DISSERTATION

MERGING SYSTEMS ENGINEERING METHODOLOGIES WITH THE AGILE SCRUM FRAMEWORK FOR DEPARTMENT OF DEFENSE SOFTWARE PROJECTS

Submitted by

Dallas Rosson

Department of Systems Engineering

In partial fulfillment of the requirements

For the Degree of Doctor of Engineering

Colorado State University

Fort Collins, Colorado

Fall 2024

Doctoral Committee:

Advisor: Thomas Bradley
Co-Advisor: Ann Batchelor

David Coleman
Kamran Eftekhari Shahroudi
Dan Wise

ABSTRACT

MERGING SYSTEMS ENGINEERING METHODOLOGIES WITH THE AGILE SCRUM FRAMEWORK FOR DEPARTMENT OF DEFENSE SOFTWARE PROJECTS

Only large-scale Department of Defense (DoD) software projects executed under the direction of the DoD Instruction 5000.2, Operation of the Adaptive Acquisition Framework, are required to follow rigorous systems engineering methods. Many software projects lack the benefits of established systems engineering methodologies and good engineering rigor and fail to meet customer needs and expectations. Software developers trained in the use of the various Agile frameworks are frequently strongly opposed to any development methodology that could be viewed as infringing on the principles of the Agile Manifesto. Agile projects, by their nature, embrace the concept of change, but uncontrolled change leads to project failures whereas controlled change can lead to sustained and innovative forward progress [1]. In order to improve the results of these vital software projects, Department of Defense (DoD) software projects require a methodology to implement systems engineering rigor while still employing Agile software practices. The Agile Scrum framework alone is not rigorous enough to fully document customer needs as User Stories are written tracking only who, what, and why at a non-atomic level and commonly never looked at again after development needs are met. Systems engineering methods alone are not flexible enough to take advantage of the inherent nature to change capability required in software projects, which require flexibility in schedule and requirements. A new methodology, the Systems Engineering Focused Agile Development method, takes a rigor-flexibility-rigor approach to development and makes use of the strengths of

the Agile Scrum framework with the best practices of systems engineering methodologies resulting in a common language that better allows cross-functional teams to communicate project needs while also allowing software developers to maintain flexibility in the execution of software projects. This research has determined that the thoughtful blending of Agile systems engineering and modern systems engineering methods has the potential to provide DoD software projects with benefits to cost, schedule, and performance.

PREFACE

This dissertation has been written to fulfill the graduation requirements of the Doctor of Engineering in Systems Engineering at Colorado State University in Fort Collins, Colorado. I engaged in this research and writing from Fall 2023 through Fall 2024. Although I am the only listed author, many have assisted me along the way to completion of this research.

Prior to my systems engineering journey, my previous studies and professional experience revolved around software development, computer science, and project management. During my professional career working on Department of Defense projects, I found myself struggling to adapt to an Agile project execution environment while maintaining good engineering rigor as higher-level authorities required. Due to this, I found myself drawn to the systems engineering field and focusing my research on the utilization of the strengths of the Agile Scrum Framework while keeping the engineering rigor of systems engineering methods.

I would like to take a moment to thank my committee. I offer special thanks to Professor Ann Batchelor, who served as one of my co-advisors. In trying to find the right "fit" for an advisor, I found that your background as a DoD civil servant and vast engineering experience greatly beneficial. Having the capability of leveraging your hard-earned experience has maximized my learning through this process, for which I am forever grateful. Little did I know that I would find not only an excellent advisor in academic matters, but also a mentor in career and life advice. I am honored to be your last doctoral student that you serve as a dissertation advisor for and will strive to uphold and represent the lofty standards that you have modeled yourself. Dr. Thomas Bradley agreed to co-advise me, regardless of his extremely busy schedule as the Department Head for the Department of Systems Engineering and has been an invaluable

resource for advice and support. Dr. Kamran Eftekhari Shahroudi will always have my gratitude for his candidness and for exposing me to dynamics modeling, which I have fallen in love with and will always be on a path to mastery towards. Dr. Dan Wise served as my external department chair but was no less of a level of importance as any other and not only brought his long serving government experience to bear in my committee, but also gave me a sense of pride in having a brother Submariner on my dissertation committee. Finally, I would like to give a special thanks to Mr. David Coleman who represented the Naval Undersea Warfare Center Detachment Keyport and served as my Department Head in the Fleet Readiness Department. Dave has been a firm advocate for my research and can always be relied on for strong and candid advice, as well as having a key ability to focus one's vision on the target at hand.

Over my academic career, I have had the pleasure to work with many amazing professors who have not only educated me, but also mentored and assisted me in my pursuit of this degree. Dr. Mohamed Ali at the University of Washington advised me during the writing of my thesis at the University of Washington. His favorite phrase "Yes, but where is the science!" has always stuck with me and I will always be indebted to him for that. Dr. Daniel Herber not only taught a rigorous Advanced MBSE course that I attended, but also assisted with co-authoring one of my journal articles. Dr. Herber's openness and mentorship cannot be overlooked. Dr. Daniel Zimmerman, previously of the University of Washington, currently a Principled Computer Scientist at Free & Fair, taught me that I needed to study, and that education is not always easy. His classes were always difficult, but I greatly appreciated the challenge and signed up for him as a teacher whenever possible. These are but a few of the amazing educators that I have had the opportunity to learn from over my greater than 14 years of attending university. It would be impossible to detail them all, but all do have my thanks.

I would like to thank the amazing leaders and mentors I have had the privilege of working with at the Naval Undersea Warfare Center in Keyport, Washington. Without their understanding and guidance, I would not have had as great an ability to perform this doctoral program. Specifically, I would like to thank Dennis Summers for teaching me the importance of the customer and focusing on building relationships, not just good code, Bill Six for putting me on the path of becoming a systems engineer, and Lisa Andrews for all the support she has given me as the Division Head of the Digital Transformation Division.

Without the cooperation and support of the various software teams that I have worked with over my career, this research would not have been possible. A special thanks is given to all the coworkers I have had the pleasure to work with on the Obsolescence Management Information System project over the years, as the seed that grew into this research topic took root there. Thank you for your flexibility with humoring my ideas and your dedication to keeping a growth mindset. This dissertation is as much your success as my own.

Finally, I want to thank my family for always being there for me through this decades-long process. You stoically listened to me panic about grades, complain about school, and worry about how I was not going to make it, always encouraging me down the path I have taken. You all listened to me droll on about systems engineering and project management when I am sure it was of extraordinarily little interest to you. Without your support, I would not have been successful.

Dallas Rosson

Bremerton, WA, 03 OCT 2024

DEDICATION

It is my honor to dedicate this dissertation to my parents, spouse, and children that supported me throughout my education. Thank you for seeing this adventure through to the end.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1 – THE NEED FOR SYSTEMS ENGINEERING FOCUSED AGILE DEVELOPMENT

Introduction

Chapter 1 presents the need for a systems engineering focused Agile development methodology for the DoD. A discussion of the Adaptive Acquisition Framework and unique problems encountered by DoD software projects is presented. Chapter 2 argues the difference between User Stories and systems engineering style shall statements. The case to approach User Stories as goals rather than requirements is detailed. Chapter 3 explores the need for this approach through the results of a literature review and the presentation of 5 case studies detailing successes and failures in software development in the DoD. These case studies and literature result in the identification of three key driving factors of success in such projects: communication, requirements quality, and the ability to adapt to changes. Chapter 4 verifies these three key driving factors using a systems dynamics model of Agile Scrum software execution. This model is used in simulations driven by real world DoD software project execution. Chapter 5 applies the learning from this research by detailing a new process which combines systems engineering methods with the Agile Scrum Framework. Finally, Chapter 6 & 7 present the contributions to research and conclusions resulting from this effort.

It is common practice for software systems to be developed utilizing teams consisting of members drawn from multiple disciplines and differing levels of management [2] [3]. Software developers for the Department of Defense (DoD) often work in cross-functional groups containing other engineers, logisticians, business consultants, and program managers as utilizing

cross-functional teams is one of the key success factors in determining the success of new projects [4]. One of the largest problems cross-functional teams experience is poor communication [5]. Each of these disciplines has its own professional language, patterns, and procedures, which can be conflicting in usage or not easy to understand outside of the field. Utilizing systems engineering methodologies can create a common language for cross-functional teams to operate under and responds and anticipates to an environment that is increasingly complex [6].

Most modern software development utilizes Agile Scrum methodologies [7]. The Agile methodologies take advantage of the attributes of many pure software development projects, which is the ability to quickly change, add, or remove requirements based upon customer needs and feedback. In the Agile Manifesto, the authors note that the core values of Agile development are individuals and interactions over processes and tools, working software over comprehensive documentation, customer collaboration over contract negotiation, and responding to change over following the plan [8]. Agile Scrum accomplishes this by tracking work items in a Product Backlog, executing tasks in short iterations called Sprints, and having a series of meetings designed to facilitate customer and development team communications [9]. This minimizes risk of a failed project "by the fact that there are short iterations and deliveries clearly defined, and second, direct communication with stakeholders, in order to develop the project (as opposed to extensive documentation) " [10]. Because of this, software developers are overly cautious about writing requirements utilizing shall statements, as is standard practice in DoD projects utilizing good systems engineering methodologies [11]. Many avoid creating documents such as Model Based Systems Engineering (MBSE) or Department of Defense Architectural Framework (DoDAF) [12] models, as it requires upkeep, takes away from development time, and requires

the “model space and, therefore, the relations between models and problems as well as solutions” which do not exist, especially early in the development lifecycle [13]. Smaller, less mission critical systems can successfully utilize Agile Scrum as is, but larger and more critical projects with mixed developer skill sets and cross functional team members require explicit documentation to succeed [14]. Agile projects, by their nature, embrace the concept of change, but uncontrolled change leads to project failures whereas controlled change can lead to sustained and innovative forward progress [1]. Avoiding documentation and structure conflicts with current DoD desires to implement systems engineering principles across program office managed projects and those associated with defense acquisition, including utilizing systems engineering to track progress of systems development in a System of Systems (SoS) methodology, which results in independent and useful systems integrated into a larger system that delivers unique capabilities [15] [16].

In Agile Scrum projects, the majority of software projects use phrases called User Stories to track requirements [17]. User Stories are normally written utilizing the Connextra template, which is as follows: “As a {role}, I want {goal}, [so that{benefit}]” [18]. Systems engineering best practices dictate that requirement statements be “single imperative instruction sentences which are made up of one or more clauses” [19]. These requirements are normally written in Shall Statements, such as the following: “The subject shall do something under conditions within constraints/tolerances” [19]. While it is easy to craft a User Story, they have no built-in traceability like Shall Statements do. Shall statements are broken down in a parent child methodology and, through requirements management and traceability, are directly linked to other requirements [20]. User Stories are stored in a repository called a Product Backlog. User Stories are written so that a developer can accomplish the associated work within one Sprint cycle. User

Stories that cannot be accomplished in one Sprint cycle are broken down into multiple smaller stories. These User Stories are of a special type called Epic User Stories and represent the rare case of traceability in Scrum requirements management [21]. There is a much more granular focus on requirements management when generating requirements as Shall Statements than User Stories, which is by design in the Agile framework. Agile is challenged in settings where cross-functional teams must interact, usually with more restrictions than software teams must, and answering to DoD program offices who balance multiple programs, contracting agencies, and other DoD government entities, using User Stories can lead to situations where software developers spend a large amount of time creating a system only to find that the development team missed essential functionality. Software developers carry the burden of translating customer processes and data into a software system, especially in the modern software environment where "systems complexity is increasing and requirements stability is decreasing" [1]. Utilizing User Stories and not more stringent shall statement formatted requirements force developers to have a more in-depth understanding of the problem domain knowledge than would be otherwise necessary. In systems engineering, "requirements that are not specifications are always converted into specifications with the help of domain knowledge" [22] and User Stories innately being high-level and abstract requirements creates a situation where software developers must attempt to translate vagaries into a solution rather than relying on concrete, atomic specifications. This can directly lead to increased schedule pressure on project test teams as new requirements are discovered due to User Stories being innately high-level artifacts leads to a rush at the end of the production cycle for the test team to finish validation and verification, possibly causing shortcuts to be taken and a faulty product delivered to the customer [23]. A merger of the capability to pivot to meet changing customer needs that the Agile Scrum framework offers with the structure

and specificity that Systems Engineering methodologies provide would make a marked process improvement.

