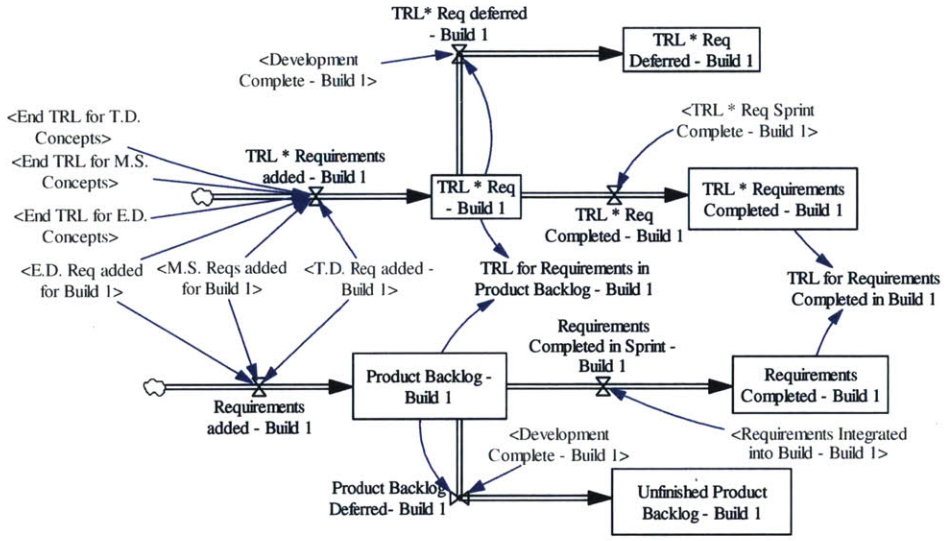


Figure 14 - System Dynamics representation of product planning and tasking for Agile development

While Build Development is ongoing, new requirements are kicked-off when the design team has work load less than 10 requirements in queue, maximum consistent with concurrent Waterfall concepts as shown in Figure 12. Once each requirement has been designed, implemented, and passed testing, the requirement is considered complete and is integrated in to the official build. When the Government Test phase begins, the "Stop Feature Addition" variable is set causing all concept development tasks halts.

3.1.3.2.2 Requirements Development

The Requirements Development process captures the Design, Code, Unit Test, and Integration efforts, as shown in Figure 15 and in Appendix 7.1. Since Agile development promotes concurrent development across efforts and tasks, it is not necessary to manage the development of each requirement separately. Contrary to Waterfall development, the development of all requirements is modeled in a single development process.

When Requirements Development begins, 10 requirements are added to the Detailed Design task. As time passes, System Engineers develop the "Build To" requirements documentation for the Software Engineers to code. Once a requirement is coded, the Software Engineer unit tests the code. Any coding defects revert back to the Code effort for re-work and non-defect code transitions to Integration Testing. The Integration Testing effort is performed by System Engineers and determines whether the requirement is ready to be integrated in the official build or if there are Concept defects. Concept defects are split between Design and Code phases in the same way they were done in the Concept Development process; however, concept level attributes are less defined during Agile development so a single time delay was added prior to demoting a requirement.

The second modeling difference between Agile development and Waterfall development is the discovery of Design Defects due to optimal final solution and architectural changes during development. Unlike Waterfall development's emphasis on designing and reviewing the full solution, Agile development is incremental making it difficult for all the stakeholders to review a requirement without having a full understanding of the final concept scope, especially since the number of reviews increases due to incremental steps. In addition, the architecture of the software system is continually changing between concept increments. Therefore, to capture the refactoring due to changing final solution and

architectural changes during the development of a concept, the discovery of Design defects occur during Integration Testing.

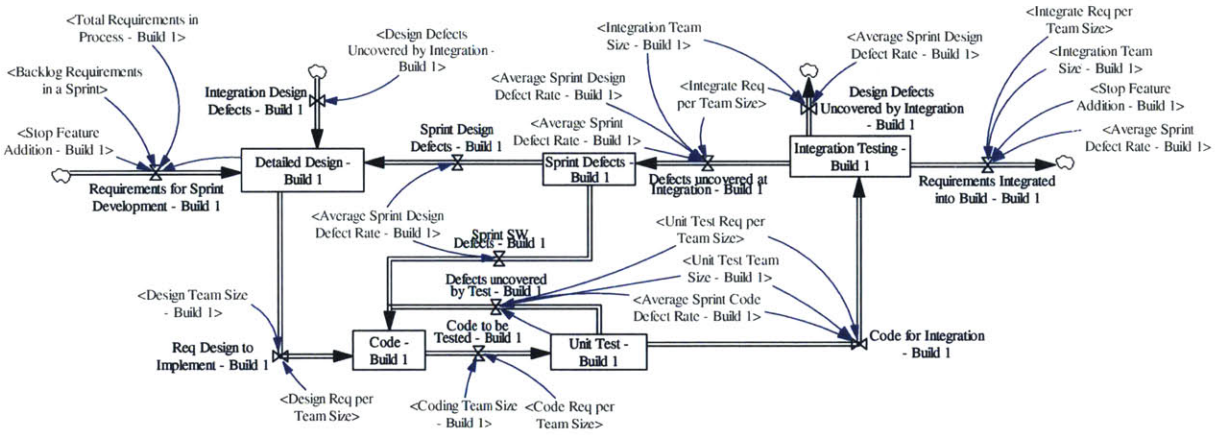


Figure 15 - System Dynamics representation of Agile development

3.1.3.2.3 Requirement TRL and Defect Ratio

During Concept Development, each requirement added has an associated TRL. This TRL is assigned during Requirements Kick-off and is based on the average TRL of the Product Backlog. The TRL must be managed for each requirement passing between phases. Therefore, a coflow was created for every stock as shown in Figure 16 or Appendix 7.2.

The TRL of each requirement is used to determine the Design, Code, and Concept Defect Rates using the associated defect table as discussed in Section 3.1.6.2 as well as the TRL of the requirements being integrated into the build.

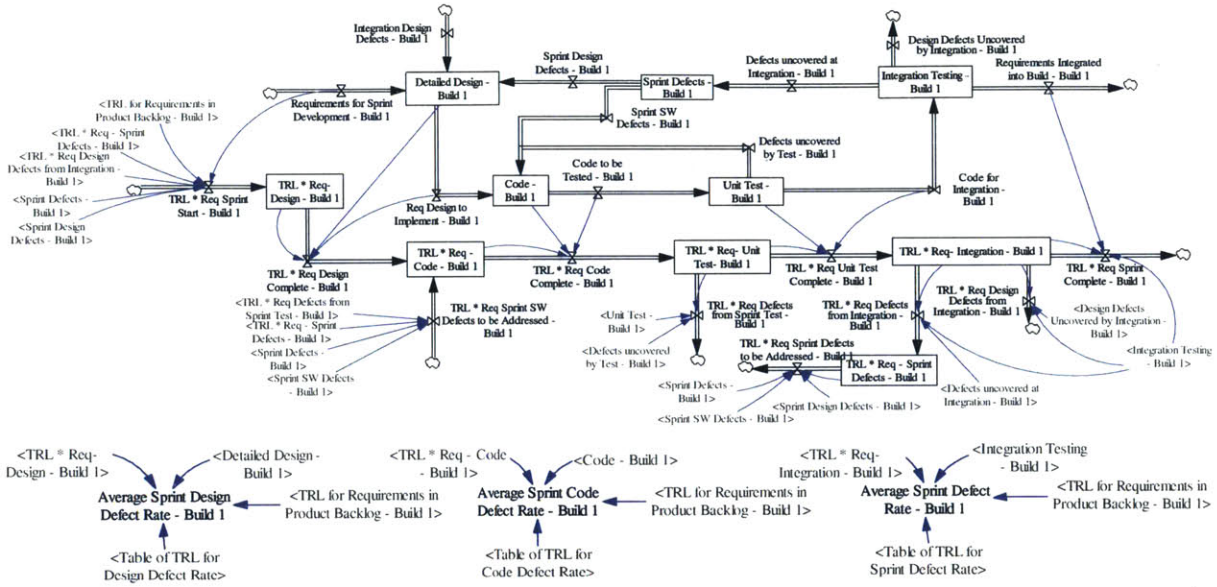


Figure 16 - Coflow for managing TRL during Agile development

3.1.4 Engineering & Manufacturing Development – System Testing

The System Testing effort ensures that the newly developed requirements improve the overall system capability without unintended system defects. The test process is broken into multiple milestones test events to assess the maturity of the build with new requirements. To generate data for analysis during each test phase, military assets and System Test engineers are required to simulate a controlled environment that represents the system's operational environment. The scheduling of these resources are done months or even years in advance causing strict schedule deadlines for the start and end of each test phase.

The structure of System Testing, shown in Figure 17, consists of the following 4 phases: Integration/Verification Testing, Contractor Testing, Government Testing, and Operational Testing. Each test event is sequential requiring only one test phase across all builds to occur at any given time. As the set of requirements are tested, requirement defects are uncovered and are either addressed during the current test phase or are deferred to be addressed in the subsequent test phase. Once a defective requirement is addressed, the requirement is ready for testing in the subsequent Test Phase.

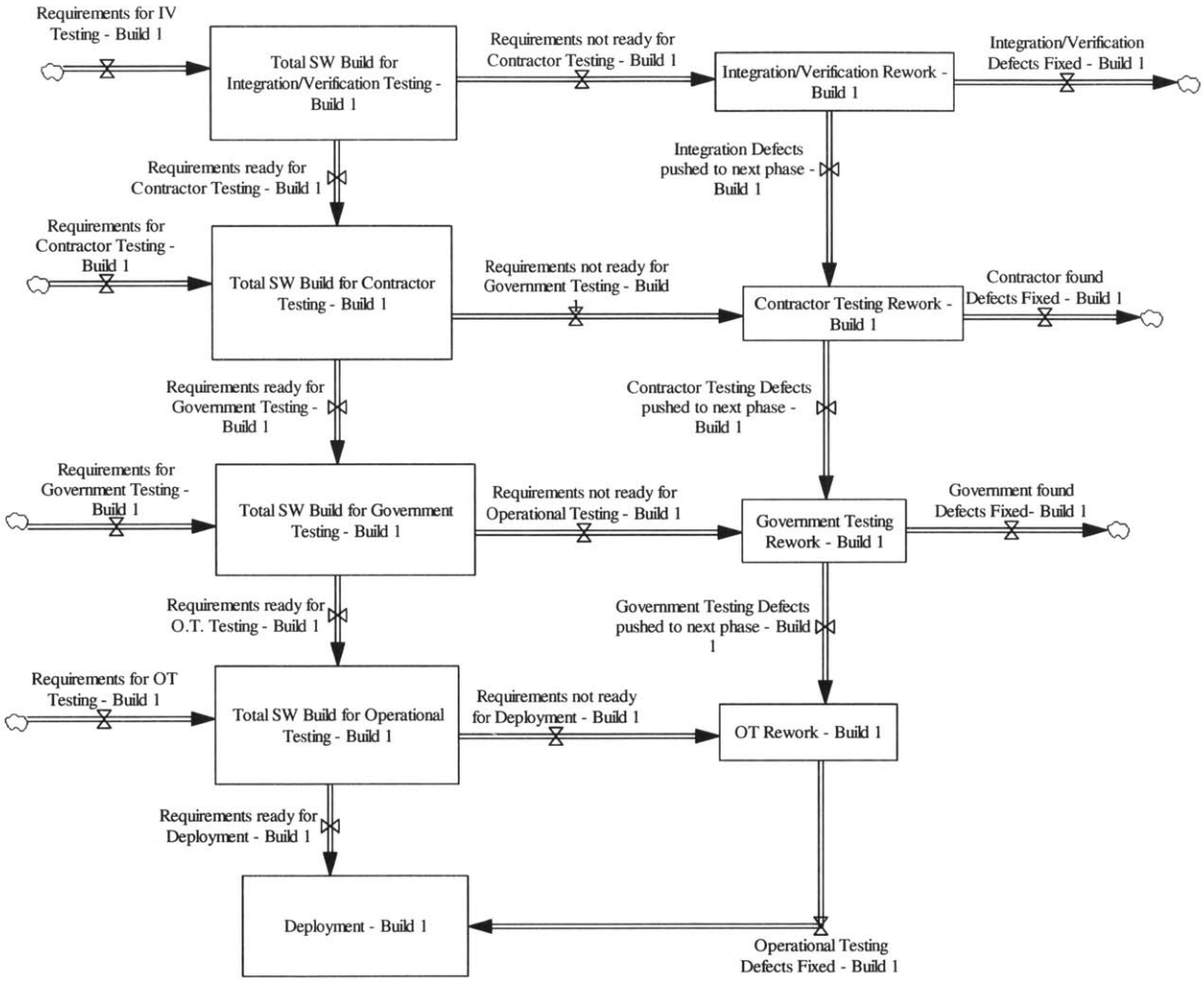


Figure 17 – Overview of System Testing

3.1.4.1 Integration/Verification Testing

Since the majority of the Concept or Requirement Development effort occurs prior to the Integration and Verification Testing phase, the majority of the requirements should be tested with representative hardware to uncover defects that can be addressed and re-tested before build completion. This effort provides insight on the stability of the new requirements developed prior to testing with legacy requirements from the prior test phase.

As shown in Figure 18, the Integration/Verification effort begins when the System Testing of the prior build is complete but, no earlier than a year after the Build Development started. Once the test phase begins (indicated by the IV Testing variable), the System Test team begins assessing the set of new and

legacy requirements via simulated testing. Based on the size and productivity of the team and System Test Defect rate, a number of requirements are passed for future testing while the defects are passed for Rework. The rate of requirements being analyzed during Integration/Verification Testing is equivalent to the System Testing Productivity of 5 requirements/(person*week).

Each requirement is analyzed by a System Engineer who is familiar with the context of a requirement. It is expected that the System Engineer performing the analysis is also supporting other efforts in the development cycle and that the System Test team is the sum of System Engineering resources supporting the test effort. To ensure that Contractor Testing begins on time, the size of the System Test team adjusts weekly in a uniform distribution to represent the urgency for System Engineers to complete System Test analysis for requirements they are assigned.