The use of User Stories in Agile projects is generally incompatible with formal requirements tracking, as might be required for DoD and Systems Engineering acquisition projects. User Stories are tracked in multiple ways by different teams. While some teams document User Stories in wikis, spreadsheets, or online Scrum Management tools such as Atlassian's JIRA, other teams utilize white boards and sticky notes [24]. Some teams completely drop any tracking of User Stories once implementation has taken place due to the Agile focus on reducing requirements specification and documentation, relying instead on the tacit knowledge of the team to remember what was required and why [24]. Not keeping a baseline requirements repository can lead to issues when referencing "needs" versus "wants." Without a requirement, a test engineer does not have a reference as to what functionality a webpage should display, nor are restrictions and constraints noted outside of any generated test cases. This can lead to test engineers writing test cases to reflect what is in a current "production" environment, rather than ensuring that the system is operating to the standards the customer actually needs. In a 2019 survey of more than 190 test practitioners, there was as strong desire for traceability between test cases and requirements, which is difficult to track with Use Cases and a simple matter when utilizing systems engineering style requirements [25]. By implementing requirements management as prescribed in systems engineering, requirements are kept throughout the lifecycle of the system and traced not only to other requirements, whether functional or non-functional, but also to test cases, use cases, and other necessary artifacts [19].

As stated above, many development teams are cautious to adopt strategies of modeling such as MBSE. This seems to be especially true for smaller teams who might not have additional

funds for a systems engineer or modeler, which leads to the effort of modeling being placed on the software developers. This aversion is not in the best interests of the software team. Modeling in software development is not a new concept. Prior to MBSE, some software development projects utilized a framework called Model Driven Engineering (MDE) [26]. In MDE, systematic use of models as primary artifacts takes place during the software engineering process [26]. Practitioners utilizing MDE were surveyed regarding whether modeling improved their capabilities to successfully develop their software project and whether models improved communication with their stakeholders. The majority of those surveyed responded that modeling positively affected their project development and communication [26]. Models can be used for a broad number of activities, such as network and functional simulation, documenting architecture, requirements traceability, automatically generating code, and more [27]. This not only allows for an ease of planning and structure by focusing on expressing the software in models that use concepts that are much less bound to implementation technology [28], but also decreases the time it takes developers to understand the potentially complex interrelations between software and the hosting hardware and network. This is shown directly in the use of Domain Engineering (DE), which is "a process in which the reusable component is developed and organized and in which the architecture meeting the requirements of this domain is designed" [29]. Models also make the communication of system capabilities to stakeholders much clearer, as the communication is no longer abstract but graphically visible, especially when models are created in compliance with commonly used frameworks, such as the DoD Architectural Framework, with which stakeholders are already familiar. A methodology prescribing the creation of models while utilizing the strength of constant customer communication and feedback that is the hallmark of Agile Scrum is needed to improve software development projects.

On the basis of the above reflections on the state of the field, we can understand that there is a need for a methodology to facilitate the merger between systems engineering best practices and Agile Scrum framework for software development projects, particularly regarding requirements. The Agile Scrum framework is not rigorous enough to fully document customer needs as User Stories are written tracking only who, what, and why at a non-atomic level [18] and commonly never looked at again after development needs are met. Systems engineering methods alone are not flexible enough to take advantage of the inherent nature change capability in software projects, which require "high flexibility and agility due to dynamic and changing environments" [10]. Cross-functional teams can struggle in creating requirements as engineers are taught to think in project execution utilizing the waterfall methodology and in writing requirements as shall statements where software developers have fully embraced Agile and User Stories. This leads to software developers having to translate the needs of engineers into User Stories, thereby potentially losing key pieces of requirements that the software needs. Software developers do not commonly create models as they do not always have the training to do so, models require upkeep, and many developers are not trained in the capabilities that are inherent in models, such as ease of communication and documentation. By creating a new methodology that merges the strengths of the Agile Scrum framework with the best practices of systems engineering methodologies, a common language can be created that will better allow cross-functional teams to communicate project needs while also allowing software developers to maintain flexibility in the execution of software projects.

Software Acquisition in the DoD - Adaptive Acquisition Framework (AAF)

In recent years, the DoD has changed from a predictive, Waterfall approach for software development and acquisition, to an iterative and Agile approach [30] [31]. In traditional major capability acquisitions, the DoD took a very structured approach. The original Defense Acquisition Lifecycle Objectives and Contributions treated all projects in a remarkably analogous manner [32].

Prior to the Adaptive Acquisition Lifecycle, major acquisition projects followed a very structured and serial process [32]. Defense acquisition programs were required to follow the process exactly, including holding specified decision reviews and milestones, as shown in Figure 1: Defense Unique Software Intensive Program Model . These decision reviews and milestones were embedded in the process to assess the readiness to move to a new phase and whether investments should be continued for defense acquisition programs [32]. Due to the danger of potentially shutting a program down or losing funding, many developers would not hold milestone and decision reviews until they were certain that their program would continue on and remain funded [33]. Another reason for not holding these essential meetings is schedule slippage, which occurs due to a variety of reasons. A few of the primary reasons are inadequate systems engineering management, unrealistic schedule expectations, and a misunderstanding of system requirements [33].

The lack, or misunderstanding, of requirements is a major concern when following a traditional, predictive project management approach as requirements are baselined at the beginning phases of a project and are not commonly updated throughout the project execution. For many projects, requirements were either infeasible, unstable, or built in an inefficient and unwieldy manner [33]. Under the previous defense acquisition framework, requirements were generally part of contractual agreements with specific deliverables specified in the contract with

some utilizing the commercial practice of freezing a design prior to production contract award [33]. This translates to serious issues in the execution of projects, especially software projects, where flexibility in requirements is built into software development core capabilities.

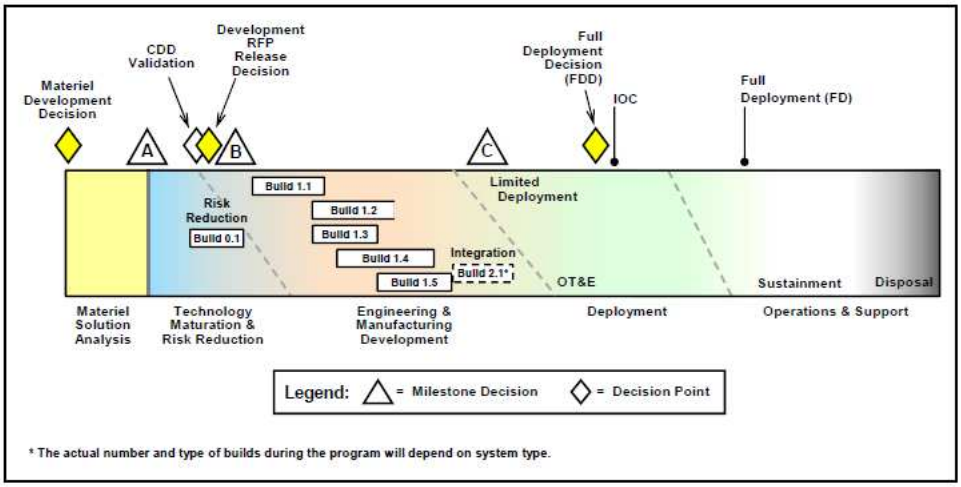

Figure 1: Defense Unique Software Intensive Program Model [32]

Due to complications with delivering software, as detailed above, the Office of the Secretary of Defense for Acquisition and Sustainment determined that a new framework was required. The Adaptive Acquisition Framework (AAF) was developed and deployed to the DoD with the objective of supporting the Defense Acquisition Strategy by delivering effective, suitable, survivable, sustainable, and affordable solutions to the end user in a timely manner [31]. The AAF integrated modern software development practices such as Agile Software Development, DevSecOps, and Lean practices, with an iterative acquisition lifecycle [31]. This new model utilized an iterative approach to defense acquisition program management and development, as shown in Figure 2: Software Acquisition Pathway . This created an environment more suited to developing and delivering software capabilities in a rapid manner, with continuous reviews and customer interaction. It focuses on government-industry teaming

leveraging automated tools for iterative development, builds, integration, testing, production, certification, and deployment of capabilities to the warfighter [31].

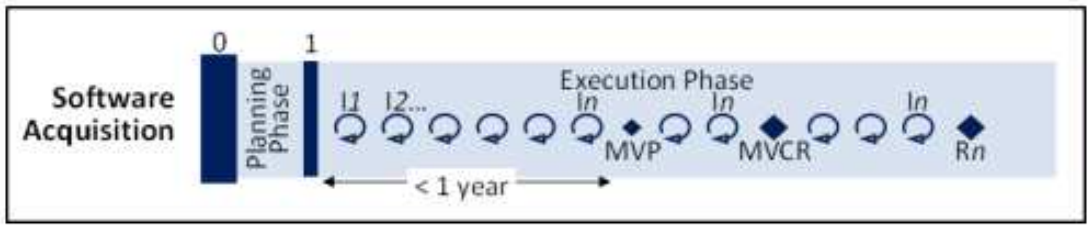

Figure 2: Software Acquisition Pathway [31] [34]

The AAF has been a great step forward for DoD acquisition programs, but it is not readily applicable to smaller programs, especially those without program office oversight. These programs are often developed with alternate development stream funding, such as the Naval Innovative Science & Engineering funding or command overhead funding. These types of funding are not directly controlled by a program office and are most commonly used to develop new capabilities. The problem is that when these types of funding are used, there is no structured oversight to ensure engineering rigor. Many projects are developed by a small team that utilize pure Agile methodologies. Most do not have requirements, and do not have baseline because with individual research-funded projects, there is no customer driving the requirements. In larger projects involving contractors, scope may change but contractual obligations ensure that requirements are being met [35]. This is not always true for government led projects developed by government workers. There are no contractual obligations for requirement generation or management, nor for any other form of engineering rigor.

The AAF as described in the DOD 5000.87 is vague in its guidance as to how requirements will be managed. The DOD 5000.87 states “Programs using the software acquisition pathway are not subject to JCIDS, except pursuant to a new process as discussed in Paragraph 2.8.a., but must be effective in capturing users’ needs, priorities, and environment.”

Further, it goes on to state that only a Capability Needs Document must be generated with further requirements created in a "streamlined and coordinated" process [36]. There is no mention of shall statements, but rather an example of User Stories or features [36]. As stated earlier, User Stories and features do not provide adequate depth and rigor to ensure customer needs are being satisfied as well as a clear definition of system functionality as required for proper in-depth testing adequacy. Programs following the AAF are not required to deliver a detailed requirements document or repository at the completion of a project, but rather, a repository of higher-level User Stories. User Stories do not provide the traceability that shall statement style requirements do, which suggests that these projects are missing essential and minimal technical documentation that will be needed by test and evaluation teams as well as future development teams. Development strictly following the AAF will not meet good standards of systems engineering practice.

Need for Systems Engineering Focused Agile Development – Conclusions

Based on these reflections, there is a need for a methodology that combines systems engineering methods with the Agile Scrum Framework. DoD software projects have much higher stakes associated with them than most civilian projects. This comes in the form of not only the misuse of taxpayer dollars but also the potential loss of life through mismanagement or defective weapons systems development, operation, and sustainment software. Due to this, thorough engineering and technical rigor is needed to ensure the successful execution of the software projects. As these projects are software projects and are not conducive to Predictive style project management, an Agile methodology is still warranted. Agile Scrum is the most common and popular Agile development methodology [7]. Therefore, a thoughtful merger of systems engineering methods in conjunction with an Agile Scrum execution is called for.

CHAPTER 2 – A COMPARISON OF REQUIREMENTS MANAGEMENT IN AGILE VS PREDICTIVE PROJECTS

This chapter seeks to answer research question 1, which is stated here as follows:

Research Question 1: What are the interrelationships between project requirements and changes that drive project success when utilizing Agile Scrum User Stories versus systems engineering shall statements?

Systems engineering principles prescribe user, or customer, driven requirements, as does Agile Scrum requirements generation. However, Agile Scrum development teams frequently readjust requirements and scope based upon customer feedback. Systems engineering requirements management breaks down requirements into much more granular, or atomic, levels, ensuring customer needs are being met. Agile Scrum developers utilize User Stories to create higher level, and more abstract, requirements, allowing for flexibility in development. Traditional Waterfall requirements management methodologies are "failing to design, build, and deploy effective systems within time and cost constraints" [1]. Merging the Agile Scrum practices with systems engineering requirements management would allow software developers to refine requirements initially and throughout the software development lifecycle, ensuring that software developers have a better understanding of what the end user needs and allowing the customer to have a more solid grasp and control in what is going to be developed.

Research Question 1 – Tasks

- *Task 1.1: Perform a focused, diagnostic literature review of systems engineering requirements management methodologies via journal articles, conference proceedings, and other peer reviewed sources.*

  Review performed with an attempt to put a special focus on requirements management with regards to software development.

- *Task 1.2: Perform a focused, diagnostic literature review of the Agile Scrum Framework via journal articles, conference proceedings, and other peer reviewed sources.*

  Review performed. The review was expanded to include professional blogs and websites dedicated to Agile execution, specifically Agile Scrum, as many Agile practitioners do not publish journal articles, but rather focus on the faster turnaround of web content communication.

- *Task 1.3: Document the differences between Agile Scrum requirements generation vs systems engineering best practices in a SysML model, noting the contents and relationships, such as traceability and process enablers, among the components of each.*

  The SysML documentation was completed and is presented in the following text entitled "A Comparison of Requirement management in Agile vs. Predictive Projects."

- *Task 1.4: Document the benefits and drawbacks of each methodology, specifically with regards to software projects. These measures and corresponding metrics will reflect project execution time, rework required, project success, and other metrics.*

  The documentation of the benefits and drawbacks for each methodology is presented in the following text entitled "A Comparison of Requirement

management in Agile vs. Predictive Projects." Metrics focused on Story Point execution, traceability and linkages between system artifacts, and cost, schedule, and performance.

A Comparison of Requirements Management in Agile vs. Predictive Projects

There are many differences between projects managed with an Agile methodology which approaches project management through an iterative and flexible approach versus a Predictive methodology, such as the Waterfall methodology, which relies on upfront and static planning of work and requirements. Agile projects seek to focus on customer needs and desires without a high burden of documentation and planning. These types of projects prioritize flexibility to be able to pivot quickly with changing customer needs. In general, Agile practitioners rely on "stories" in lieu of requirements to explain customer needs to a development team. In contrast, Predictive projects focus on planning projects out in detail to capture all customer needs, goals, and requirements in documentation, with the objective of clear communication and effective capture of project scope. The Department of Defense (DoD), along with many other groups and companies in the software engineering industry, are increasingly adopting Agile project methodologies, but this article describes cases where traditional, predictive project management is not only preferred, but more appropriate. By comparing the techniques of requirements management, of traceability transformation, and of goals to requirements refinement, this article will illustrate the costs and benefits of each requirements management framework for DoD software development projects.

Overview of Agile Software Development

Agile project management is a flexible form of project management, primarily used in software projects. The Agile Manifesto sets the baseline standards of what an Agile project should strive for. The original authors of the Manifesto strove to "uncover better ways of developing software" [8] [37]. Using these principles, multiple ways of executing Agile projects have come into being since then, including Crystal, eXtreme Programming, Scrum, and numerous others [38]. Agile practices have further evolved to program and business management through the utilization of scaling agile with methods such as Scrum of Scrums (SoS) or the Scaled Agile Framework (SAFe) [39].

Agile project management is most useful when a project is high risk, low certainty, and requirements are only partially known [40] [41]. Agile Scrum lowers project risk by employing an incremental delivery method and frequent communications and demonstrations with customers [41] [9]. As shown in Figure 3: Life Cycle with Adaptive Development Approach , software development teams work on small increments of projects, delivering minimum viable products to customers, and utilizing a customer feedback loop to integrate changes driven by delivery of product. This iterative and incremental approach helps to lower risk and optimize predictability through customer and subject matter expert engagement with the development team [9]. By focusing on the most important need of a customer and then iteratively revisiting the project requirements, Agile methods make use of Lean thinking, reducing waste and focusing on what the customer needs, lowering overhead burdens [43].

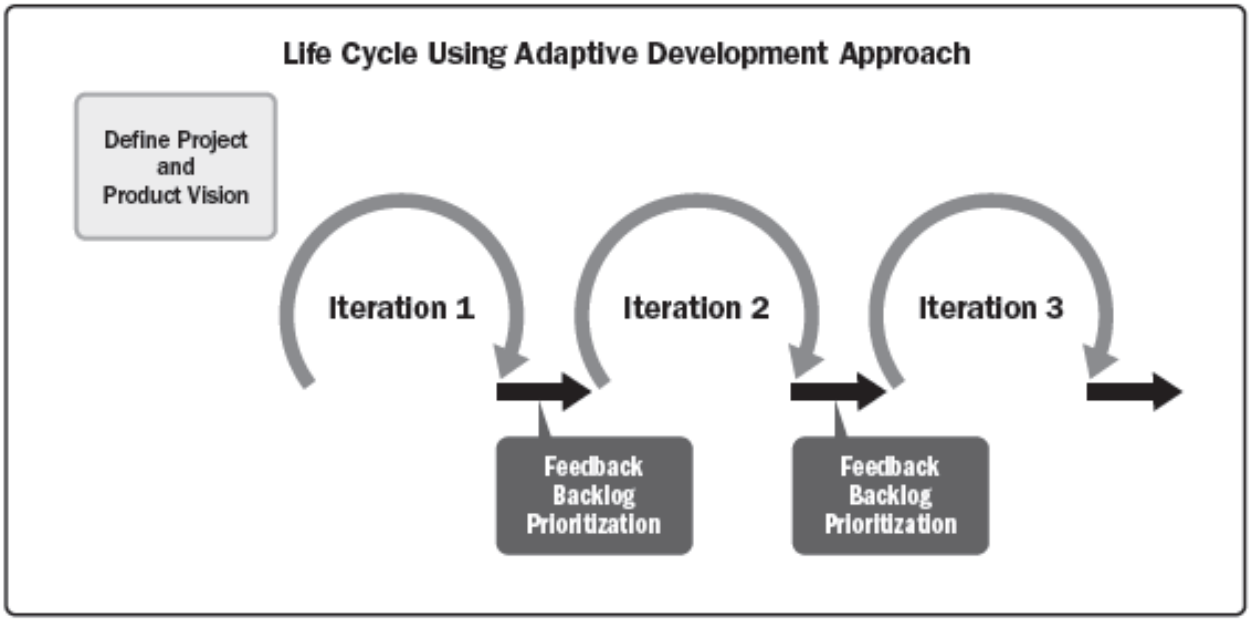

Figure 3: Life Cycle with Adaptive Development Approach [42]

While there are many forms of Agile program management, the most utilized is Scrum [44] [45]. Scrum is so widely used that many people assume that Scrum is the only form of Agile, so much so that practitioners have made many websites dedicated to explaining that Scrum is not the only Agile framework [46] [47] [48] [49]. The human mind struggles to perform complex knowledge work in detail such as planning, understanding scope, task creation and assignment, consequences of actions across a system, and estimating complexity [50]. Scrum assists with this effort by optimizing heuristics, providing continuous feedback, and creating external, cognitive artifacts, such as sticky notes, models, the Scrum Board, etc. [51] [52]. Figure 4: Example Agile Scrum Development Lifecycle shows an example of the Scrum product lifecycle. This lifecycle is a constant loop, progressing through the closed loop stages of Plan, Build, Test, Deploy Increment, and Review. During the Build Stage, a Daily Scrum activity occurs. After the Review stage, a Final Product might be produced if the Product Owner deems

the product ready for production. This execution loop occurs each Sprint and continues until project completion.

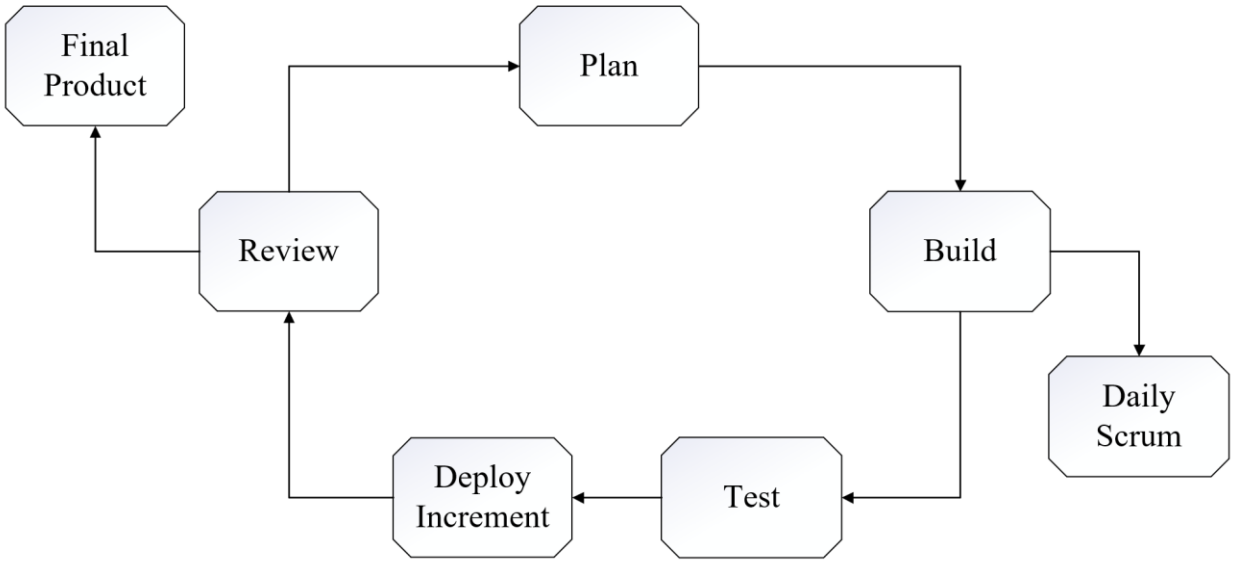

Figure 4: Example Agile Scrum Development Lifecycle

Agile software development focuses on lightweight, flexible, and adaptable project execution. It is most useful when a project is high risk, low certainty, and requirements are only partially known [40] [41]. Agile Scrum further codifies the broader Agile methodology by implementing a structured framework which includes an iterative development cycle called a Sprint. Agile Scrum practitioners plan projects in Sprint increments, modifying execution plans as customer needs and requirements change. Through the utilization of Agile project management tools, Agile practitioners can swiftly adapt to changes to projects with a minor impact on cost, schedule, and performance.