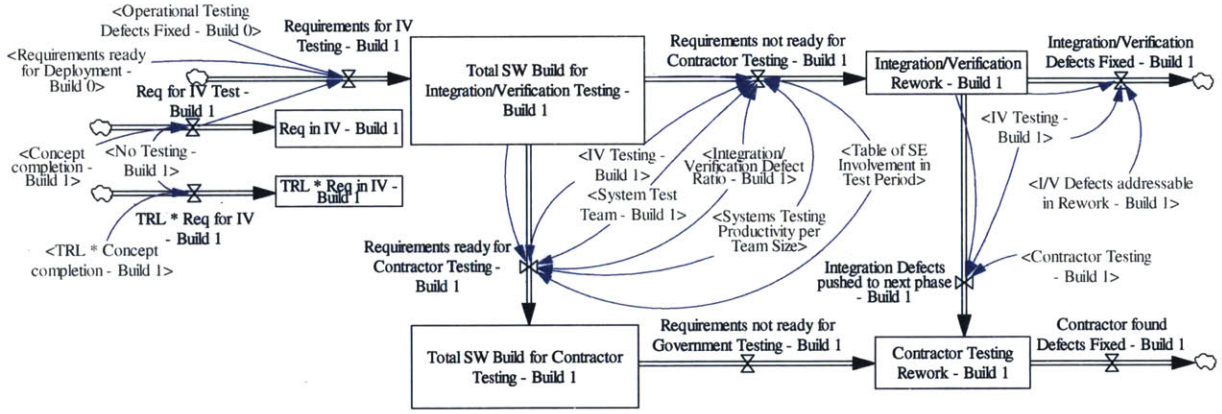


Figure 18 - System Dynamics representation of Integration/Verification Testing

While Integration/Verification Testing is ongoing, the defects are addressed by the System Defect team with a ratio of two System Engineers for every Software Engineer. The number of defects addressed is function of the System Defect team, System Defect team's base productivity (System Defect Productivity), and level of detail to mitigate risk of future follow-on defect (Productivity Rate over Time, Section 3.1.6.3.3). Once the Contractor Testing begins, all remaining defects are deferred to the Contractor Testing re-work phase.

3.1.4.2 Contractor Testing

For the Concept or Requirement Development effort that occurs during the Integration and Verification Testing phase, requirements are tested with the actual system in a representative operational

environment. This test effort provides operational insight on the stability and performance of the new requirements developed.

As shown in Figure 18 and Figure 19, the Contractor Testing effort begins when the Integration/Verification Testing is complete but, no earlier than 13 weeks, or ¼ year, after Integration/Verification Testing was scheduled to start. Once the test phase begins (indicated by the Contractor Testing variable), the System Test team begins assessing the set of new and legacy requirements via live or simulated testing. Based on the size and productivity of the team and System Test Defect rate, a number of requirements are passed for future testing while the defects are passed for Rework.

The rate of requirements being analyzed during Contractor Testing is equivalent to the System Testing Productivity of 5 requirements/(person*week). To ensure that Government Testing begins on time, the size of the System Test team adjusts weekly in a uniform distribution to represent the urgency for System Engineers to complete System Test analysis for requirements they are assigned.

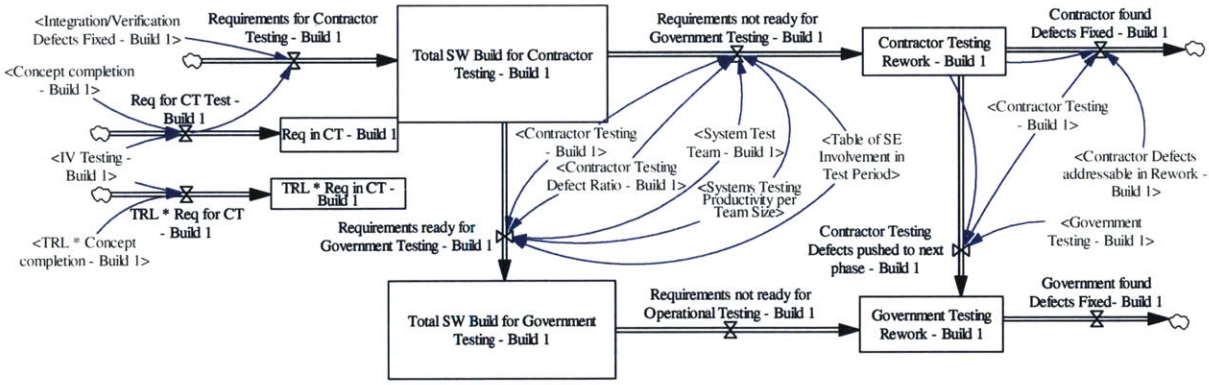


Figure 19 - System Dynamics representation of Contractor Testing

3.1.4.3 Government Testing

For the Concept or Requirement Development effort that occurs during the Contractor Testing phase, requirements are tested with the actual system in a representative operational environment. This test is led by the Government and provides operational insight on the stability and performance of the new requirements developed and legacy requirements.

As shown in Figure 19 and Figure 20, the Government Testing effort begins when the Contractor Testing is complete but, no earlier than 13 weeks after Contractor Testing was scheduled to start. Once the test phase begins (indicated by the Government Testing variable), the System Test team begin assessing the set of new and legacy requirements via live or simulated testing. Based on the size and productivity of the team and System Test Defect rate, a number of requirements are passed for future testing while the defects are passed for Rework.

Since Government Testing is executed by the customer, the involvement of the defense contractor, system developer, during this test effort decreases. Therefore, the rate of requirement analysis must increase by a factor, denoted by "System Engineering Involvement in Test Period", see Section 3.1.6.3.4.

To ensure that Operational Testing begins on time, the size of the System Test team adjusts weekly in a uniform distribution to represent the urgency for System Engineers to complete System Test analysis for requirements they are assigned.

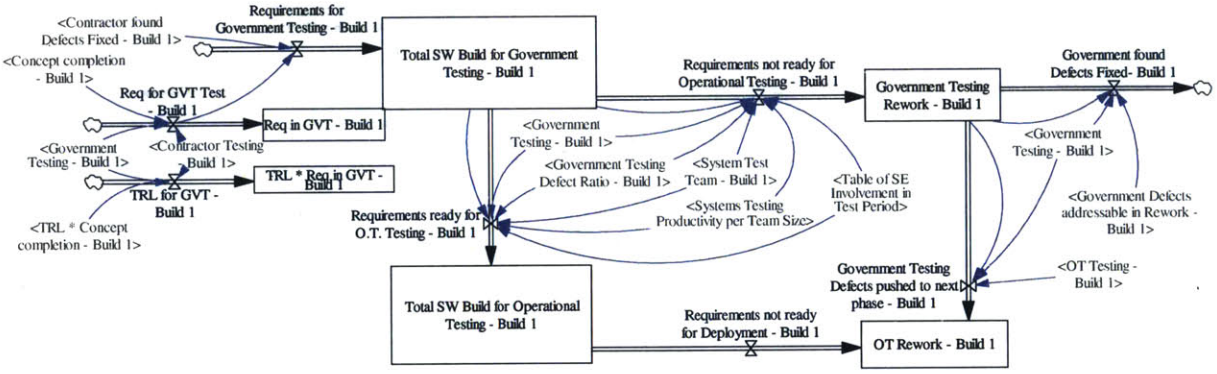


Figure 20 - System Dynamics representation of Government Testing

While Government Testing is ongoing, the defects are addressed by the System Defect team with a ratio of two System Engineers for every Software Engineer. The number of defects addressed is a function of the System Defect team, System Defect team's base productivity (System Defect Productivity), and level of detail to mitigate risk of future follow-on defect (Productivity Rate over Time, Section 3.1.6.3.3). Once the Operational Testing begins, all remaining defects are deferred to the Operational Testing re-work phase.

3.1.4.4 Operational Testing

Since by this phase all new functionality has been developed, integrated in the main build, and tested, the final build is tested with the actual system in a threat representative operational environment. This test is led by the Government and provides operational insight on the performance of the new capabilities developed.

As shown in Figure 20 and Figure 21, the Operational Testing effort begins when the Government Testing is complete but, no earlier than 13 weeks after Government Testing was scheduled to start. Once the test phase begins (indicated by the OT Testing variable), the System Test team begin assessing the performance of the system via live testing. Based on the size and productivity of the team and System Test Defect rate, a number of requirements pass the final testing while the defects are passed for Rework.

Since Operational Testing is the second test event executed by the customer, the involvement of the defense contractor, system developer, during this test effort decreases even more. Therefore, the rate of requirement analysis must increase by a factor, denoted by "System Engineering Involvement in Test Period", see Section 3.1.6.3.4.

To ensure that Operational Testing begins on time, the size of the System Test team adjusts weekly in a uniform distribution to represent the urgency for System Engineers to complete System Test analysis for requirements they are assigned.

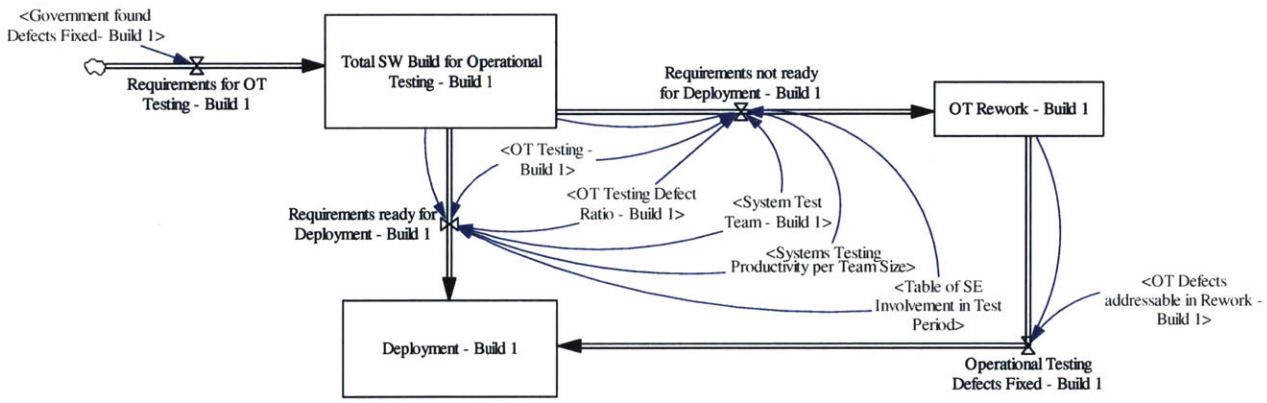


Figure 21 - System Dynamics representation of Operational Testing

While Operational Testing is ongoing, the defects are addressed by the System Defect team with a ratio of two System Engineers for every Software Engineer. The number of defects addressed is a function of the System Defect team, System Defect team's base productivity (System Defect Productivity), and level of detail to mitigate risk of future follow-on defect (Productivity Rate over Time, Section 3.1.6.3.3).

Once Operational Testing and Operational Testing Rework are complete, the build is considered mission ready and is deployed.

3.1.5 Human Resourcing

To control the cost element for comparing Waterfall and Agile development approaches, both models have 50 System Engineers and 15 Software Engineers dedicated to support the parallel build development effort. During each time step, System and Software Engineers are assigned task(s) based on need and priority. It is possible that there may be periods of time where the number of engineers exceeds the workload. However, it is assumed that the team would remain intact to preserve knowledge and be available when there is increased flux of engineers needed in the following time step. During an idle period, the additional engineers would support the planning between cycles or work on deferred tasks not directly related to ongoing development efforts.

The need for each Build Development task is determined by the number of engineering resources needed to complete a Concept or Requirement work unit. The need for System Testing tasks is determined by the average number of engineering resources to complete a phase within the remaining time period. It is assumed that engineers are supporting multiple tasks given experience and knowledge, therefore, the number of engineers supporting a phase is based on the number of hours spent working on sub-tasks within each phase.

The priority of tasks is based the build priority and then respective task priority. Since there are 2 builds being developed at any given time, the priority in assigning engineers to tasks is shown in Figure 22.

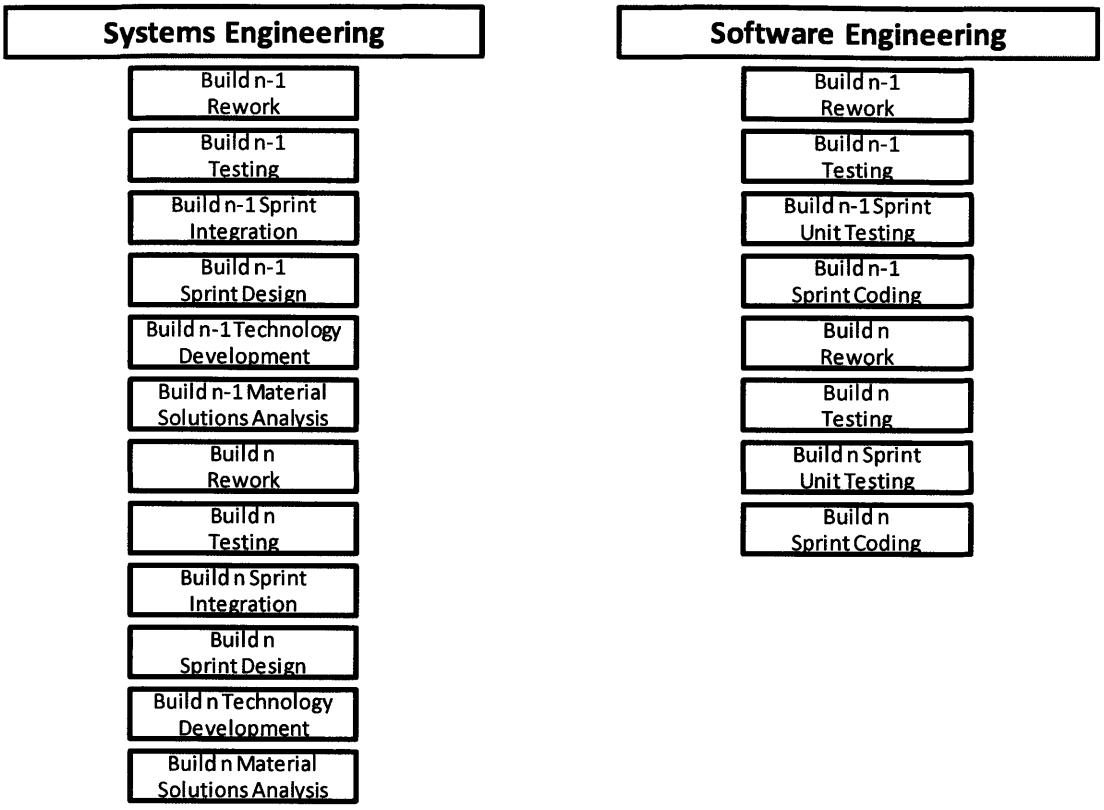


Figure 22 - Priority for allocating System and Software Engineers

3.1.6 Table Constant

The values used for the following constants do not reflect the metrics of any program of record. They were manufactured at the author's discretion in attempt to represent a reasonable development process. Although these metrics may not be actual metrics, the analysis is based on the tendencies/trends of each model.