Overview of the Predictive Development Methodology

Traditional, predictive project management its predicated on a predictive lifecycle and has been around far longer than any other form of project management, having evolved during

the 1950s [40]. Predictive lifecycles are expected to have little flexibility or change over the course of execution, have readily available and stable team members, and accept negligible risk [41]. These projects are most successful when “project and product requirements can be defined, collected and analyzed at the start of a project” [42]. For a project to be successfully executed using a predictive method, it must have a very well written set of requirements, a stable funding stream, a consistent and well-organized team, and high confidence that changes will not be forthcoming. This type of project management is well suited for construction, with a repeatable and sequenced progression of work [41] [42]. An example predictive lifecycle flow is shown in Figure 5: Example Predictive Lifecycle where project phases are executed in a serial manner, starting with the Plan phase and working through all phases until the Deploy phase is completed. Many modern software projects are often have evolving project and product requirements, are high risk, and have significant uncertainty, making it hard to apply the predictive development approach to these types of projects successfully [41] [42] [40].

Figure 5: Example Predictive Lifecycle

The predictive methodology of project management is a linear and serial effort. The previous phase must be completed prior to moving to the next phase [42]. Phases are not repeated, and work products should not be touched upon again [40]. The only feedback loop that is present in this type of model normally occurs during the Test portion of the lifecycle. Defect fixes and changes can be approved and performed during this period. One of the major issues with this linear model is that when changes are requested or defects occur, there is a resultant

extension of the project completion date, which creates Earned Value compromising problems [40].

When applied to software projects, predictive development has many issues. Due to the strict adherence in an attempt to "provide straightforward, systematic, and organized processes" which involves detailed planning and documentation, it commonly fails to adapt to the "rapidly changing business requirements", a focus on project metrics, such as Earned Value, and less of a focus on flexibility in execution for process improvement [53]. For the predictive approach to work, a large emphasis on up-front planning, measurement, and control is put in place, which is a very good use of time reserves when very little is planned to be changed in a project or when a product is difficult to change after development execution begins [42]. Neither of these things are true in modern software development, making the predictive method more costly due to a larger overhead and planning burden along with inflexibility in project execution. Metrics of project performance for predictive projects are often not a reflection of reality with the most common, Earned Value, lacking the ability to track software execution and health [41]. Due to these reasons, most software development projects have moved away from the traditional, predictive project management methodology.

Overview of Requirements Management in Predictive and Agile Project Management

Requirements management is approached differently in Predictive projects and Agile projects. Predictive projects generate detailed requirements documents that are baselined prior to the start of development efforts [42]. Agile projects use more flexible requirements, generally gathered at a very high-level, to form an idea of what a project goal should be prior to the start of development [41]. Predictive focuses on having a solid plan and sticking to it, where Agile focuses on flexibility in execution and welcoming changes [8] [41] [42].

Predictive and Agile methods are dramatically different and are best applied in different scenarios. The predictive approach is best suited to projects where a detailed requirements document can be generated and when the project has a minimal risk of change and uncertainty. In these cases, detailed planning can lead to cost savings through avoidance of unnecessary schedule slippage and cost planning. Predictive approaches are most suited to applications such as manufacturing and construction, where the final product is known, tolerances are documented, and the customer has a low likelihood of presenting changes to requirements [40] [42]. Agile methods are suited to projects with a high amount of uncertainty, medium to high risk, and where it is easy to pivot, and course correct in the middle of project execution. In these types of projects, requirements do not need to be fully specified prior to the start of development, but rather elicited and documented at a higher level, noting the needs and goal of a customer, but not necessarily down to the atomic level [8] [40] [41]. This is especially applicable in the software development industry, as producing code has little overhead cost outside of labor. There are no concerns of retooling, logistics and supply issues, nor burdensome preplanning required.

Requirements management in predictive projects often follow the IEEE and INCOSE standard of shall statements. IEEE stipulates that these statements “translate or expresses a need and its associated constraints and conditions” and are “written in a language which can take the form of a natural language” [54]. INCOSE further defines requirements utilizing specified language, such as a shall statement [55]. These requirements are very detailed and must meet the following characteristics: necessary, appropriate, unambiguous, complete, singular, feasible, verifiable, validatable, correct, and conforming [55]. Furthermore, the set of requirements must adhere to the following characteristics: complete, consistent, feasible, comprehensible, validatable, and correct [55]. Requirements management in predictive projects is therefore a

detailed and a large undertaking. The goal of an initial baseline requirements set is to have a fully formed vision and understanding of the proposed system.

In contrast, Agile DoD projects usually generate requirements in the form of Connextra User Stories. The Connextra form of User Story is written as the following: “As a {role}, I want {goal}, [so that{benefit}]” [7]. These User Stories became popular through the ideas of eXtreme Programming, where development teams use User Stories to “understand the flow of requirements from initial ideas to final product” [8]. These User Stories are stored by the developers in a backlog where they are prioritized by a Product Owner that sets priority according to their understanding of customer needs and desires [2].

User Stories are tracked via a system called “Story Points.” Story Points “rate the relative work, risk, and complexity of a requirement or story” [41]. As it is difficult to know exactly how much work, risk, or complexity a work item will have without breaking the item down into an atomic, easily testable, and understandable state, this can be a difficult task to accomplish. Many teams use a form of Story Pointing called Planning Poker to form a level set baseline for Story Point values [56] [57]. Each team has a different method to set point equivalency, and it can take time for the point values to stabilize with a new team. Experienced developers are of great assistance in this aspect, and it is recommended that teams be made up of junior and senior developers, as a study performed by Mahnic et al. found that “optimism bias caused by group discussion diminishes or disappears by increasing the expertise of people involved in the group estimation process” [57]. Furthermore, the complexity of estimation increases as groups can become over-confident in the accuracy of their estimates when given tasks more difficult than are usually presented, individual software development capability does not equate to estimation expertise, software developers often have problems predicting their own capability of work, and

overall experience can be very “narrow” [58]. Therefore, while Story Pointing efforts can be effective in an experienced and fully realized team with mixed experience levels, there are valid concerns regarding the validity of storming to norming teams and their ability to adequately estimate software project workloads.

So, although both Agile and Predictive methods use these techniques to develop and communicate customer intent, in practice, they are not equivalently effective in a variety of project contexts. In “Do Agile Methods Work for Large Software Projects,” it was reported that 46% of customers report Agile projects as “unsuccessful” within the boundaries of client benefits, cost control, and time control [9]. Furthermore, for Agile projects to be successful, customers must actively participate in the process. Customers are charged to be “Collaborative, Representative, Authorized, Committed, and Knowledgeable” [91]. Communication is a driving factor for failure in Agile projects as well as good requirements development [10].

Through the implementation of systems engineering requirements management, communication between stakeholders and developers is focused and clear. This is especially true in the instantiation phases of projects where Needs, Goals, and Requirements are generated, even if they are solely high-level and abstract. The process of using User Stories can lead to vague requirements for Agile projects and misunderstanding or confusion by the development team due to User Stories limited and abstract nature. This can cause communication issues when stakeholders do not have high availability, directly leading to problems in cost, schedule, and performance.

Requirements management is approached differently in Predictive projects and Agile projects. Predictive projects use a rigorous methodology of identifying and detailing requirements prior to the start of development execution in a project. Agile projects often use a

abstracted form of requirement tracking, often using User Stories. Predictive projects suffer from the inability to be flexible when changes in requirements occur. Agile projects suffer from User Stories not communicating customer needs in detailed enough form. There are benefits and drawbacks to how Predictive projects and Agile projects approach requirements management.

A Comparison of Requirements Traceability

Introduction to Requirements Traceability

The traceability of requirements to needs and to requirement owners is crucial to project success and to managing relationships between developers and customers [5]. In the 1994 paper "An Analysis of the Requirements Traceability Problem", Gotel and Finkelstein define requirements traceability as "the ability to describe and follow the life of a requirement in both a forwards and backwards direction (i.e., from its origins, through its development and specification, to its subsequent deployment and use, and through periods of ongoing refinement and iteration in any of these phases)" [11]. In the text "Requirements Engineering," Dick et al. defines traceability as "the relationship of requirements to other sources of project data, including the drivers of stakeholder needs" [12]. In the common vernacular, traceability is the linkage of a requirement to any information source that drives the requirement. Mature projects include traceability not only during one point of time, such as between customer requirements and system requirements, but also throughout time as requirements change and evolve [13]. Through the application of traceability, the disparate artifacts that encompass a system are connected through linkages to other system components. Figure 1: Traceability Transformation shows how individual and disconnected system artifacts flow through a transformation activity where in needs, goals, and requirements are traced directly to the artifacts resulting in a network

of system components, all traced through interlinkages. Maintaining traceability of system artifacts to individual requirements creates greater visibility and understanding of the system.

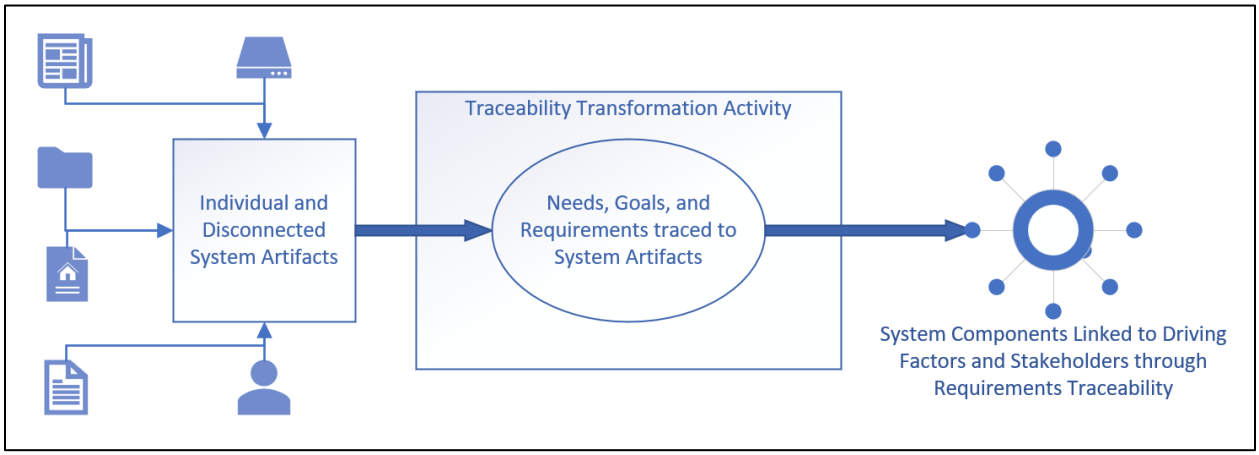

Figure 6: Transformation of Disconnected System Artifacts to Requirements Traced System

When performed in a structured manner, traceability can assist in moderating system complexity through visibility system component linkages to requirements, thereby increasing understanding of the system of interest. Whether this is through traceability management through diagramming and modeling, or through more traditional traceability through documentation, traceability is "a key tool to help define and manage complex systems … to ensure what is needed is included and what is not needed is excluded" [55]. Through the use of a thorough and mature traceability method, a better understanding of complex systems is achieved, lowering the risk of missing key stated or derived requirements, thereby decreasing the realization of catastrophic edge cases occurring [59]. In particular, the morphic nature of software and software development creates multiple factors that drive complexity through an almost unlimited capability of software to realize anything the mind can imagine, the presence of instant distribution to the public, and the integration of black box commercial off the shelf components, making requirements traceability not only desired, but necessary, for a fully understood system [60].