An electronic copy of both models is available upon request for organizations to use their own proprietary information to compare approaches. If the electronic version of the model is not attached to this document, please email radian@alum.mit.edu to request a copy.

3.1.6.1 Productivity Rates

The Productivity Rates used for Build Development Tasks are shown in Table 2. Since a Concept is equivalent to 5 requirements, the Productivity rates for Concept Development and Waterfall Development tasks are proportional given size of scope.

Table 2 - Productivity metrics for System Design Tasks

	Ratio
Design Concept per Team Size	0.2
Design Requirement per Team Size	1
Code Concept per Team Size	0.2
Code Requirement per Team Size	1
Unit Test Concept per Team Size	0.2
Unit Test Requirement per Team Size	1
Integrate Concept per Team Size	0.2
Integrate Requirement per Team Size	1
M.S. Completion Productivity per Team Size	0.2
M.S. Logical Analysis Productivity per Team Size	0.2
M.S. Prototyping Productivity per Team Size	0.2
T.D. Completion Productivity per Team Size	0.2
T.D. Logic/Design Productivity per Team Size	0.2
T.D. Prototyping/Demo Productivity per Team Size	0.2

3.1.6.2 System Design Constants

3.1.6.2.1 Design Defect Rate

The Design Defect Rate for the System Design effort is based on the associated TRL of the Concept or the average TRL of the Requirements. The Design Defect Rate is lower for more mature technologies as can be seen by the curve in Figure 23. The slope of the curve revolves around TRL 5 and has a minimum of 10% and maximum of 100% error. The average Concepts and Requirements developed in each model ranged from TRL 5 to TRL 7.

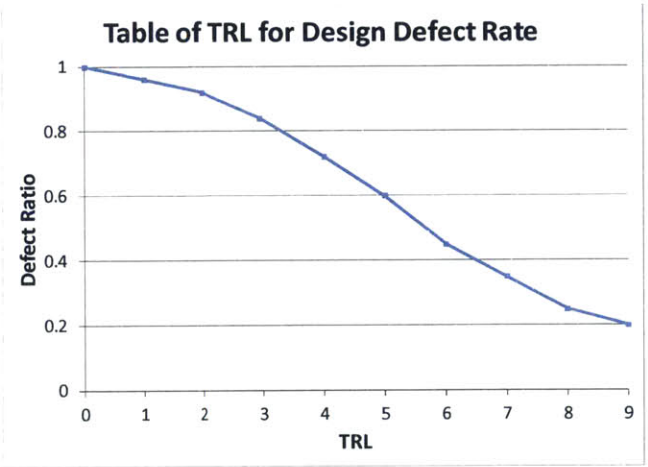


Figure 23 - Design Defect Rate Curve

3.1.6.2.2 Code Defect Rate

The Code Defect Rate for System Design effort is based on the associated TRL of the Concept or Requirements being coded. The Code Defect Rate Curve in Figure 24 is higher than the Design Defect Rate Curve due to Software Engineering's dependence on quality of the Design. It is assumed that the Design of lower TRL technologies is more abstract causing a higher probability of re-work.

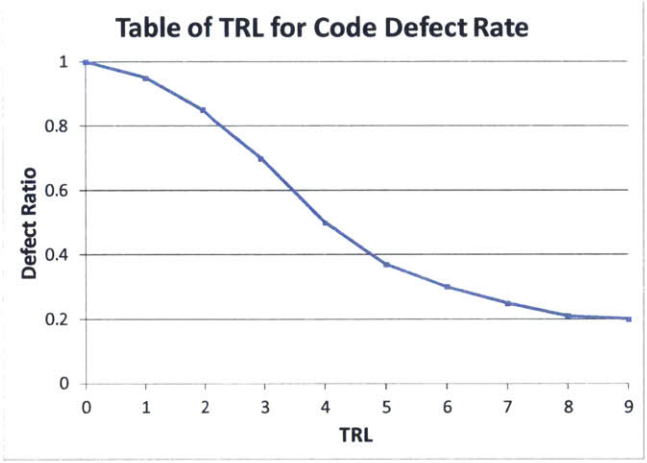


Figure 24 - Code Defect Rate Curve

3.1.6.2.3 Concept Defect Rate

The Concept Defect Rate for System Design effort captures defects that would not have been detected during Design or Code reviews but are identified during Integration. The rate is based on the associated TRL of the Concept or Requirements being integrated. The Code Defect Rate Curve in Figure 25 ranges from 50% and is limited to 16% error.

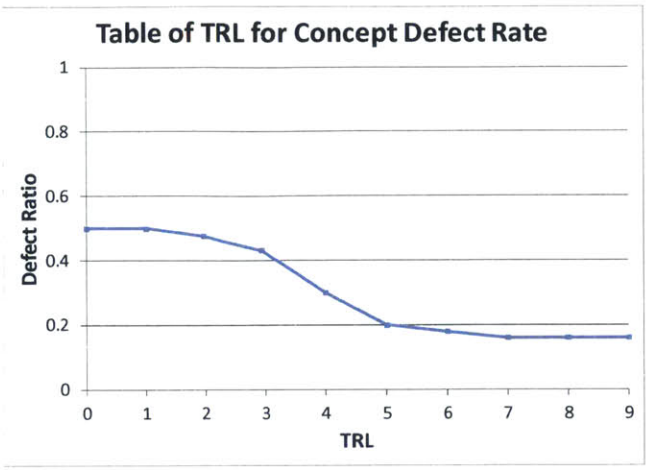


Figure 25 - Concept Defect Rate Curve

3.1.6.3 System Test Constants

3.1.6.3.1 System Test Defect Rate

The System Test Defect Rate for System Testing efforts capture the defects that are identified when testing of all requirements against an expected operational environment. The Defect Rate curve in Figure 26 is based off the average Technology Readiness Level of all requirements in the build: the Technology Readiness Level of new requirements are managed during System Testing and the Technology Readiness Level of legacy requirements are assumed to be Technology Readiness Level 9 since the legacy requirements have passed Operational testing.

Since the initial set of requirements in Build 0 have not undergone final testing, the Test Defect rate is based on Technology Readiness Level 8.

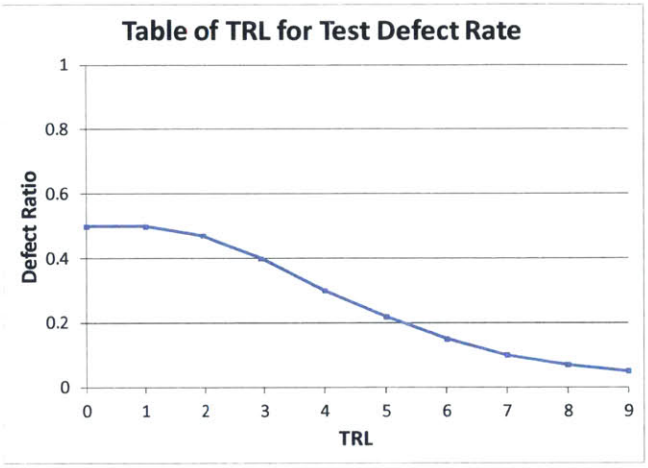


Figure 26 - System Test Defect Rate Curve

3.1.6.3.2 Defect Rate throughout Testing

The Defect Rate Factor in Figure 27 adjusts the defect rate for requirements that have undergone prior testing. Every time the requirement is tested, the probability of an error occurring in the current test phase is 50% less likely than the prior test phase. To manage the mixing of requirements, the System Testing model accounts for what test phase each requirement was introduced to capture the proper number of defects.

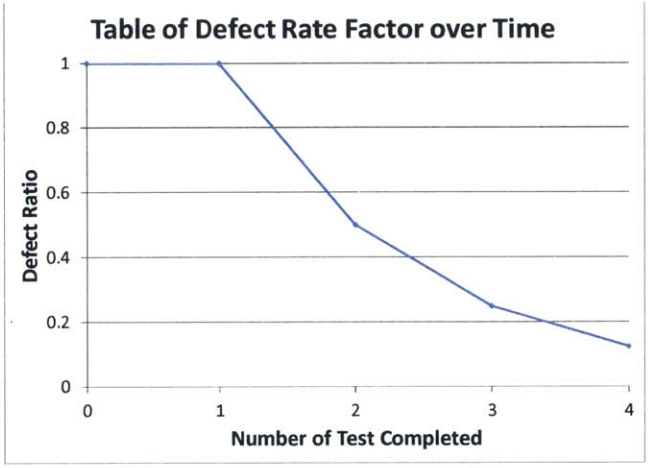


Figure 27 - Defect Rate Factor over Time

3.1.6.3.3 System Engineering Productivity through Defect Resolution

During each subsequent phase of testing, the Government and Contractor become more risk-adverse in accepting rework due to lack of testing. To offset risk of future defects, extra time is spent to review

and test rework. Therefore, the defect resolution productivity decreases to account for extra review and testing. The "Productivity Rate over Time" curve in Figure 28 illustrates that the productivity in addressing defects uncovered during System Testing decreases linearly by 25% from one phase to another.

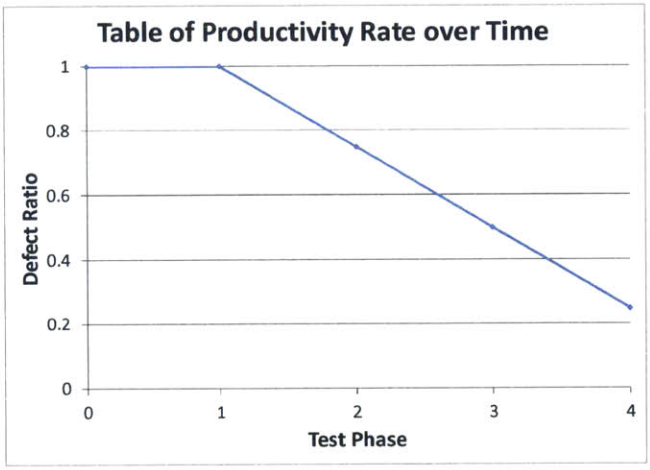


Figure 28 - Productivity Rate over Time

3.1.6.3.4 System Engineering Involvement in Test Period

During Integration/Verification and Contractor Testing, the Contractor's System Engineering team is responsible for leading System Testing and analyzing each system requirement. When Government and Operational Testing occur, the Government takes more lead of the effort and the Contractor is less involved with the testing and analysis. Therefore, the productivity to analyze each requirement increases during later test phases.

To capture the decreased involvement of the Contractor System Test team, the curve in Figure 29 is used to increase the productivity of the System Test team during Government and Operational Testing denoted as Test Phase 3 and 4, respectively.

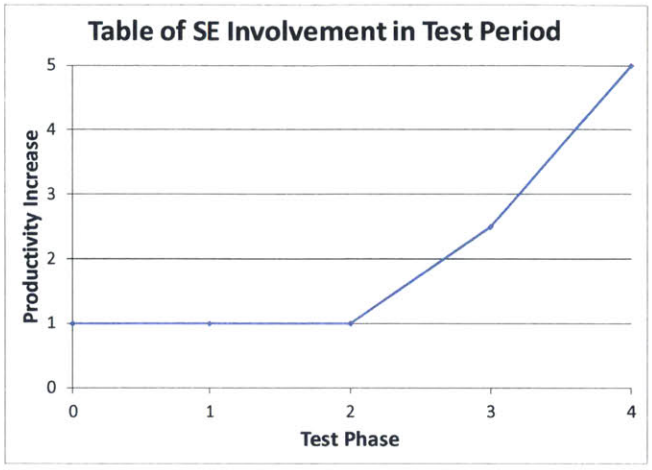


Figure 29 - System Engineering Involvement in Test Period

4 Analysis of Waterfall and Agile development Models

This chapter compares the simulation of parallel development models using Waterfall and Agile development process. As discussed in Section 3.1.3, the difference between both models is the application of Waterfall and Agile development during System Design. Waterfall development model manages the design and implementation efforts separately while the Agile development model allows parallel design and implementation.

4.1 Overview of Waterfall and Agile Model results

Both models replicate the Build Development and System Testing cycles for Build 0 to Build 3 as shown in Figure 8. Build 0 begins with 1,500 requirements for System Testing. The number, type, and frequency of concepts to be kicked-off during Builds 1 to Build 3 are shown in Table 3. The number of Engineers supporting the parallel development during all build activities is shown in Table 4.

Table 3 - Technologies considered for Build Development

Type of Concepts kicked-off	Concepts kicked-off	Weeks between kick-off
Materiel Solution Analysis	5	10
Technology Development	10	5
Engineering & Manufacturing Development	15	4

Table 4 - Size of Development Team

Type of Engineer	Number of Engineers
System Engineers	50
Software Engineers	15

4.1.1 Requirements Completion

While Build Development is ongoing, the System Design team designs, codes, unit tests, and integrates requirements for future System Testing. Completed requirements in each build are added to the associated System Testing phase based on the time completed, as shown in Table 5.

Table 5 – Associated System Testing phase for completed requirements

	Build 1	Build 2	Build 3
Integration/ Verification	Week 0 - 51	Week 52 - 103	Week 104 – 155
Contractor Testing	Week 52 - 64	Week 104 - 116	Week 156 - 168
Government Testing	Week 65 - 78	Week 117 - 130	Week 169 - 182

Table 6 presents the number of new requirements added to respective System Testing phase. Consistent with software development literature, the Agile development model provides earlier delivery and introduces more requirements in each build than Waterfall model.

Table 6 – Number of Requirements introduced during each Build's Testing phase

Waterfall	Build 1	Build 2	Build 3
Integration/ Verification	65	65	55
Contractor Testing	30	30	0
Government Testing	20	10	35
Total	115	105	90

Agile	Build 1	Build 2	Build 3
Integration/ Verification	105.9	88.9	82.0
Contractor Testing	25.0	26.6	17.4
Government Testing	9.3	12.9	21.2
Total	140.2	128.4	120.6

4.1.2 Types of Technologies Developed

While Build Development is ongoing, the Materiel Solution Analysis and Technology Development teams mature new technologies for implementation. These new technologies are associated with lower Technology Readiness Levels as compared to existing technologies that do not require technology maturation.

The average Technology Readiness Level for requirements added to each build for both models is shown in Table 7. Although the Agile development model includes earlier requirements completion and integrates more requirements than the Waterfall development model, the Waterfall development model included a higher percentage of newer technologies across each build.

Table 7 – Average TRL for Requirements added to each Build

	Build 1	Build 2	Build 3	Average
Waterfall Model	6.53	6.57	6.72	6.61
Agile Model	6.76	6.88	6.92	6.85

4.2 Analysis of Waterfall and Agile model development tendencies

This section includes the performance tendencies of both parallel development models discussed in Section 3. The Agile and Waterfall development models are used to compare the effect of the System Design processes on resource dependencies.

4.2.1 Requirements Development in each Build

Based on the requirements identified in Table 6, the following sub-sections discuss the development of requirements in each build.

4.2.1.1 Requirements Development in Build 1

Build 1 development starts at week 0 and ends at week 78. During development, the number of requirements completed is higher and the rate starts earlier in Agile development than Waterfall development, as shown in Figure 30. The requirements completed at week 52, week 65, and week 78 represent the total number of new requirements being tested during Integration & Verification, Contractor, and Government Testing, respectively.

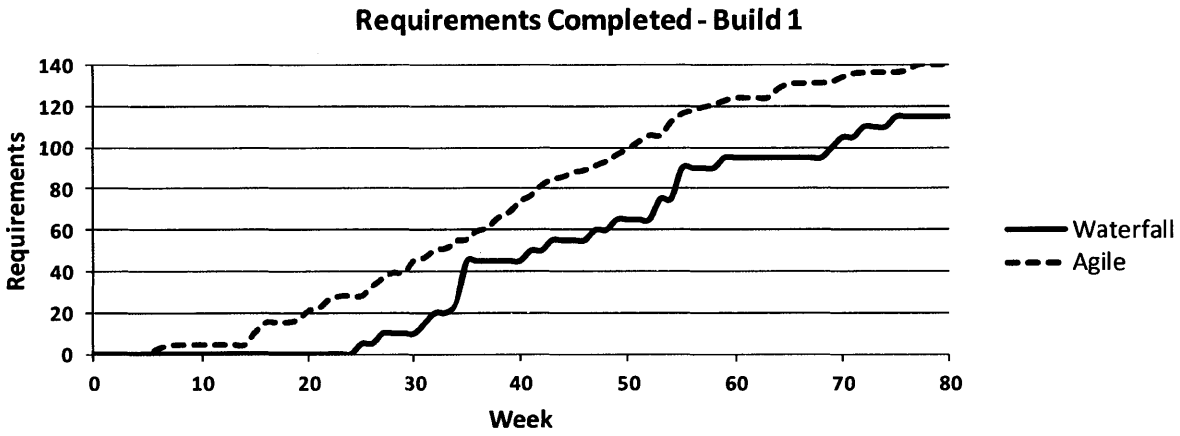


Figure 30 - Requirements Completed in Build 1

The cause for the number of requirements completed being higher during Agile development is based on the number of System and Software Engineers supporting the System Design effort. The number of engineers supporting a phase is based on the number of available engineers and the number of needed engineers. For System Design in Build 1, only the System Testing and Defect efforts supersede the

System Design effort. Figure 31 and Figure 32 illustrates the number of System and Software Engineers supporting the System Design effort.

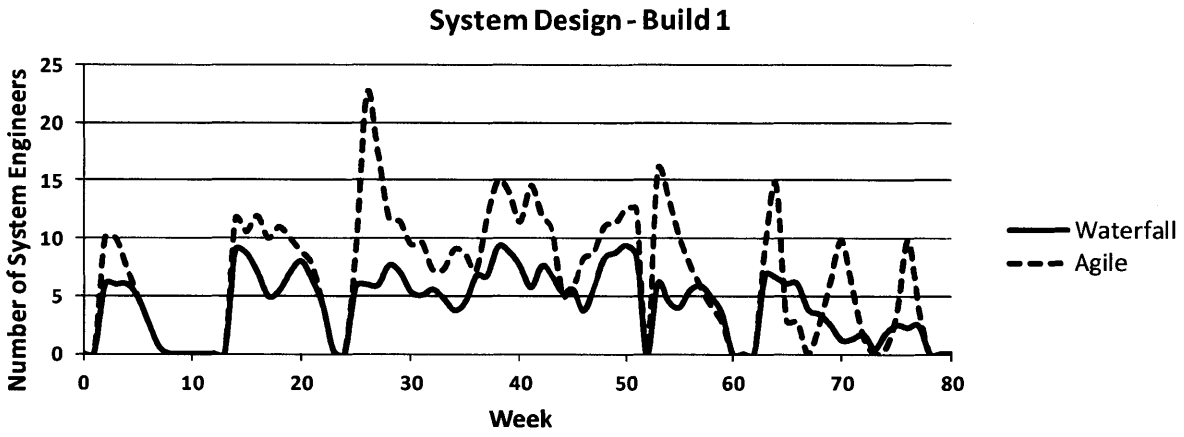


Figure 31 - System Engineers supporting System Design in Build 1

The number of System Engineers during Agile development is higher in Agile development due to parallel development efforts of Detailed Design and Integration work, as shown in Figure 31. By decomposing a concept into increments, the design team can develop parts of many different concepts at a time. System Engineers supporting Waterfall development must wait to perform Integration until all of the Design is completed, making the overall engineering need lower.

The number of Software Engineers starts earlier and is much higher in Agile development due to the early release of requirement design, as shown in Figure 32.

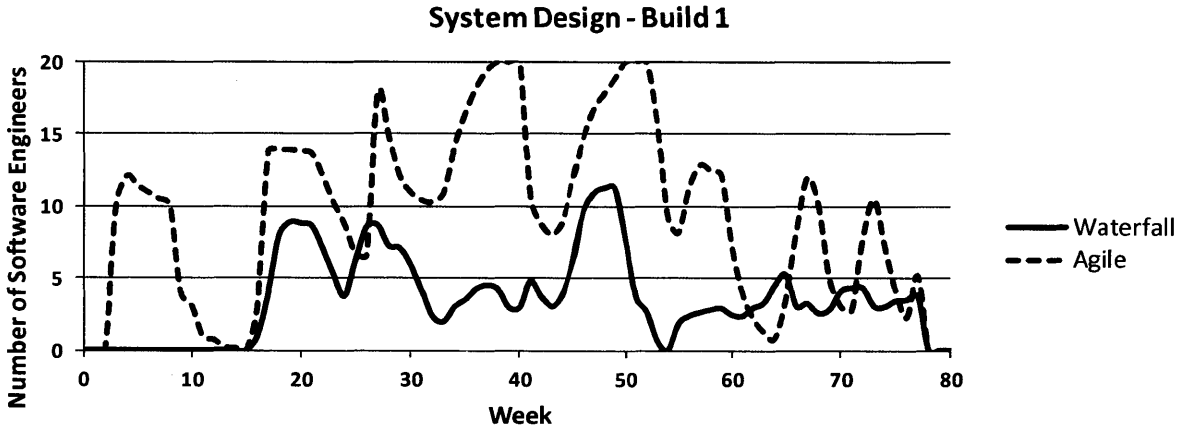


Figure 32 - Software Engineers supporting System Design in Build 1

Although the number of System and Software Engineers supporting System Design is higher in Agile development, Figure 33 illustrates that the number of requirements in progress is higher during Waterfall development. The reason for this contrast is due to the Agile development approach for taking on requirements that the team can manage. Every number requirement in progress represents the distribution of work across System and Software Engineering efforts as oppose to Waterfall development that bottlenecks requirements work between Design and Code efforts.

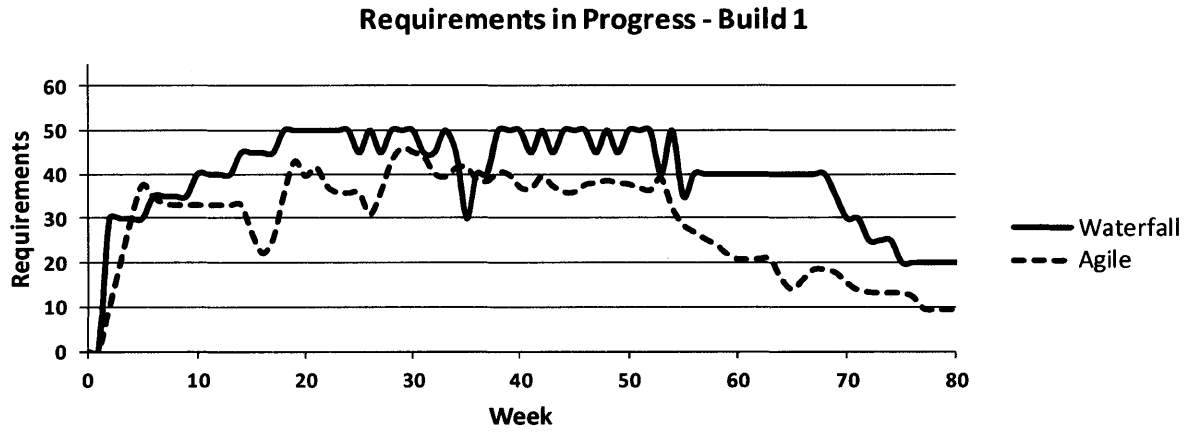


Figure 33 - Requirements in Progress in Build 1

4.2.1.2 Requirements Development in Build 2

Build 2 development starts at week 52 and ends at week 130. Similar to Build 1 development, the number of requirements completed is higher and the rate starts earlier in Agile development than Waterfall development, as shown in Figure 34. The requirements completed at week 104, week 117, and week 130 represent the total number of new requirements being tested during Integration & Verification, Contractor, and Government Testing, respectively.

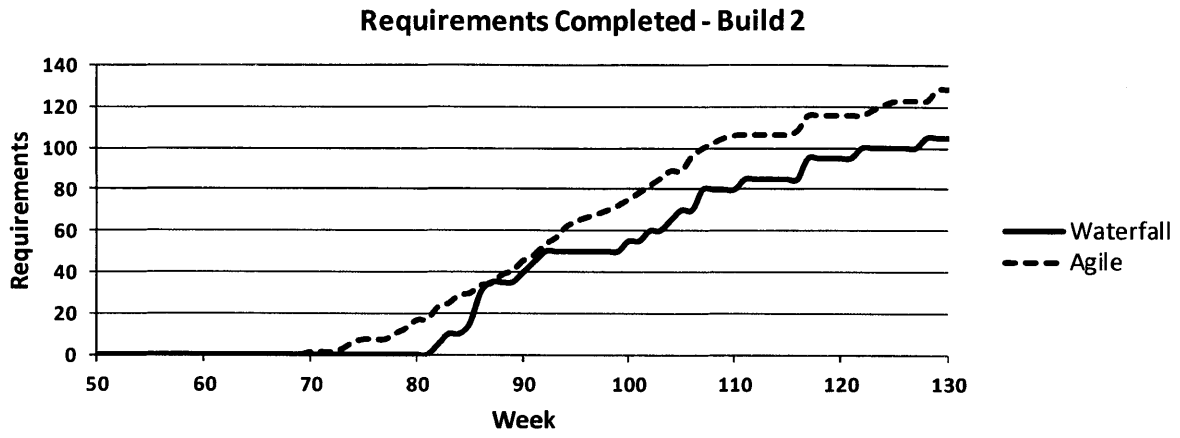


Figure 34 - Requirements Completed in Build 2

The cause for the number of requirements completed being higher during Agile development is based on the number of System and Software Engineers supporting the System Design effort. For System Design in Build 2, all Build 1 efforts and Build 2 System Testing and Defect efforts supersede the Build 2 System Design effort. Unlike Figure 31 and Figure 32 that have a larger upfront System and Software Engineering team, Figure 35 and Figure 36 illustrate the number of System and Software Engineers supporting the System Design effort.