Agile Requirements Traceability

Many Agile Scrum teams recognize the importance of requirements traceability as a key enabler in maintaining customer focus during the software development process [61]. Requirements traceability is not inherent in Agile Scrum and is not mentioned in the Scrum Guide, Agile Software Development, or the Agile Practice Guide [41] [9] [62]. Agile projects generally track requirements at four levels, Epics, Features, User Stories, and Tasks. A search of Scrum.org returned a blog discussion where a developer asked the question of "Is it worth using Requirement Traceability Matrix in scrum?" [63]. The general discussion posited that it was up to the team and asked whether it would add value [63]? This represents the feeling of Agile Scrum practitioners where engineering and technical rigor are not normative, but instead only enacted if viewed as a net-gain in value. Due to this, traceability utilization beyond the hierarchy prescribed by basic User Stories is not standardized across the development industry.

The most basic hierarchy that is found in most Agile Scrum Product Backlogs follows the format of Epic -> Feature -> User Story -> Task. Epics are the highest form of User Story representing an idea that is too large to comprehend by itself or more clearly, "a large User Story that cannot be delivered as defined within a single iteration or is large enough to be split into smaller User Stories" [64] [65]. Many teams use Epics to track unrealized ideas within a Product Backlog prior to a team identifying the deliverable needed [66]. Features are the next step down from Epics. The relationship between Epics and Features is a one-to-many relationship [67]. The term Feature is not well defined but are generally realized as larger groupings of User Stories that are too big to be depicted as a single item, but too small to be considered Epics. User Stories clustered in this way are tied to specific common functionality [61] [68] [67]. Below Features in the hierarchy is the User Story. The relationship between Features and User Stories is a one-to-

many relationship. A described above, User Stories are an “end goal, not a feature, expressed from the software user’s perspective” that is to be realized in the software system [69]. Each User Story should be small enough to be completed within one Sprint cycle [67] [70]. In the most basic of terms, a User Story is “something a user wants” [71]. Finally, User Stories can be further broken down into Tasks [70]. The relationship between a User Story and a Task is a one-to-many relationship. This hierarchical traceability is shown in Figure 7: Agile Scrum User Story Hierarchy.

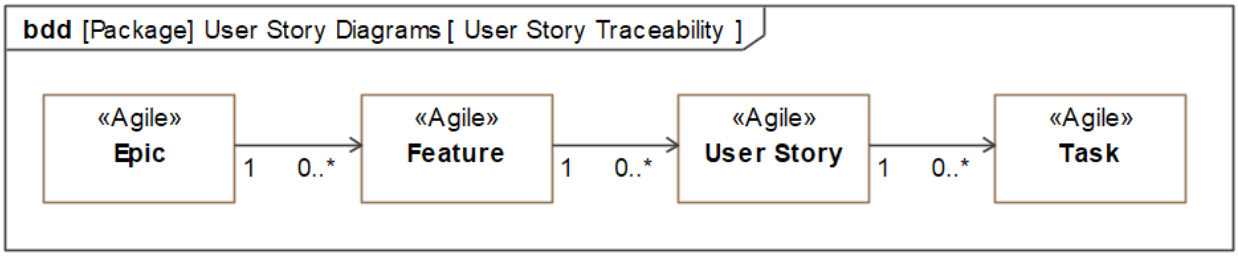

Figure 7: Agile Scrum User Story Hierarchy

Agile Scrum User Stories track traceability through natural language in the Connextra format. In the statement “As a {role} …,” the role keyword is replaced by a user type. Many times, this is expressed as a defined “Persona” [72] [73]. The purpose of the persona is to assist the Agile Scrum team in discovering the value of the product they are making [72]. A Persona is “a customer profile that puts a face on the facts by encapsulating and representing the data” [74]. In other words, Personas are a fictionalized representation of a customer type that Agile Scrum software developers use to gain insight into customer needs, goals, and requirements [75] [73]. This traceability can be useful but is limited as it does not identify specific stakeholders as requirement owners. Rather, it identifies customer types, which software developers may or may not have access to for elaboration of needs, goals, and requirements. Agile Scrum teams that do not use Personas often do not have firm definitions of user roles, limiting traceability use.

Software developers have not settled on a single method for more rigorous and thorough requirements traceability. Some teams opt to use Excel style spreadsheet for requirements tracking, but this can lead to increased costs of traceability creation and maintenance [76] [77]. Other teams may opt for a project wiki to gather various artifacts, including requirements documentation, to allow for an ease of editing as many wikis have versioning built into the core functionality [77]. Various software applications have appeared on the market to solve the traceability issues inherent in software design, specifically with User Stories. Atlassian's Jira tool allows software developers to link work items such as User Stories and Tasks to test artifacts, defects, change requests, and other items [78]. None of these methods are industry standard and teams use whatever is preferential internally.

While these examples are a few ways that Agile Scrum User Stories attempt to meet requirements traceability, each project implements traceability differently. Precluding organizational directives, each Agile Scrum team is free to cherry pick what parts of User Story requirements traceability that the team desires. Some Scrum Teams will ensure that there is as high a level of traceability as possible for their projects. Other teams will ignore traceability all together, outside of the minimal traceability User Stories have built in inherently. Therefore, there cannot be a definitive case set forth that the majority of Agile Scrum teams utilize good requirements traceability.

INCOSE Requirements Management Traceability

According to the INCOSE Guide to Writing Requirements, traceability is "critical to managing relationships" [55]. It is therefore not surprising that the INCOSE methodology of requirements management has traceability inherent in the system. Requirements traceability takes many forms in the INCOSE requirements management methodology. Establishing

traceability is an essential activity in the Technical Processes as described in the INCOSE "Systems Engineering Handbook" [79]. In the INCOSE standard, traceability allows systems engineers to "understand the identify, location, relationships, pedigree, origin of data, materials, and parts of the objects/entities/items" [79].

Requirements, both as a set and individually, are traced both vertically and horizontally [55]. Vertical traceability is traceability of a hierarchical nature [79]. Practitioners commonly refer to relationships of this type to as "parent-child" relationships. The parent is considered directly "above" the child in the hierarchy. Tracing requirements in this manner allows for a top-down perspective of a system and defines system hierarchical architecture. Horizontal traceability is used to trace requirement data through the system lifecycle, as well as across a given level of system architecture [55] [79]. Linkages of items generated in a lifecycle stage that are to be used by another lifecycle stage is an example of horizontal traceability. Horizontal traceability relationships are referred to as "peer" relationships [79]. By tracing relationship connections both vertically and horizontally, a network representing the full view of the system of interest emerges, decreasing the effect of large and interconnected systems complexity.

Due to the way that requirements are documented in INCOSE requirements management, any item can be traced to any other item. Documentation of linkages to other artifacts that each requirement traces to is part of a requirements register, linking each unique line item to other unique line items, as shown in Figure 8: INCOSE Requirements Management Traceability. Thorough requirements traceability is not only encouraged but required in the technical processes [79]. Without the built-in traceability, many requirements registers would be flat files and indecipherable to most people. Examples of common traceability found in INCOSE requirements are parent-child relationships, source derivation, interfaces, and dependencies [79].

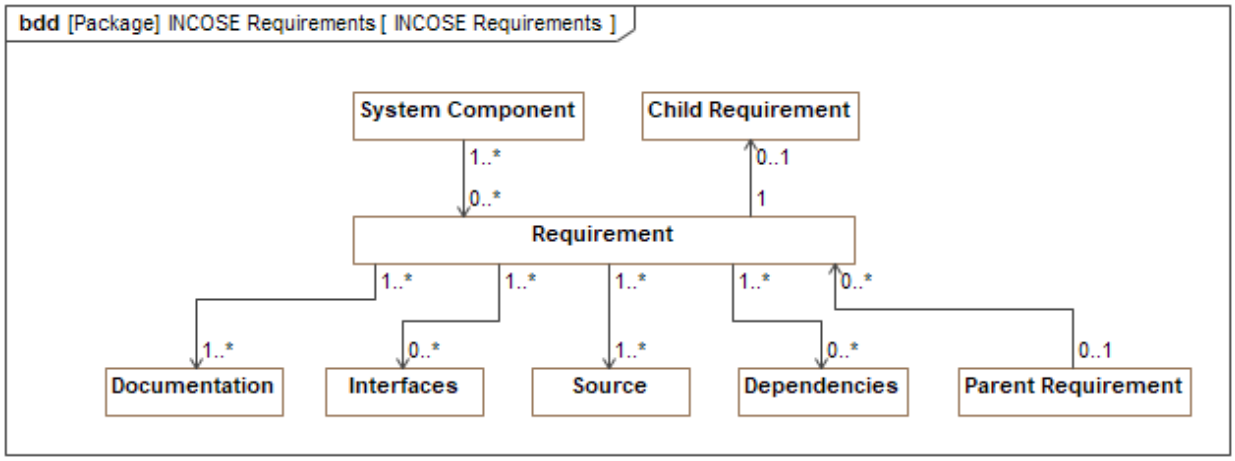

Figure 8: INCOSE Requirements Management Traceability

Model Based Systems Engineering activities expand upon traceability in INCOSE requirements management, directly increasing the level of visibility and understanding that requirements offer. As MBSE efforts, specifically SysML, rely heavily on INCOSE requirements definition and standards, MBSE should be considered a part of INCOSE Requirements Management rather than a neutral third-party methodology. Requirements managed in SysML allow for additional traceability relationships realized through metadata connections. These relationships include, but are not limited to, derive, refine, satisfy, verify, and trace [80] [81] [82]. The trace relationship is a generic relationship that is discouraged in favor of more precise relationships [82]. When making use of other MBSE tools such as Model-based Structured Requirements (MBSR), even more detailed traceability is allowed for, such that attributes "can be linked to other model elements that represent aspects of the physical system, precise mathematical conditions, test cases, etc." through the use of stereotypes [83]. A stereotype is a form of metadata that defines how an existing meta class may be extended enabling the use of platform, domain specific terminology, or a notion in place of, or in addition to, the ones used for the extended meta class [84]. Further, documenting the rationale of requirements is incredibly

important to traceability and the understanding of the "why" of the requirement itself. A MBSE engineer can easily accomplish this through the <<rationale>> stereotype in MBSE linked to a textual comment detailing the rationale [85]. With modeling, linkage relationships are easily understood by a larger audience as the media presented does not require mental modeling to visualize. Rather, the connections are plainly drawn in a graphical manner.

The INCOSE methodology of integrating traceability directly into requirements management in an open and flexible manner is much more robust and useful for system modeling and understanding complex systems. INCOSE requirements traceability allows for modularity through the flexibility to allow a requirement to be traced to any system element, giving clear visibility into how requirements are being met. This visibility directly results in increased verification and validation capabilities through a clearer understanding of the system. Utilizing MBSE techniques, requirements can be traced directly to system architecture, better displaying how requirements are being realized through implementation. Tracing requirements back through derivations, sources, interfaces, documentation, and many other inputs allows systems engineers to fully conceptualize the problem space, building a mental model of not only what needs to be built, but also how and why each requirement is necessary.

Requirements Traceability Conclusions

Both Agile Scrum User Stories and INCOSE style requirements management have a form of traceability inherent in the system. The traceability is present in diverse ways in both systems. INCOSE requirements management uses a structured, bidirectional, and hierarchical methodology that has firm rules that requirements engineers must follow [55] [79]. Agile Scrum User Stories allow for a more diverse inheritance process, but it is generally considered a good form to follow a top-down hierarchical structure. Requirements management style shall

statements have inherent traceability to any other item that may affect them, whereas Agile Scrum User Stories are self-contained, except in cases where teams use metadata, which is not generally proscribed. Both styles of requirements have traceability, though INCOSE requirements management presents a more robust capability.

While both Agile Scrum User Stories and INCOSE requirements management methodologies include built in traceability, User Stories' traceability is not as robust as requirements management style "shall" statements. User Stories have minimal traceability, prescribing only a "parent-child" relationship. This restriction can be overcome but requires outside intervention using such tools such as Jira. INCOSE requirements management builds modular and flexible traceability into the system, allowing for traceability to any other requirement or system element. Not only does it allow for a greater range of traceability capability, but requirements management also dictates that all requirements should name types of traceability outside of the "parent-child" relationship, such as requirement derivation, verification methods, how requirements are refined, and how requirements are satisfied in the system. This allows for a more robust mental model to be built by the development team, facilitating an increased likelihood of delivering what the customer needs while minimizing rework that results in project cost, schedule, and performance slippages. It is clear that although User Stories do have minimal traceability inherent in the Connextra format, INCOSE requirements management outperforms User Stories in every metric.

Goals versus Requirements

Introduction to Goals

ISO/IEC/IEEE 29148-2018 states the follow about goals, "The term 'Goal' (sometimes called 'business concern' or 'critical success factor') refers to the overall, high-level objectives

of the system. Goals provide the motivation for a system but are often vaguely formulated. It is important to assess the value (relative to priority) and cost of goals. [54]" Goals differ from requirements as they are abstract expressions of a customer desire for a system function or performance, where requirements are a much lower level, many times describing a functionality. Goals are normally vaguely formed and are tied to cost to benefit analysis and document stakeholder intention refining the system vision into objectives to be fulfilled [86].

Goals are commonly associated to scenarios in requirements management [86]. Pohl states scenarios "describe concrete system interactions and hence enable the stakeholders to describe concrete examples of satisfying the identified goals" [86]. Goals describe what the stakeholders want at a and scenarios describe situations in which the goal is used or needed. Goals and scenarios also work very well together to assist in the development of new requirements as stakeholders detail what the system "should" do [86]. Goal traceability and relationships are shown in Figure 9: Goals - Traceability and Relationships.

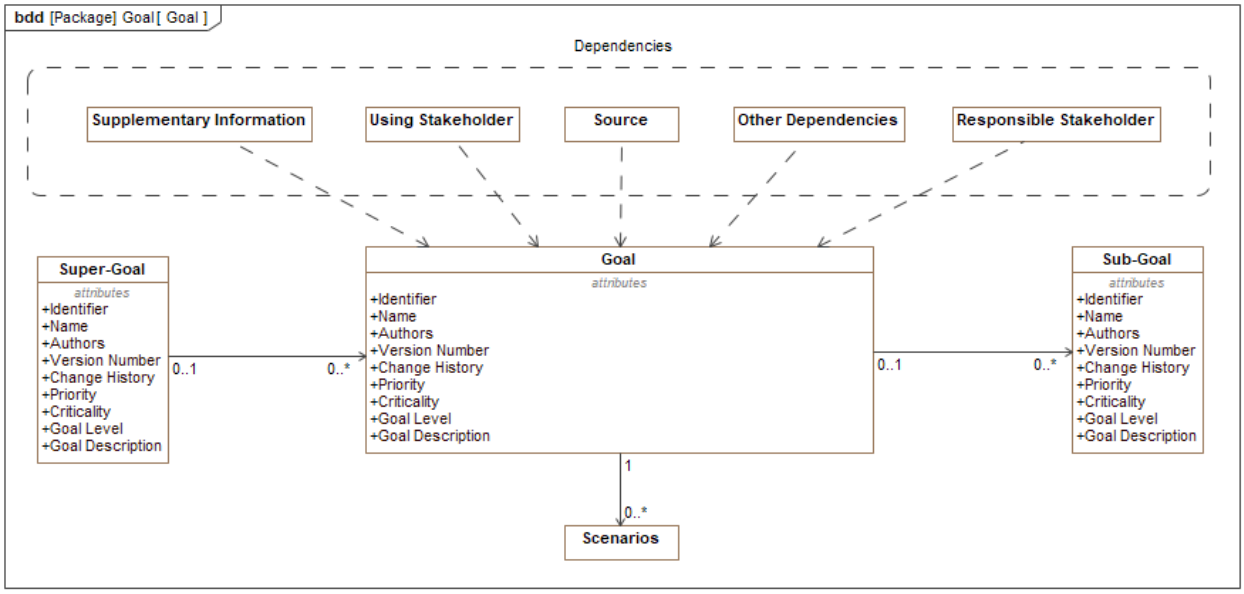

Figure 9: Goals - Traceability and Relationships

User Stories as Goals

User Stories are not detailed enough to be considered true requirements, as they do not have all the metadata associated with an INCOSE requirements management style requirement. The simple structure of the Connextra User Story format prevents a full understanding of the domain and system function and relations to be drawn from a User Story [87]. User Stories are realistically goals and not requirements. In the Connextra format, the word goal is used vice requirement [18], further reinforcing this. The ambiguity of User Stories, therefore, aligns with goals versus requirements.

As User Stories are not detailed enough to be considered true requirements, documentation using solely ambiguous User Stories creates problems in understanding the proposed system domain and functionality. The problems, if left undetected, propagate throughout the Agile software development process and documentation, which leads directly to a result of negative affect on the system in development [88]. By not detailing requirements at the lowest level possible, but instead relying on goals, software developers' risk not fully understanding the problem domain and therefore may not architect software correctly. This leads to waste in the form of rework that can dramatically affect cost, schedule, and performance in a project [89].

Requirements Management - Conclusions

What are the interrelationships between project requirements and changes that drive project success when utilizing Agile Scrum User Stories versus systems engineering shall statements? There are many commonalities and differences between User Stories and INCOSE requirements management style requirements. INCOSE style requirements offer more engineering and technical rigor through built in robust traceability and metadata. User Stories

offer a high-level, abstract, and easy to execute system. Both forms of requirements management offer minimal traceability to user, or persona, types, needed function, and reasoning. User Stories are used in Agile software development and INCOSE style requirements are more often used in traditional, predictive projects. Agile Scrum User Stories should not be considered requirements when compared to the INCOSE definition. Instead, they are goals. They are abstract definitions of what a customer needs and wants in a system. Systems engineering shall statements are very concrete statements of what a customer needs a system to do. User Stories are too abstract to communicate customer needs while shall statements can be too rigid in management and tracking to allow for the flexibility software developers need to perform Agile development. The abstract nature of User Stories can directly cause the introduction of defects, both in requirements and in the system, through misunderstanding or miscommunication with the customer. In contrast to systems engineering shall statements, Agile User Stories do not have thorough traceability horizontally and vertically throughout the system life cycle. This creates difficulties in the performance of Verification and Validation activities, which leads to the introduction of defects. Both systems engineering shall statements and User Stories have benefits and drawbacks when applied to DoD Agile software projects and neither individually fully capture the needs of development teams with regard to executing projects.

Due to the lack of large-scale traceability and metadata linkages, User Stories alone do not meet the needs of DoD systems engineering focused, integrated software projects. INCOSE requirements management is not flexible on its own to allow ease of tracking in Agile Scrum Sprints, as requirements are considered too “low” in scope. On the basis of the above reflections on the current state of User Stories and INCOSE requirements management, it is clear that a dual system is needed. Initial requirements elicitation through INCOSE requirements management

methods should be performed to create a requirements baseline. From this initial baseline, Epics can be drawn and traced directly to individual requirements. In this way, the flexibility of User Stories can be taken advantage of while the traceability and rigorous technical documentation of INCOSE requirements management can still be employed.

CHAPTER 3 – CASE STUDIES OF CAUSES OF SOFTWARE PROJECT SUCCESS OR FAILURE

This chapter seeks to answer research question 2, which is stated here as follows:

Research Question 2: What are the systemic causes of success and failure of DoD software projects?

Each software project in the DoD is managed and executed in an individual and unique manner. Some projects follow a very rigid project management and systems engineering process, requiring formalized project documentation for every customer change, causing a large amount of effort on the project management and customer team, but leading to a very controlled development environment. There are also projects that are extremely Agile, reacting quickly to changing customer needs, but result in a very unstable development environment with increased project failure risk. It is posited that the optimal execution environment leverages the benefits of both methodologies.

Sub-question 2.1: What are the systemic causes of failure and successes in Agile Scrum projects?

- *Task 2.1.1: Perform a diagnostic literature review of journals, conference proceedings, textbooks, and professional opinion articles to determine what the software development industry believes causes Agile Scrum projects to fail.*

  Diagnostic literature review complete.

Sub-question 2.2: What do experts believe are the advantages and disadvantages with Agile Scrum execution?

- *Task 2.2.1: Perform a diagnostic literature review of journals, conference proceedings, textbooks, and professional opinion articles to determine what the experts in the field of Agile Scrum software development believe are the weaknesses in the Agile Scrum framework.*

Sub-question 2.3: How have these failures and successes been demonstrated in DoD Agile Scrum software projects?

- *Task 2.3.1: Research and document the successes and failures of the DoD Qualification Management System (DQMS) system project utilizing historical project records and staff interviews.*

  Researched and documented the successes and failures of the DQMS project, viewing approximately 15 years' wort of data through interviews, project execution data, and system and project documentation. The results of this research are in the following text entitled "Case Study 4 – How Too Much Rigor can Stifle Innovation as Shown in the DoD Qualification Management System (DQMS) System".

- *Task 2.3.2: Research and document the successes and failures of the DoD Logistics Management System (DLMS) project utilizing historical project records and staff interviews.*

  Researched and documented the successes and failures of the DLMS project, viewing approximately 10 years' worth of data through interviews, project execution data, and both historical and current customer satisfaction reports. The results of this

research are in the following text entitled “Case Study 2 – Implementing Systems Engineering Rigor in the DoD Logistics Management System (DLMS) Project”.

- *Task 2.3.3: Research and document the successes and failures of the DoD Data Exchange Software System project utilizing historical project records and staff interviews.*

  Researched and documented the successes and failures of the DoD Data Exchange Software System as pertaining to the implementation of systems engineering methodologies over the course of one year. The results of this research are in the following text entitled “Case Study 1 – Implementing Structured Requirements in the DoD Data Exchange Software System”.

- *Task 2.3.4: Research and document the successes and failures of the DoD Process Automation System (DPAS) project utilizing historical project records and staff interviews.*

  Researched and documented the successes and failures of the DPAS project, viewing approximately 15 years’ worth of data through interviews, project execution data, and both historical and current customer satisfaction reports. The results of this research are in the following text entitled “Case Study 3 – Project History of the DoD Process Automation System (DPAS) 3.0 Client/Server to Web Application Server Conversion Project”.

- *Task 2.3.5: Research and document the successes and failures of the DoD Work Management System (DWMS) project utilizing historical project records and staff interviews.*

Researched and documented the successes and failures of the DWMS project, viewing approximately 20 years' worth of data through interviews, project execution data, and both historical and current customer satisfaction reports. The results of this research are in the following text entitled "Case Study 5 – Adapting the DoD Work Management System (DWMS) to Modern Software and Systems Engineering Practices".

- *Task 2.3.6: Research and document the successes and failures of the DoD Tactical Support System (DTSS) project utilizing historical project records and staff interviews.*

  Did not perform this case study as common trends were observed in the five other case studies.

Case Study 1 – Implementing Structured Requirements in the DoD Data Exchange Software System (DESS)

Background on the DESS System

The DoD Data Exchange Software System (DESS) is a data transfer system owned by Department of Defense. It is also utilized to host various other systems. Two teams perform sustainment of the system, a software development team and an information technology and cybersecurity team. A Branch Manager performs resource management for the teams with the project management performed by a Project Lead and the technical processes, policy, and system owned by a Systems Engineer.