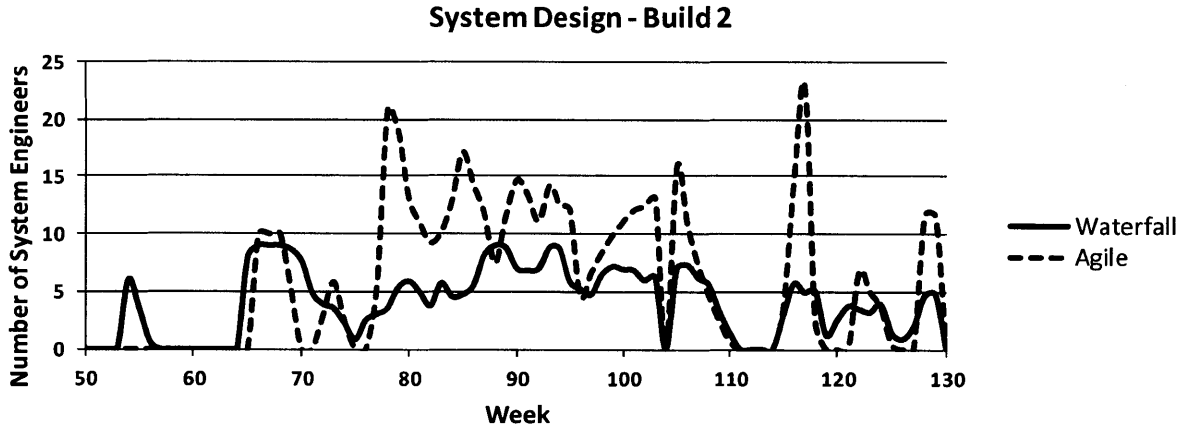


Figure 35 - System Engineers supporting System Design in Build 2

Although no System Engineers are supporting early System Design during Agile development, the number of System and Software Engineers dramatically increases once the Build 1 – Build Development efforts is complete. As discussed in Section 0, the size of the engineering team can increase quickly due

to the parallel development approach that generates immediately starts generating work across all System Design development tasks.

Although the System Engineering team starts later during Agile development, the Software Engineering team starts slightly earlier and remains higher than Waterfall development, as shown in Figure 36.

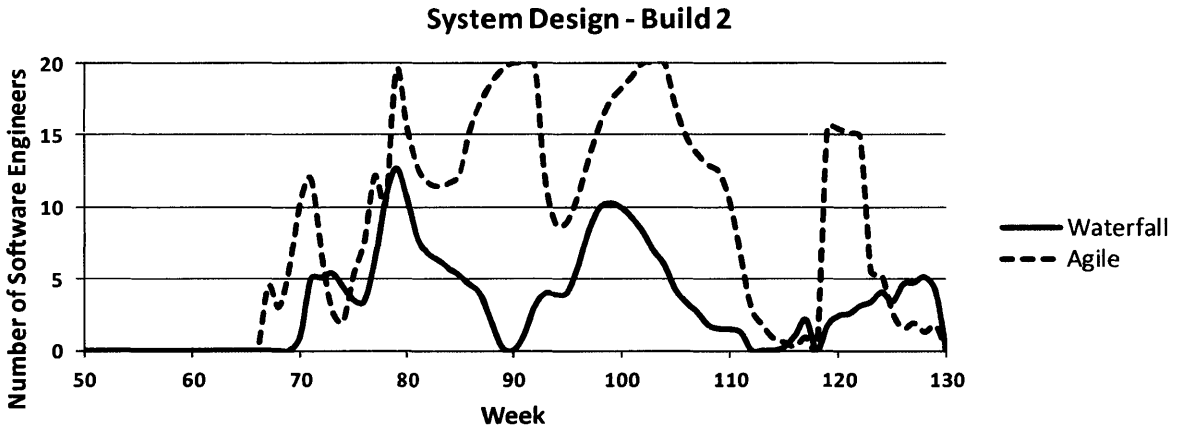


Figure 36 - Software Engineers supporting System Design in Build 2

Since the Agile System Engineering team does start immediately, the number of requirements in progress does not increase until week 66, as shown in Figure 37. Unlike Build 1, there is a period in time where number of requirements in progress is higher for Agile development. This increase is due to the increase in System and Software Engineers supporting the System Design effort, as shown in Appendix 7.4 and 7.5.

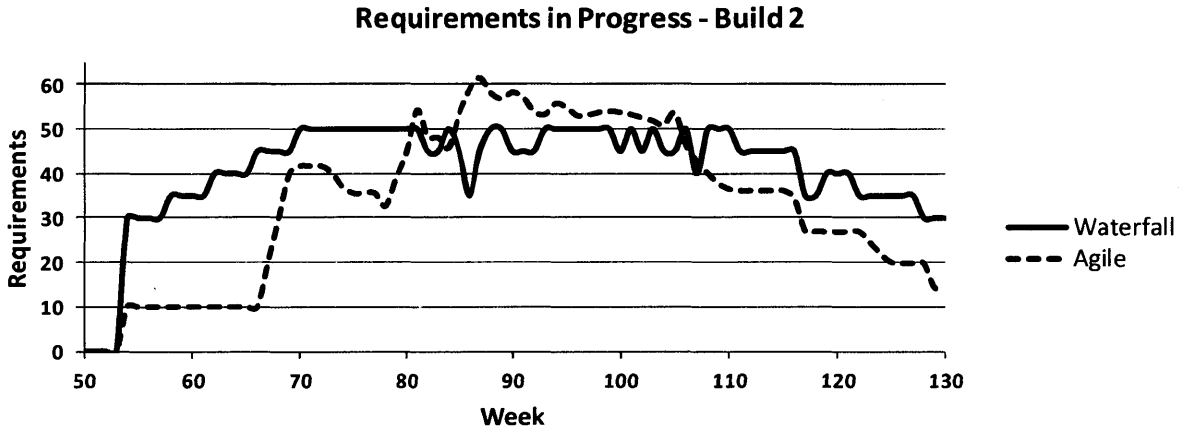


Figure 37 - Requirements in Progress in Build 2

4.2.1.3 Requirements Development in Build 3

Build 3 development starts at week 104 and ends at week 182. Similar to Build 2 development, the number of requirements completed is higher and the rate starts earlier in Agile development than Waterfall development, as shown in Figure 38. The requirements completed at week 156, week 169, and week 182 represent the total number of new requirements being tested during Integration & Verification, Contractor, and Government Testing, respectively.

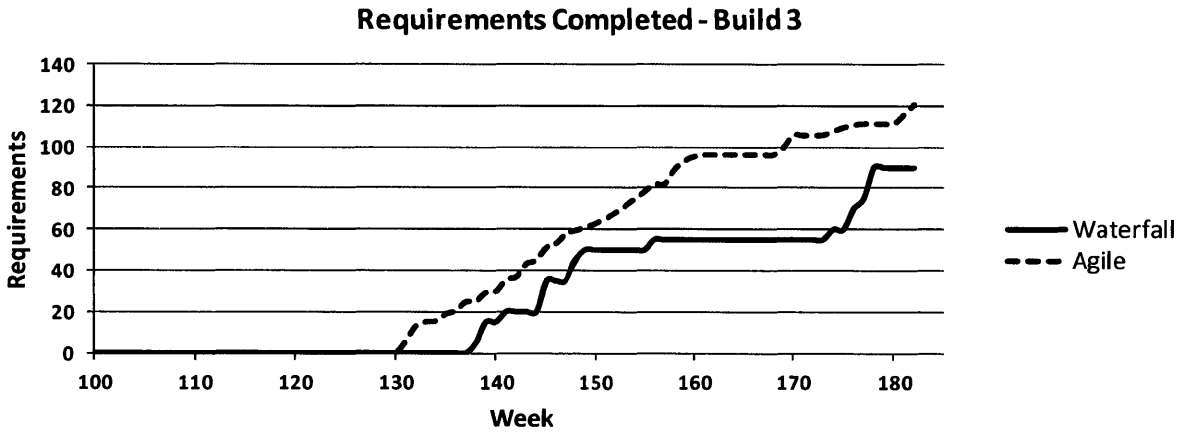


Figure 38 –Requirements Completed in Build 3

The cause for the number of requirements completed being higher during Agile development is based on the number of System and Software Engineers supporting the System Design effort. For System Design in Build 3, all Build 2 efforts and Build 3 System Testing and Defect efforts supersede the Build 3 System Design effort. Unlike Figure 31 and Figure 32 that have a larger upfront System and Software Engineering team, Figure 39 and Figure 39 illustrate the number of System and Software Engineers supporting the System Design effort, which are similar to Build 2 Figure 35 and Figure 36.

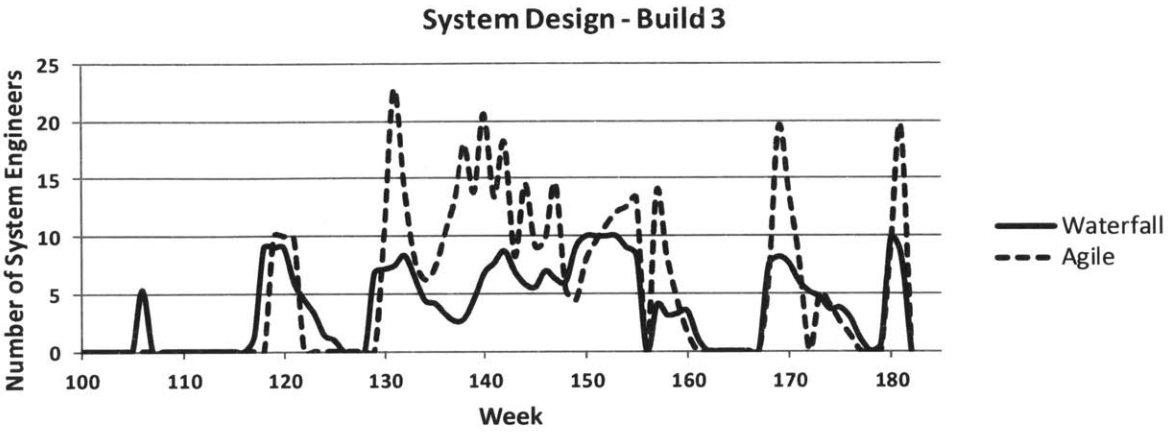


Figure 39 - System Engineers supporting System Design in Build 3

Although no System Engineers are supporting early System Design during Agile development, the number of System and Software Engineers dramatically increases once the Build 1 – Build Development efforts is complete. As discussed in Section 0, the size of the engineering team can increase quickly due to the parallel development approach that generates immediately starts generating work across all System Design development tasks.

Although the System Engineering team starts later during Agile development, the Software Engineering team starts slightly earlier and remains higher than Waterfall development, as shown in Figure 39.

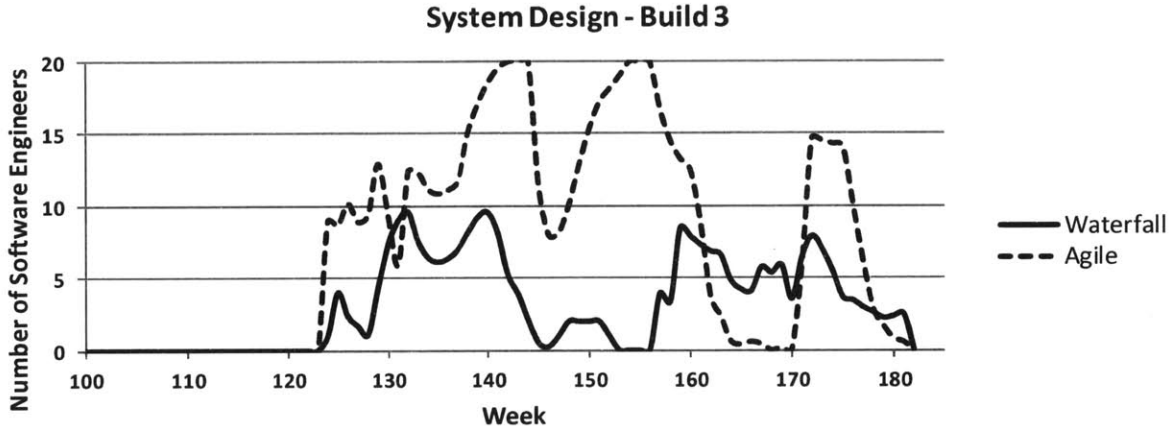


Figure 40 - Software Engineers supporting System Design in Build 3

Since the Agile System Engineering team does start immediately, the number of requirements in progress does not increase until week 118, as shown in Figure 37. Contrary to Build 1 but similar to Build 2, there is a period in time where number of requirements in progress is higher for Agile development. This increase is due to the increase in System and Software Engineers supporting the System Design effort, as shown in Appendix 7.4 and 7.5.

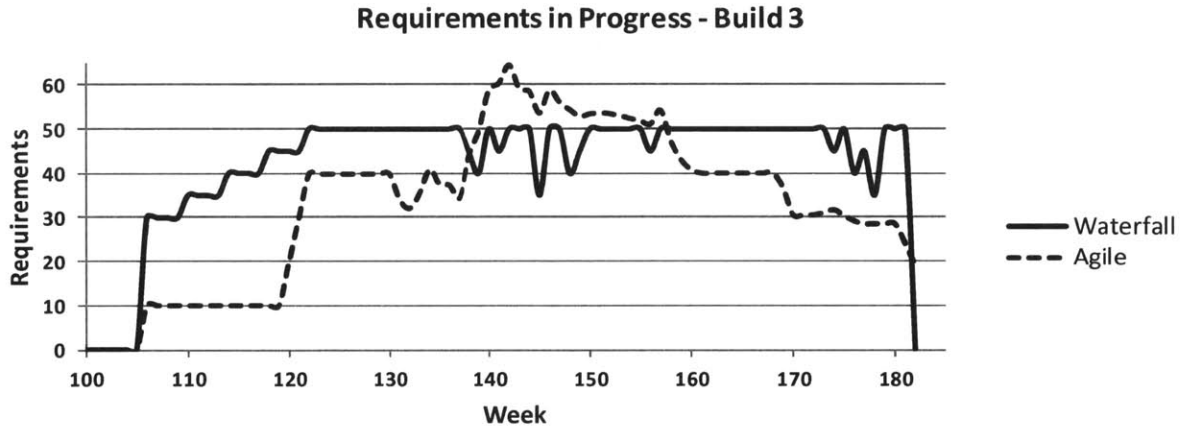


Figure 41 - Requirements in Progress in Build 3

4.2.2 Types of Technologies Developed

Although the quantity and rate of requirements completed in each build is higher during Agile development, Waterfall development introduces newer technologies in each build, as presented in Table 7. As oppose to existing technologies that are shared among systems, new technologies must be matured before being designed and implemented in a build. The maturation of technologies occurs in Materiel Solution Analysis and Technology Development efforts. However, these efforts are lower priority than System Design and System Testing efforts since the value is less tangible. The contrast in level of technologies implemented stems from the difference in allocation of engineers.