Due to personnel issues, the branch responsible for the sustainment of the DESS system had to go without the oversight of a Systems Engineer for over a year. Without technical oversight, the duties were split up amongst other team members, primarily the Branch Head and

Project Lead. Neither the Branch Head nor the Project Lead had formal training in systems engineering and struggled to build a good path forward for the project. Furthermore, the Project Lead had a background in IT and networking, with little to no experience in software development. This left the software team with no technical oversight. Due to the Wild West nature of this environment, most of the technical decisions fell upon the Software Development Lead, who struggled to plan, execute, and maintain system development goals while also attempting to train junior personnel. Most of the Software Development Lead's time was spent putting out fires as he was the only person who understood all aspects of the system. This further aggravated the situation as employees who should have been cross trained in the various subsystems, along with related status and development efforts, were required to pick up what they could from the little time the Software Development Lead had left in his day that could be allocated to them. In short, the DESS team had pockets of knowledge spread throughout the team, with no knowledge sharing occurring, high personnel turnover, and no overarching technical guidance. Due to these issues, the DESS team had not successfully released a major system update in over two years, with the last release occurring in March 2020. In March 2022, a Systems Engineer from another division in Keyport was rotated into the team temporarily to assist in solving the inability of the DESS team to successfully publish major releases and increase the technical rigor of the team.

Prior to the implementation of requirements management, the system was developed based upon verbal communication from the Program Office and through work prioritization as determined by the Software Development Lead. The team utilized a loose Agile Scrum methodology with work planned out on a Sprint-to-Sprint basis and rarely any further. There was no release plan in place, nor a test plan. The development team worked on individual projects

and the customer needs were met in a “who yells the loudest” fashion with special consideration given to Program Office requests. This led to an environment where large periods of time with no system updates were performed and in which the development team struggled to gain a cohesive vision of prioritized customer needs.

Testing in this environment was not thorough. With no requirements to test to, the development team relied on a procedure referred to in software development as a “smoke test” [90]. The smoke test listed out the areas of the DESS program and instructed the user to perform various data entry and form validation. It did not cover all areas of interest to the system and had poor coverage, allowing defects to be introduced as thorough testing was not taking place [90]. Normally, a smoke test is used by a software developer prior to a software push to a repository to ensure that the code being pushed does not affect systems stability [90]. It is a preliminary test and not to be considered a thorough test plan or utilized as a final testing solution with regards to verification of a system. Furthermore, with no test plan in place, the team found it exceedingly difficult to coordinate the complex production testing required for the environment the DESS system operated in. This environment included shore facilities on both coasts, as well as out of country, afloat vessels both in home port and docked outside of the continental United States (OCONUS), and internally at the Keyport facility. As is clear, the testing environment was complex and adequate testing could not be accomplished in a single document smoke test as the team was attempting to do.

The lack of requirements led to the misunderstanding of potential changes to the DESS system. The change control board often argued over whether a task was a defect or new functionality. This made classification of work items essentially impossible to properly determine. All knowledge of what functionalities the system should have, along with the why it

does the things it does and how it should perform those things, rested in tribal knowledge held by members of both management and the development team. This created a dynamic in the team where debates would commonly occur on whether a functionality was a defect and how the system should function in a given scenario. Most of the time, decisions were not made based upon noted customer needs, but rather the opinions of senior personnel. More junior personnel and those new to the teams were hard pressed to make arguments against said functionalities as without requirements documented, everything was pure guesswork and supposition. Furthermore, non-software development members of the change review board had truly little, if any, training in the difference between a defect and a change request, creating great confusion amongst what should have been a team with a full understanding of the development process.

Implementing Requirements Management

The first steps taken to move DESS into a systems engineering focused development methodology was to implement requirements management. A survey of the branch, starting with team leads and continuing down the ladder to the floor level technical personnel, determined that an extremely small percentage of the team had exposure to systems engineering style requirements management. There were hurdles to overcome, as it is common for software developers to push back against any method viewed as anti-Agile and a strong fear that they would be forced to transition to a Predictive, or Waterfall, methodology [18]. The customer was not fully available for requirements generation and did not have knowledge that requirements were not being tracked. There was no requirements repository in place.

During the survey of the DESS team, it became readily apparent that there had been little exposure to systems engineering practices, let alone requirements management methodologies, by members of the team. Working closely with the Branch Head, time was set aside for training

for the branch in requirements management. The team leaders were trained first with the plan to have the other teams trained after the leads had a good understanding of basic requirements engineering. As none of the team were engineers, this was the first exposure to requirements management methodologies that many had. As is expected, it was a struggle at first, but the team readily accepted the changes, and a Requirements Management Board (RMB) was stood up. The membership was to be the same members as the Government Leadership Team, including the Branch Head, Project Lead, Systems Engineer, Software Development Lead, Logistics Lead, Implementation Lead, and the Technical Writer. It was determined that the software was the first area that requirements needed to be generated for, and the Software Development Lead was tasked with working with the Systems Engineer to make it happen.

After the leadership team was trained, other teams were then introduced to requirements management, with most accepting the changes, though there was some pushback. The software team in particular was not very open to change, having a fear of requirements management stemming from previous experiences in Waterfall style development. The major concern of the team was that if requirements were written down, there would be no flexibility in the development of the software, directly leading to a static development environment. The team wanted to still be able to continue their Agile Scrum style development, changing course as new needs arose. Careful consideration was given to their feelings, with one-on-one meetings held with developers that requested it and concerns addressed. By working closely with the development team, addressing their concerns, and hearing them out, the team was able to come to a point where they were open to accepting a change to a more rigorous requirements management methodology.

As no requirements repository was in place, the Systems Engineer was tasked with setting one up. The DESS team already utilized JIRA for project work tracking, so it was an easy choice to implement Requirements for JIRA (R4J) as the requirements repository. R4J is a plugin for JIRA that allows teams to create a top-down break down style tree of requirements, with full traceability to work items and test cases, along with obligatory parent-child relationships [91], as shown in Figure 10: Example R4J Project . It also allows for baselining of requirements and had a low entry point of learning how to use it as it functioned similarly to JIRA. R4J does lack in the ability to integrate with other systems modeling tools, such as CAMEO System Modeler, but the RMB decided that having the capability to link requirements directly to the existing JIRA work items was a much higher priority, and decided to accept the lower functionality.

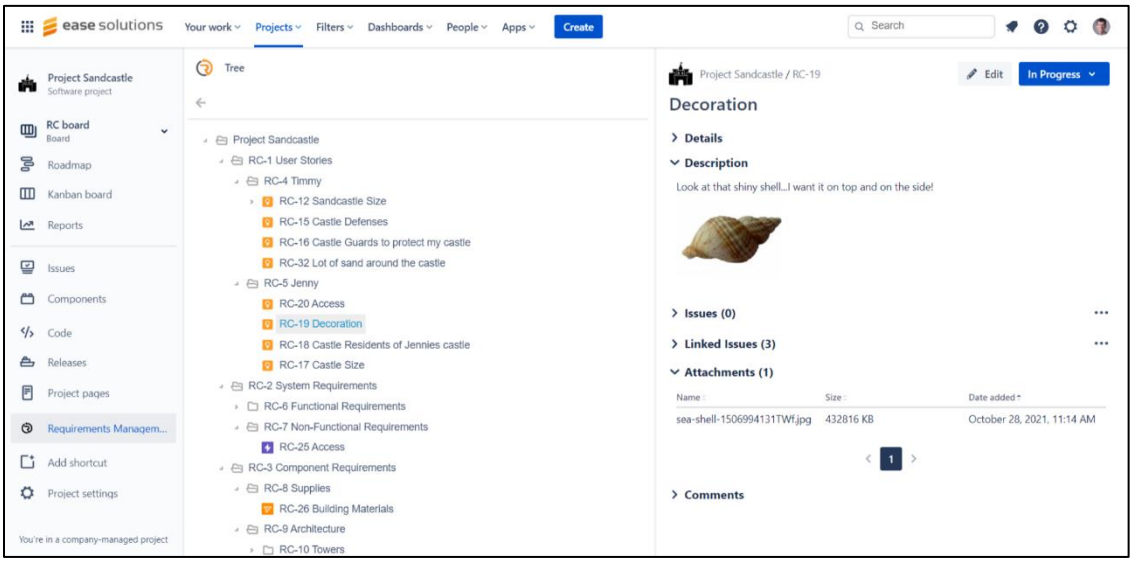

Figure 10: Example R4J Project [91]

The software team was trained in systems engineering style requirements management. This included the utilization of “shall” statements and the importance of writing requirements that follow standard requirements practice as follows: “The subject shall do something under

conditions within constraints/tolerances" [19]. The software development team struggled with the concrete nature of shall statements at first, oftentimes falling back on writing more vague requirements in a way similar to User Stories. User Stories are a way of writing software requirements in an abstract manner and this team used the common method of the Connextra in the form of "As a {role}, I want {goal}, [so that{benefit}]" [18]. While User Stories are a fine tool for developing software, in a DoD environment where it is increasingly difficult to pivot and change system connections and needs, especially when dealing with the Submarine community, having rigidity in requirements definition is a positive and not a negative. Utilizing User Stories alone leaves too much interpretation of customer needs for the developer. By transitioning from User Stories to Shall statements, the development team was able to start documenting the functionality of the system.

The development team started requirements documentation by working on each page of the DESS web application and subsystem separately. Starting with simple functionality, such as the login page, the team started to capture requirements and document them in a repository. For each page, a high-level requirement was generated. Children were populated under the high-level requirement and so forth, following standard requirements engineering practices [11]. As the team generated requirements, they would periodically return to previous requirements, refining them, as necessary. The software developers quickly grew in confidence, learning what should be documented and what should not. The Systems Engineer functioned as a mentor and guide through the process, allowing the team to lead the generation and writing of requirements and stepping in only when needed to course correct or when advice was asked for.

As requirements were generated, change requests and defects started to be traced back to the requirements. The Configuration Management Plan (CMP) was revised to ensure that any

new change request or defect must have an associated requirement, or it would not be worked on. A new workflow was devised and implemented, adding the RMB to the work approval process, as shown in Figure 11: DESS Change Request Flow Process. Through the implementation of requirements traceability, the leadership team was able to quickly determine the difference between a defect and a change request, something that the board struggled with prior to the more structured requirements management policy. This directly improved the DESS team's capability of identifying and prioritizing defects and change requests.