4.2.2.1 Technology Development

During Build Development, concepts for maturing new technologies in Technology Development are kicked-off at the fidelity shown in Table 3. When scope is added to the Technology Development effort, System Engineering resources are requested to complete the task. However, the priority of Technology Development is lower than the priority of the previous build and the priority of System Design and Testing.

Figure 42 compares the number of System Engineers supporting each build's Technology Development effort during Waterfall and Agile development. Unlike Agile development, there are periods between Waterfall System Design tasks when System Engineers are idle. Due to reduced need of System Engineering resources during System Design, System Engineers responsible for System Design tasks spend their extra time performing side tasks that include analysis and trade studies that lead to innovation.

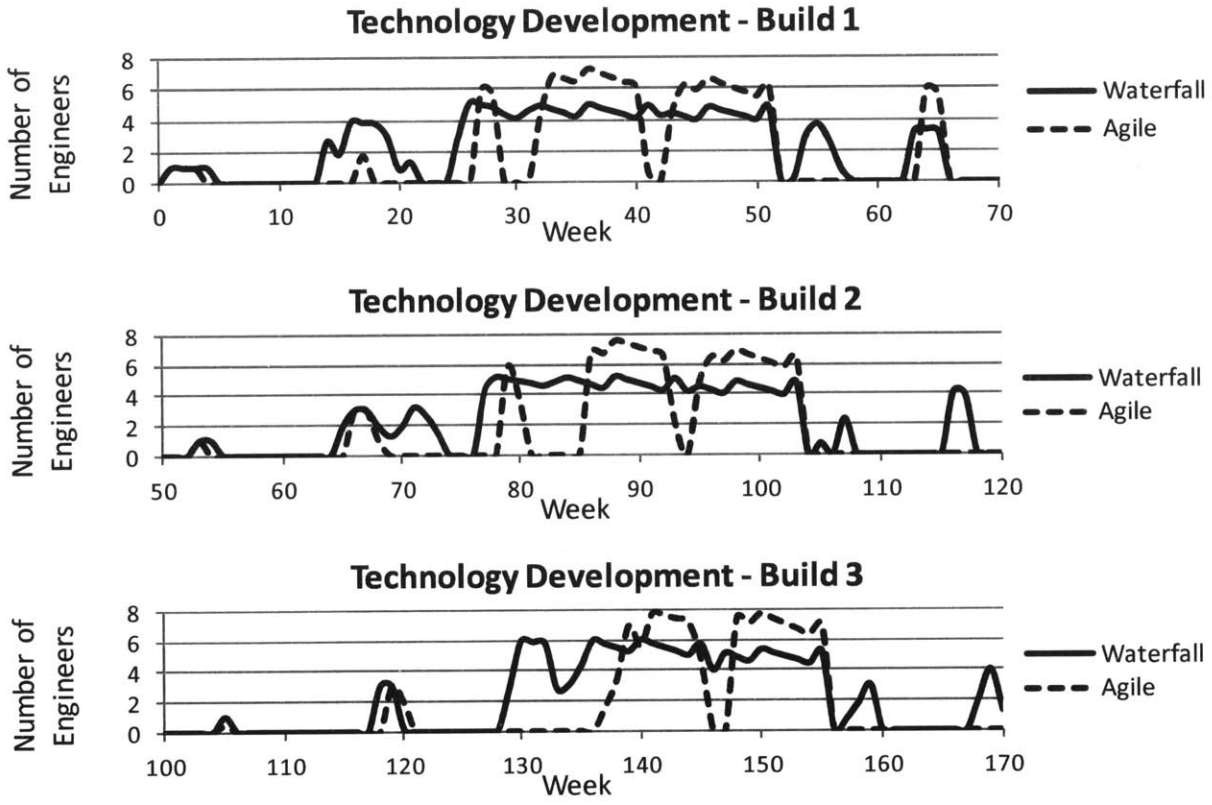


Figure 42 - Engineers supporting Technology Development

In all three builds, Waterfall development exhibits more upfront time spent developing new technologies. By spending more time up front, new technologies are matured in time to be designed and implemented in the current build. The extra time spent during Technology Development, shown in Figure 42, translates to the higher number of new technologies added to a build, shown in Table 7.

4.2.2.2 Materiel Solution Analysis

During Build Development, concepts for maturing new technologies in Materiel Solution Analysis are kicked-off at the fidelity shown in Table 3. When scope is added to the Materiel Solution Analysis effort, System Engineering resources are requested to complete the task. However, the priority of Materiel Solution Analysis is lower than the priority of all tasks in the previous and current build.

Figure 43 compares the number of System Engineers supporting each build's Materiel Solution Analysis effort during Waterfall and Agile development. During Build 1 where there is only 1 build in development, the number of System Engineers supporting Materiel Solution Analysis is relatively similar in both models. During Build 2 and Build 3 where there is parallel development, the level of early Materiel Solution Analysis support decreases more so in Agile development than Waterfall development. Once the prior build's development is complete, the number of engineers that can support the current build increases, leading to higher resource allocation of all Build Development tasks.

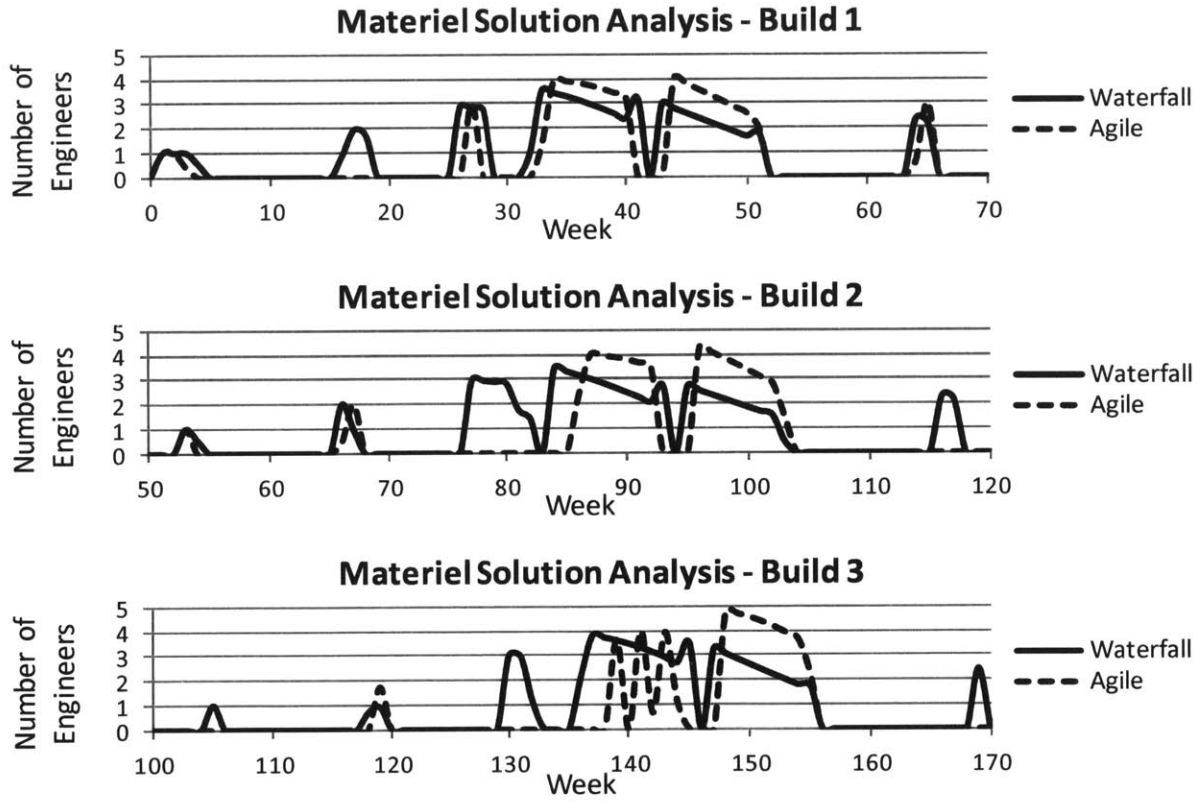


Figure 43 - Engineers supporting Materiel Solution Analysis

4.3 Discussion

The following sections summarize the analysis in Section 4.2 with respect to the three Research Questions posed in Section 1.2.

4.3.1 Reducing conflict between cost, schedule, and capability constraints

Defense Contractors are required to implement Earned Value Management (EVM) during development of most defense programs. Each task has an associated set of expected costs in labor hours to perform the work within an expected time period. During each development cycle, the scope of individual tasks may change during periodic customer reviews due to changes in development performance and warfighter needs. If there were to be shifts in funding and/or schedule constraints, it is expected that the number and/or scope of tasks would be reduced. To compare models, the size of the engineering team and development release schedule are fixed to determine if capability slack is created.

Based on the analysis of both models, the Agile development model consistently developed and implemented more technologies under the fixed cost and schedule, as shown Table 8. The reason for an increase in technologies developed and implemented is attributed to how Agile development maximizes productivity across engineering disciplines through distributed parallel development. Given the short development cycle of each increment, switching to Agile development would provide the Government program office capability slack to manage competing cost, schedule and capability constraints.

Table 8 – Number of Technologies Adopted in a Build

	Build 1	Build 2	Build 3
Waterfall	23	21	18
Agile	28	25	24

4.3.2 Reducing Firefighting

Firefighting is a development state when early development of the next release is delayed due to resourcing needs to address defects detected late in the current release. To prevent firefighting during development of military systems, it is important to complete requirements earlier so that they are test more often, thereby reducing the probability of defects being detected later in the build.

Figure 44 and Figure 45 illustrate the number of Technologies introduced in each test phase of Waterfall and Agile development models. Waterfall development is negatively impacted due to competing

development resources during later builds leading to a later technology introduction. Since Agile development implements fractions of a technology at a rate proportional to available engineers, the ratio of technologies introduced in each test phase is visually consistent from build to build.

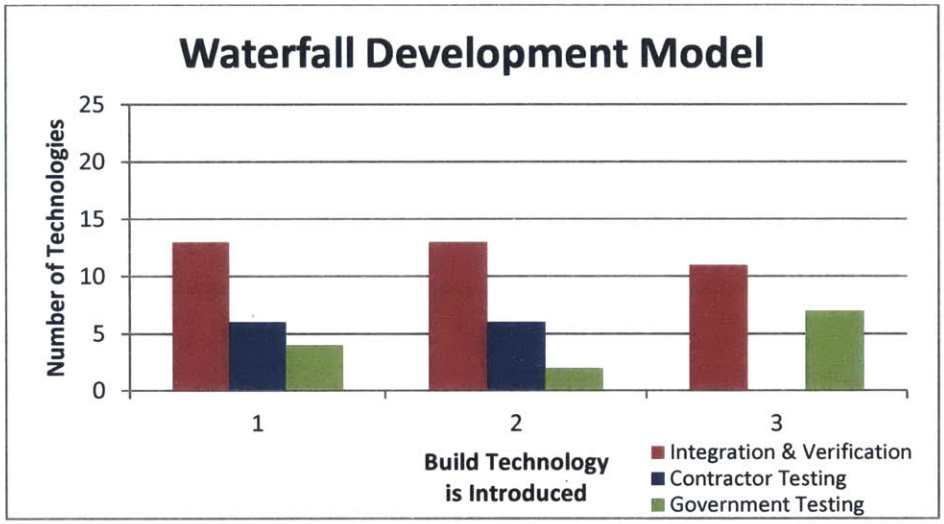


Figure 44 – Technologies introduced during each Build's test phase of Waterfall

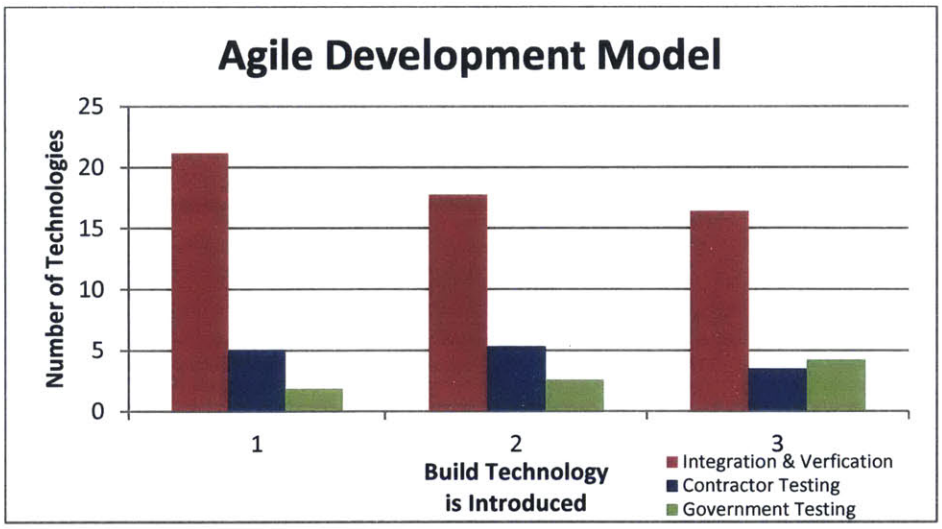


Figure 45 – Technologies introduced during each Build's Test Phase of Agile

Table 9 presents the number of requirements introduced in each build's test phase for Waterfall and Agile development. In proportion to the total number of requirements, Waterfall development tends to

introduce more requirements later in the test phase. While Agile development could have stopped development early yet introducing slightly more requirements in the build.

Table 9 - Requirements introduced in each Build's Test Phase

Waterfall	Build 1	Build 2	Build 3
Integration/ Verification	65	65	55
Contractor Testing	30	30	0
Government Testing	20	10	35
Total	115	105	90

Agile	Build 1	Build 2	Build 3
Integration/ Verification	105.9	88.9	82.0
Contractor Testing	25.0	26.6	17.4
Government Testing	9.3	12.9	21.2
Total	140.2	128.4	120.6

Given that the purpose of Government Testing is to determine capability and performance, similar to "beta testing", it is less desirable to introduce new capabilities so late. What makes Waterfall development more susceptible to firefighting is the average Technology Readiness Level of requirements introduced in each test phase, shown in Table 10.

Table 10 - TRL of Requirements introduced in each Test Phase of Build 1

	Waterfall	Agile
Integration/ Verification	6.88	6.93
Contractor Testing	6.32	6.37
Government Testing	5.72	5.92

Given the time delay and increased defect rates, it is expected that the number of new technologies adopted are introduced during later test phases. However, introducing new technologies late in the test phase increases the probability of defects being uncovered during the final test phases and post-deployment. Depending on severity, late defects may be deferred to the next build or addressed in an intermediate "clean-up" build.

4.3.3 Impact on Development Tasks

As discussed in the prior Discussion sections, switching to Agile development increases the number of requirements in a build and promotes early software delivery. However, these two benefits are

observed at the expense of maturing new technologies that support long-term growth, as shown in Table 7 and discussed in Section 4.1.2. Table 11 compares the estimate of new and existing technologies implemented based on the average Technology Readiness Level of each build.

Table 11 - Comparison of technologies developed based on average TRL

	Waterfall		Agile	
	New Technology	Existing Technology	New Technology	Existing Technology
Build 1	10.8	12.2	6.6	21.4
Build 2	9.0	12.0	3.2	22.5
Build 3	5.1	12.9	1.9	22.2

Although it is advertised that Waterfall development bottlenecking between phases is a limitation, the limitation indirectly promotes innovation through non-System Design time spent maturing a new technology. The time spent maturing a new technology is important since the newly developed technology provides the warfighter an operational advantage. In addition, the "new" technology can be shared with similar systems as an "existing" technology.

5 Conclusion

The United States Department of Defense has been plagued with failing programs that are over budget, late on schedule, and exhibit poor performance during testing. Once a program has cost, schedule, or capability issues, follow-on development efforts adopt the underlying issues only to reinforce poor performance. To address these issues that lead to firefighting, one option for improving the quality of the system is to introduce capabilities early in the development process so that intensive testing can reduce the probability that defects are uncovered later in the life-cycle. One potential option is to leverage Agile development methodologies.

Given the recognition Agile development has received in the commercial sector, there are a number of organizations promoting Agile development to be used in the United States Defense sector, (O'Connell, 2011) (Samios, 2012). Agile is an iterative incremental development style that focuses on needs and priorities. This development style has been linked to improved software quality, productivity and delivery as well as ability to adapt to changing requirements.

However, Agile may not be a silver bullet and applying this development style may cause unintended effects during the entire life-cycle. Therefore, two System Dynamics models were developed based on the Defense Acquisition Framework to compare Agile and Waterfall development approaches and determine: if Agile development reduces the conflict between cost, schedule, and capability constraints; if Agile development reduces firefighting; and will Agile development impact other development tasks.

Based on the analysis of simulations of each model, Agile did improve the dynamics of parallel development cycles by maximizing the productivity of the entire development team. Under the same System and Software Engineering team size and development release schedule, Agile development increases the quantity of requirements introduced within a development cycle. These requirements are introduced earlier in Agile than they are in Waterfall development leading to reduced Test Debt.

The strength of Agile development lies in the ability to manage and implement increments of a capability across multiple development cycles. By modularizing each capability, Agile development maximizes the Design, Code, Unit Test, and Integration Testing work load improving the team efficiency.

During analysis of impact on other development tasks, it was identified that Agile development consistently integrated capabilities based on existing technologies. Given that Agile development maximizes team work load, the Agile development effort starves Materiel Solution and Technology Development tasks leading to reduced maturation of new technologies to integrate within current development cycle. This tendency is the effect of tactical management policies that are driven to increase the number of capabilities versus the maturation of technologies.

To ensure success of Agile development, Government project/program office and defense contractors/suppliers should increase the fidelity and level of interaction to align strategic goals. To ensure sustainable performance, Government policies should promote flexible capability development time lines, limit introduction of new capabilities in later test phases, and invest in separate technology maturation efforts that support strategic needs.

6 Future Work

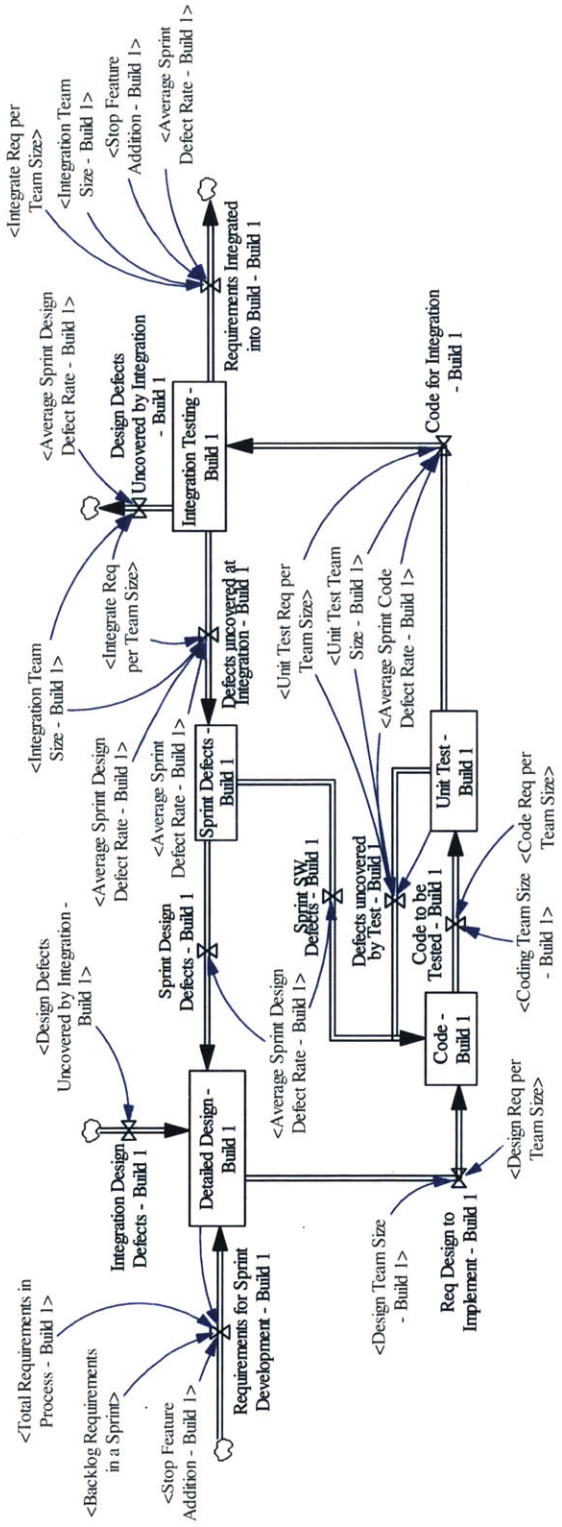
This work compliments existing research that compares Waterfall and Agile software development and provides the foundation for future research on the impact of policies during parallel development of military systems. The models are based on the actual development process but do not use historical data to determine the parameters or capture the human element during development. To improve the accurateness and holism of both models, future research is needed to calibrate both models and include additional development factors that legitimize each model's prediction.

The intent of this thesis is to compare development tendencies between Waterfall and Agile development. Since each type of military system varies in complexity, both models should be calibrated using historical development metrics from similar types of military systems e.g. communication systems, sensor systems, engagement systems, aircraft, ships, etc. By leveraging historical data, both models could be used to predict the impacts of different policies.

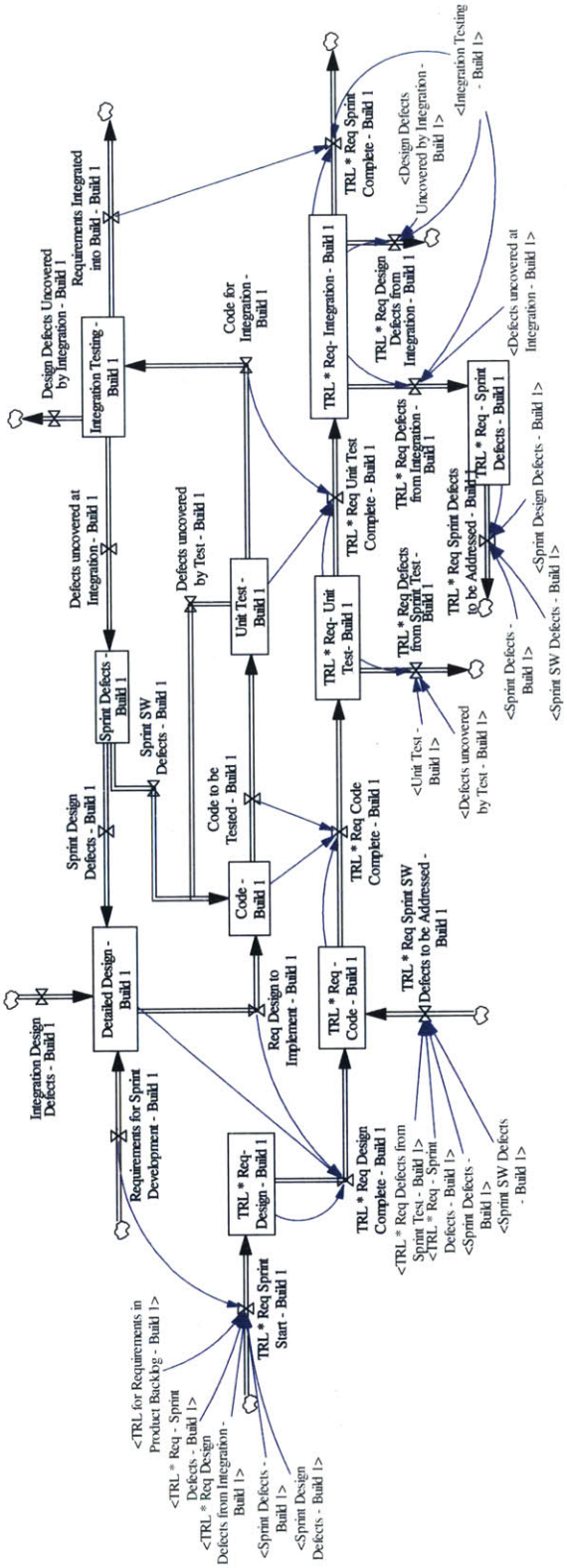
In addition to model calibration, future researchers can extend both models to capture human and organizational factors that influence the execution of the system's development. Each model assumes a constant size and team experience, but does not account for shifts in funding or staffing that can occur during development; individual engineering experience of unique domain knowledge; or the complexity in managing a portfolio of concepts in development. These factors provide additional layers of resource dependencies for future modeling and analysis.

7 Appendix

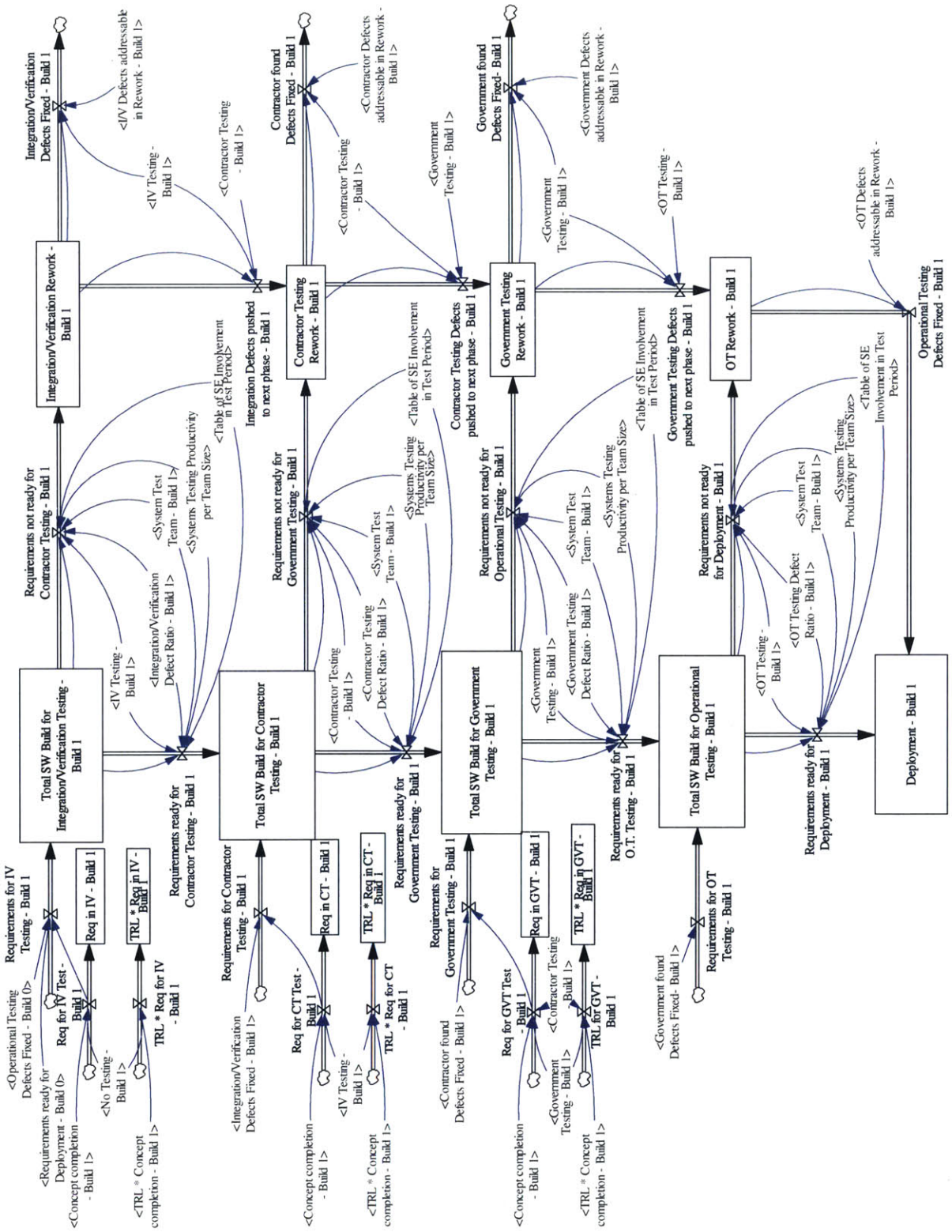
7.1 System Design during Agile Development



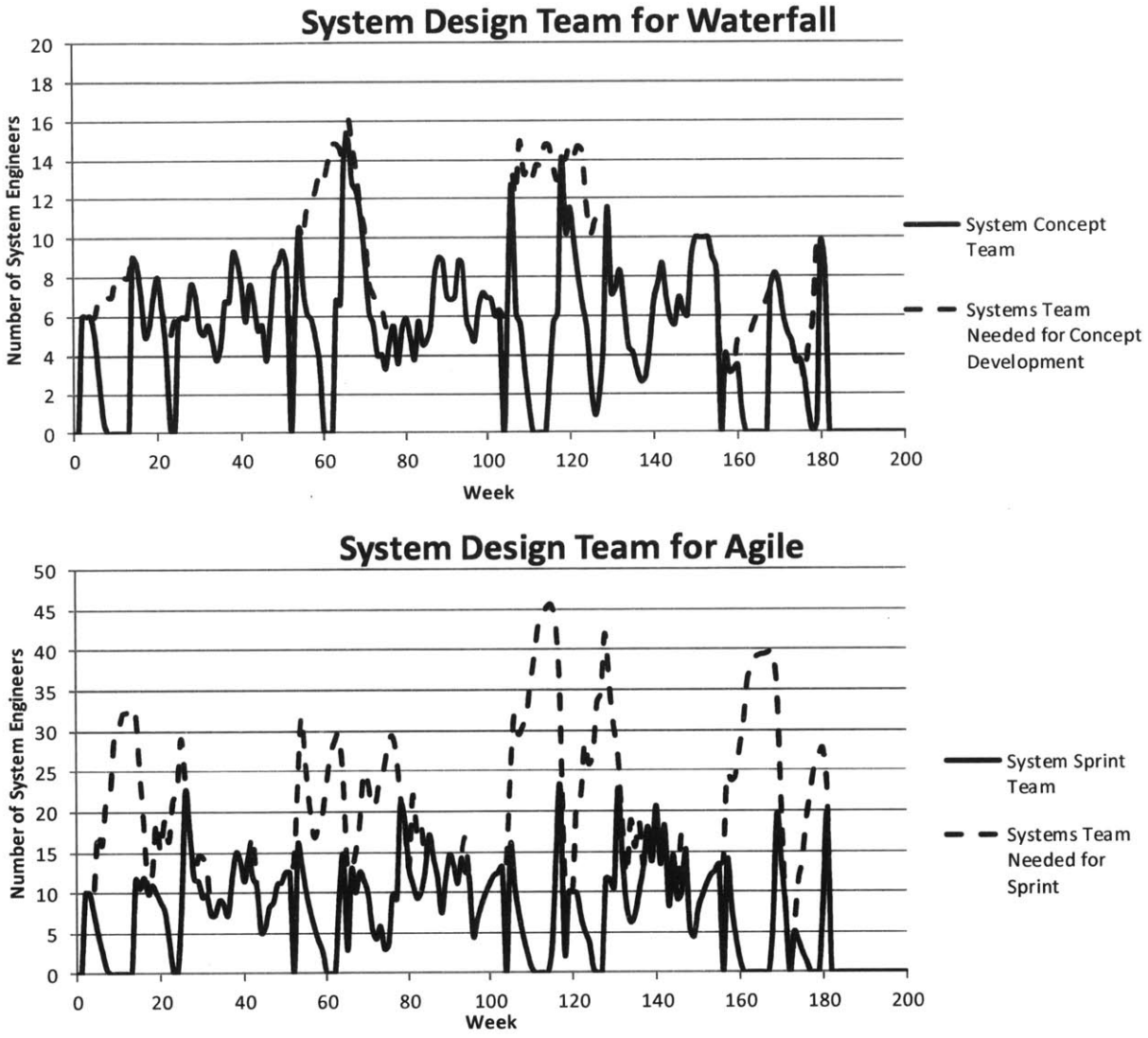
7.2 System Design TRL Coflow during Agile Development



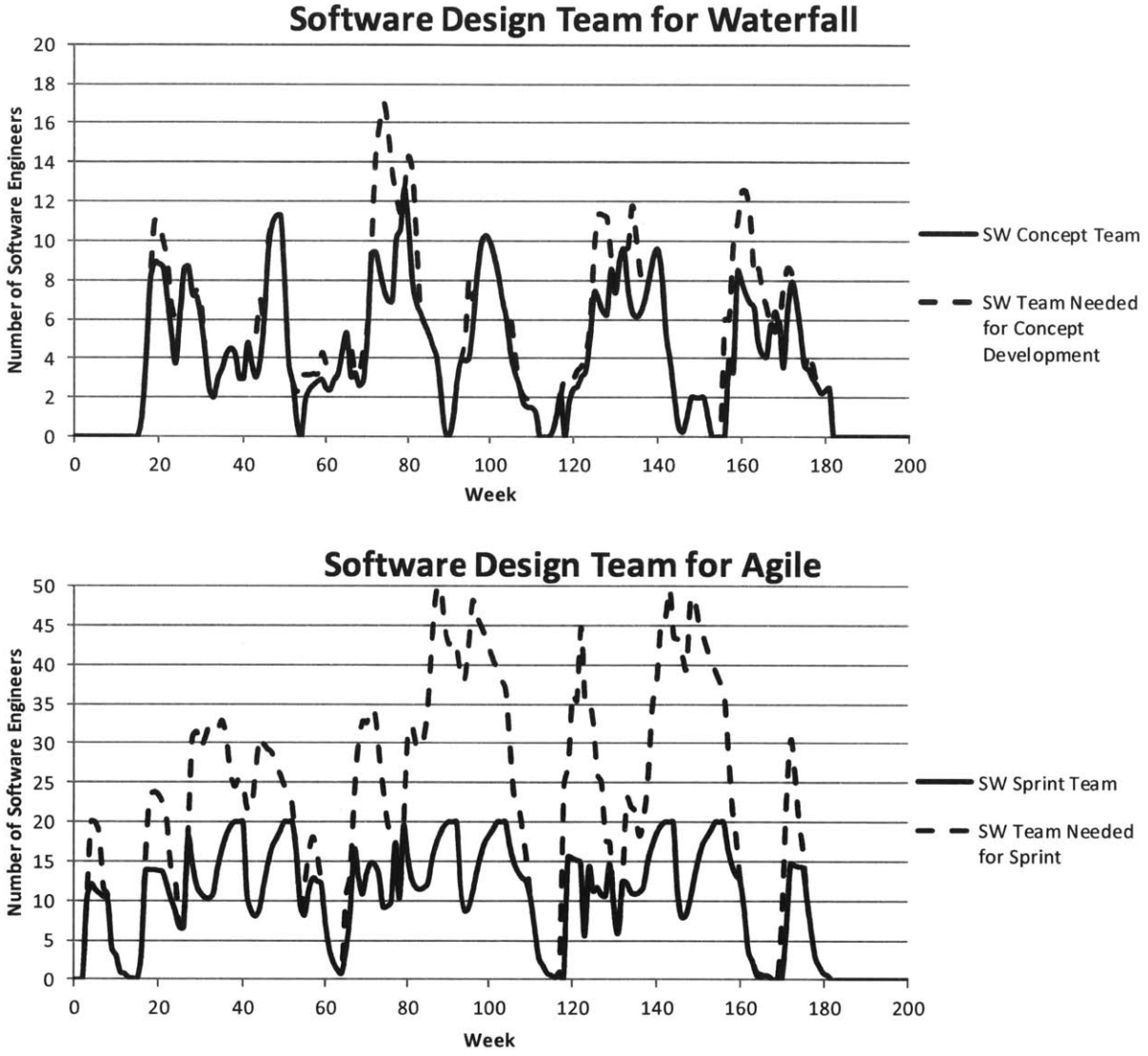
7.3 System Testing for both models



7.4 System Design Team



7.5 *Software Design Team*



8 Works Cited

Abran, A. (2004). *Guide to the Software Engineering Body of Knowledge: 2004 Edition: SWEBOK.* Retrieved February 4, 2014, from IEEE Computer Society: http://common.books24x7.com.libproxy.mit.edu/toc.aspx?bookid=14089>

ACQuipedia. (2013, October 7). Retrieved Feburary 6, 2014, from Defense Acquisition University: https://dap.dau.mil/acquipedia/Pages/ArticleDetails.aspx?aid=a896cb8a-92ad-41f1-b85a-dd1cb4abdc82

Assistant Secretary of Defense for Research and Engineering ASD(R&E). (2011, April). *Technology Readiness Assessment (TRA) Guidance.* Retrieved December 18, 2013, from Defense Acquisition University: https://acc.dau.mil/CommunityBrowser.aspx?id=461216

Bell, T. E., & Thaer, T. A. (1976). Software requirements: Are they really a problem? *Proceedings of the 2nd international conference on Software engineering* (pp. 61-68). Los Alamitos: IEEE Computer Society Press.

Carter, A. (2011, 09 13). Advance Policy Questions for Ashton B. Carter Nominee to be Deputy Secretary of Defense. Washington D.C.

Cocco, L., Mannaro, K., Concas, G., & Marchesi, M. (2011). Simulating Kanban and Scrum vs Waterfall with System Dynamics. *XP* (pp. 117-131). Madrid: Springer.

Cunningham, W. (2011, 01 22). *Ward Explains Debt Metaphor.* Retrieved 12 05, 2013, from http://c2.com/cgi/wiki?WardExplainsDebtMetaphor

De Neufville, R. (2011). *Flexibility in engineering design .* Cambridge: MIT Press.

Decker, G. F., & Wagner JR., L. C. (2011, January). *Army Strong: Equipped, Trained and Ready.* Retrieved January 05, 2014, from U.S. Army: http://usarmy.vo.llnwd.net/e2/c/downloads/213465.pdf

Defense Acquisition Guidebook. (2013, May 15). Retrieved November 01, 2013, from Defense Acquisition Guidebook: https://dag.dau.mil/Pages/Default.aspx

DoD Instruction 5000.2. (2013). *Operation of the Defense Acquisition System.* Washington: DoD.

Frand, E. (1980, December). Erwin Frand's Thoughts on Product Development. *Industrial Reasearch & Development*, p. 27.

Gilmore, J. M. (2011). Key Issues Causing Program Delays in Defense Acquisition. *International Test and Evaluation Association*, 389-391.

Glaiel, F., Moulton, A., & Madnick, S. (2013, March). Agile Project Dynamics: A System Dynamics Investigation of Agile Software Development Methods. Cambridge, MA.

Lehman, M. M. (1980). Programs, life cycles, and laws of software evolution. *PROCEEDINGS OF THE IEEE (Volume 68, Issue 9)*, 1060-1076.

Luisanna Cocco, K. M. (2011). Simulating Kanban and Scrum vs Waterfall with System Dynamics. *XP* (pp. 117-131). Madrid: Springer.

McConnell, S. (1996). *Rapid Development: Taming Wild Software Schedules.* Microsoft Press.

McQuarrie Jr., A. J. (2004, September). *Fire Fighting in Aerospace Product Development: A Study of Project Capacity and Resource Planning in an Aerospace Enterprise.* Cambridge.

MITRE. (2010, December 15). *Handbook for Implementing Agile in Department of Defense Information Technology Acquisition.* Retrieved February 4, 2014, from MITRE: http://www.mitre.org/sites/default/files/pdf/11_0401.pdf

O'Connell, D. (2011). *When Agile Software Development and. Software Architecture Collide.* Retrieved 11 01, 2013, from DEFENSE TECHNICAL INFORMATION CENTER: http://www.dtic.mil/dtic/tr/fulltext/u2/a558044.pdf

Parrish, K. (2013, July 31). *Pentagon Review Reveals Best, Worst Case, Hagel Says.* Retrieved 01 12, 2014, from U.S. Deparement of Defense: http://www.defense.gov/News/newsarticle.aspx?ID=120559

Project Management Institute. (2013). *A guide to the Project Management Body of Knowledge (PMBOK guide).* Newtown Square: Project Management Institute, Inc.

Rahmandad, H. (2005). Dynamics of Platform-based Product Development. Boston.

Rahmandad, H., & Repenning, N. (n.d.). *The Dynamics of Capability Development and Erosion.* Retrieved August 9, 2013, from http://www.sdl.ise.vt.edu/research.html#ProductDevelopment

Rahmandad, H., & Weiss, D. M. (2009). Dynamics of concurrent software development. *System Dynamics Review Volume 25, Issue 3*, 224-249.

Repenning, N. (2001). Understanding Fire Fighting in New Product Development. *Journal of Product Innovation Management*, 285-300.

Royce, W. (1970). Managing the Development of Large Software Systems. *IEEE WESCON*, (pp. 1-9).

Samios, H. (2012, February 29). *Introducing Scrum to an Organization.* Retrieved January 05, 2014, from Defense Acquisition University - Acquisition Community Connection: https://acc.dau.mil/CommunityBrowser.aspx?id=501105

Sterman, J. D. (2000). *Business Dynamics: Systems Thinking and Modeling for a Complex World.* New York: Irwin/McGraw-Hill .

Tavassoli, D. (2007). Agile software development of military embedded systems. *Military Embedded Systems*.

The Agile Manifesto. (2001). Retrieved February 4, 2014, from Manifesto for Agile Software Development: http://agilemanifesto.org/

Ulrich, K. T., & Eppinger, S. D. (2008). *Product Design and Development 4th ed.* New York, New York: McGraw-Hill/Irwin.

Version One. (2009). *State of Agile Survey*. Version One Inc.