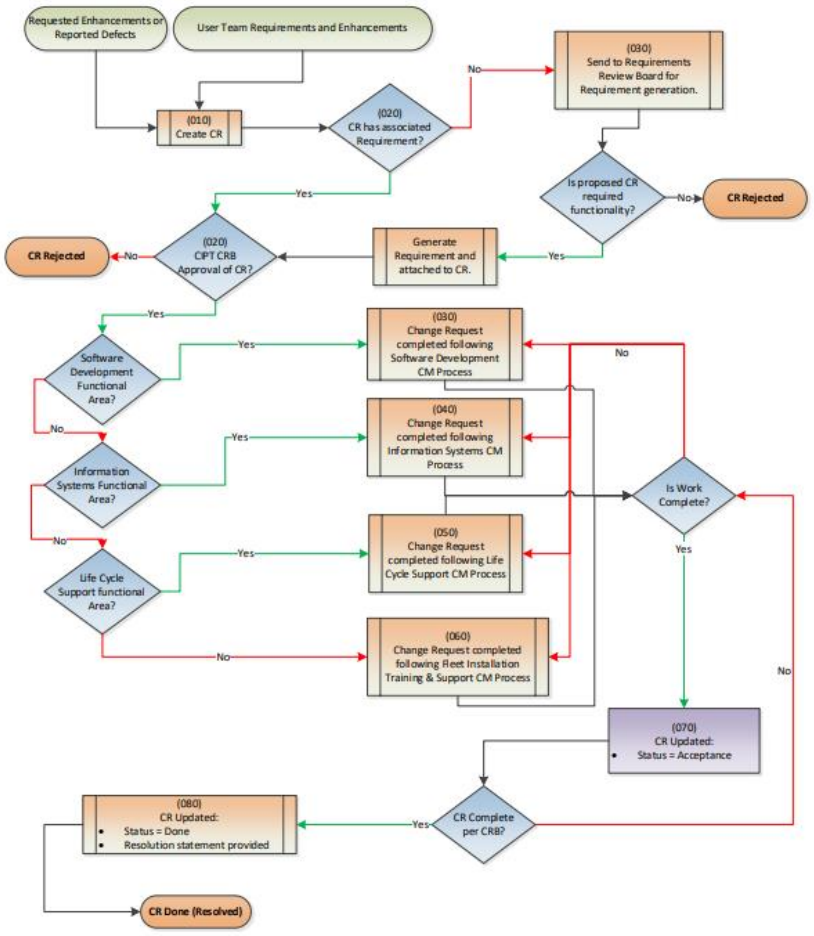

Figure 11: DESS Change Request Flow Process

Within nine months, the software team had completed enough requirements to perform an initial baseline. A method of determining the difference between new, approved, and depreciated requirements was needed. The team decided that new requirements were added with a (New) metatag until approved, at which point the metatag was removed. Approved requirements needed no changes. Depreciated requirements were marked with an (OBE) metatag, which stands for "Overcome by Events," a common term used in the DoD for depreciated or no longer needed items. This simple policy change allowed for the team to more easily search the R4J repository, writing JQL queries to filter on those tags, or the lack thereof. It also allowed new change requests to be tied back to requirements, facilitating communication between the RMB and the CRB that a requirement had been generated for the change, but had not been approved yet. This streamlined the process for the software team, giving them the power to document potential upcoming work without having to utilize a separate system to do so.

Implementing Test Management

As requirements were being developed, there was still the need for a formalized test management plan. As stated earlier, the DESS team had not successfully released an updated version in over two years. The sponsoring Program Office was not happy with this situation, as major changes to the system were tied to the updated version. Holding and passing a Test Readiness Review (TRR) was of paramount concern to the Systems Engineer and the team as a whole. In the DoD, a TRR "provides the formal approval authority with a review showing that the system is ready to enter the test and that the funding and execution of a test executes the test and gathers the required information" [92]. A successful TRR is required prior to a system completing formalized testing and being deemed ready for a Production Readiness Review (PRR).

The first steps taken by the Branch Manager and the Systems Engineer were to formally identify one of the development team members as a Test Engineer. While the person placed in the Test Engineer position did not have previous experience as a system tester, he did have a background and degree in Computer Science. To rectify the lack of knowledge, the Systems Engineer dedicated the first two months of the Test Engineer's time to training in systems test and evaluation (T&E). Time was set up with the DoD Logistics Management System (DLMS) Lead Test Engineer to assist in training the new tester, creating a mentor / mentee relationship. Once the training was complete, the Test Engineer had a much more solid foundation of T&E to work with.

As one of the key components of a test plan is having test cases, the team set about breaking the smoke test up into individual test cases. These cases were sorted, organized, and stored in the branch's Confluence site. Confluence allows files to show historical data, along with any changes, creating a configuration control system for test cases. This allowed tracking and security of the test cases, rather than storing them in a shared drive. As the smoke test was broken up, coverage was assessed. Where holes in coverage were noted, new test cases were created. This process continued until the DESS system test coverage was at 100%.

The Test Engineer would review requirements as they were posted in the R4J system. If a requirement represented new functionality that was not previously accounted for in the test cases, a new case would be written and documented. The Test Engineer would remove functionality from the test cases if the requirement were for an OBE function. This created a system that accounted for changes in requirements, directly saving time and money for the team by ensuring the test plan was kept up to date.

As formal test cases were being developed, the system still required a major release to be executed. To facilitate the DESS 4.1 release that had been in the queue for two years, the Systems Engineer devised a plan to ensure all discrete system types were tested. This required facilitation of testing aboard afloat commands, both CONUS and OCONUS, as well as various shore facilities. It was determined that to have full coverage for the fleet, afloat assets would be identified with connections both via the Navy & Marine Corps Intranet (NMCI) and through non-NMCI connectivity. Special case afloat assets would also have to be tested, but only through a pier connection. The Type Commander (TYCOM), crew offices and Trident Training Facilities and various other local assets would also have to participate in testing.

As each platform utilizing DESS had different requirements associated with them, a test matrix was devised that showed test coverage. The smoke test was still available, along with the DESS User Guide. The User Guide detailed step by step instructions to perform specific functionalities. It was utilized by Sailors to perform their duties and to learn DESS. As the Sailors were familiar with the User Guide, it was determined that only internal testing would utilize the new test cases and external testing would be performed following steps in the User Guide. This was communicated and approved through the Program Office and TYCOM. A matrix was devised that tracked what and how each platform would test, as shown in Figure 12: DESS v4.1 Test Matrix.

Test Plan Release Version: 4.1.1.0 Date: 5/10/2022 Matrix Version: 1.0		Tester Group				Total Tests Required/Passed
		Fleet Testers	Fleet Testers (Special)	TYCOM	Local	
	Required:	7	6	17	24	54
User Guide						
4.1.1	Afloat to Ashore Transfer Process: Download the DTU	X				
4.1.1	Data Transfer: Back Up Data – Afloat	X				
4.1.2	Data Transfer: Restore Data – Ashore	X				
S2DE Production Testing						
	LAN Admin Functionalities					
	Login			X	X	
	Home Page			X	X	
	LAN Admin Landing Page			X	X	
	Backup Process			X	X	
	Restore Process			X	X	
	SRT Upload			X	X	
	Account Management			X	X	
	Download/Upload			X	X	
	User Functionality					
	Login			X	X	
	Home Page			X	X	
	Third Party Application Access			X	X	
	Super Administor Functionalities					
	Login				X	
	Home Page				X	
	Super Administrator Dashboard				X	
	Configurations				X	
	Crew Management				X	
	Link Configurations				X	
	Announcements				X	
	Third Party Application Access					
	Afloat Readiness Tracker	X	X	X	X	
	Afloat Onboard Training			X	X	
				X	X	
		X	X	X	X	
	(IETM)	X	X	X	X	
	Downloads			X	X	
Fleet Access Test						
	LOCAL AREA ADMINSTRATOR ACCESS					
	Login and Landing Page		X			
	LAN Admin View		X			
	User Access					
	Login and Landing Page		X			
	MRC - SRT Upload					
	SRT Upload	X				

Figure 12: DESS v4.1 Test Matrix

By June 2022, the team had gotten approval to pass through the TRR. They proceeded to conduct a full systems test, successfully ensuring that all parties performed systems testing as needed. Prior to publication of the new version, the DESS system had to successfully gain approval for release to production at a PRR. For DoD programs, a PRR "determines whether the system design is ready for production, and whether the developer has accomplished adequate production planning for entering Low-Rate Initial Production (LRIP) and Full-Rate Production (FRP)" [93]. As of In July 2022, DESS v4.1.1 was able to successfully hold and pass a PRR, gaining approval from the Program Office for distribution to the fleet. Gold disc production was scheduled, as well as the concurrent release of an updated User Guide.

Results of Policy Changes in DESS

Applying requirements management principles and formalized test management to the DESS system resulted in increased efficiency and team cohesiveness. Through the utilization of requirements management versus loose User Story tracking, the development team increased their knowledge of what the system was required to do and how it should function. This directly created an environment where a formal test plan could be created and tracked. Furthermore, the difference between change requests and defects were solidified. Utilizing R4J allowed the team to trace all changes and defects directly to requirements, easing the ability to note new functionality from existing.

Of special note is the working environment and stress levels of the software development team. Prior to the implementation of the requirements management and formal test planning, the team often reported high stress levels and frequent confusion on what needed to be worked on. Few had a full understanding of what the full system should do and how it should behave in most scenarios. Knowledge was very pocketed with most knowledge transfer occurring on the job and

large swaths of information held in tribal knowledge. After the changes to process and policy were enacted, the team frequently reported higher job satisfaction, less stress, and increased understanding of the system as a whole. While this is just one team, it is a good example of how bringing systems engineering methodologies into Agile Scrum environments can be greatly beneficial to programs.

Case Study 2 – Implementing Systems Engineering Rigor in the DoD Logistics Management System (DLMS) Project

Introduction

Many software projects in the Department of Defense (DoD) lack the benefits of established systems engineering methodologies and good engineering rigor. Software developers are trained in the use of the various Agile frameworks and have extraordinarily little exposure to systems engineering methodologies. It is common for software developers to be strongly opposed to any development methodology that can be viewed as infringing on the principles of the Agile Manifesto [7] [94]. Only large-scale DoD software projects executed under the direction of the DoD Instruction 5000.02, Operation of the Adaptive Acquisition Framework, are required to follow rigorous systems engineering methods [31]. Due to this, many smaller projects in the DoD have little engineering rigor and frequently fail to meet customer needs and expectations. The DoD Logistics Management System (DLMS) is one such project. The following text details the transition of the DLMS project development from no engineering rigor to one that merges systems engineering methodologies with Agile Scrum principles. By selectively increasing engineering rigor through merging systems engineering principles while maintaining and leveraging the flexibility of the Agile Scrum Framework in the DLMS project,

the project lowered risks associated with cost and schedule while increasing customer collaboration and communication.

Background on the DLMS Project

The DoD Logistics Management System is a Diminishing Manufacturing Sources and Material Shortages (DMSMS) [95] management system that facilitates management of obsolescence of equipment and parts which is developed, maintained, and utilized by the Department of Defense. The DLMS system is used to determine mitigation strategies, manage military platforms, such as ships, aircraft, and land vehicles, and perform logistics and lifecycle planning. The DLMS system started as a small system and has grown into a large, multifunctional system. This growth took place with very little thought of systems engineering, engineering rigor, or technical rigor.

DLMS, created in the early 2010's, evolved from a spreadsheet-based system into a web application. As the spreadsheet system was used, it was quickly determined that a centralized data repository in a web-based system was necessary. A software development firm was contracted to transition the spreadsheet-based system into a web application, which was then turned over to the DoD for sustainment. A small software development team consisting of one government and one contractor was created to maintain the system and develop new capabilities.

Though small, the development team was made up of senior programmers that had been developing software since the late eighties and early nineties and the team utilized their own form of project management. Neither developer had a formal education in software development, having been fully self-taught during the dot-com bubble. The team used a SharePoint system to track proposed changes and defects. Changes requests were referred to as "enhancements" and written in the form of User Stories, a practice that originated in Extreme Programming practices.

User Stories are normally written utilizing the Connextra template, which is as follows: “As a {role}, I want {goal}, [so that{benefit}]” [18]. Both defects and enhancements were prioritized using a high, medium, and low priority system, though this system was not clearly defined. The team did not plan work based upon Sprints or utilize a work tracking system, such as a Kanban board. Each team member pulled work from the backlog that they felt needed to be done and accomplished it at their own pace. There was no plan for code publishes to the production system and testing was performed through a large “smoke test.” A smoke test is “a test suite that covers the main functionality of a component or system to determine if it works properly” [96]. The DLMS smoke test was a large test case that walked through the core functionality of the system but did not cover all edge cases nor was it designed to test individual sub-systems or functions. Engineering rigor was low to non-existent, and the project struggled to meet Capability Maturity Model Integration (CMMI) [97] standards as set forth by the department. Over the course of five years, the team grew to five developers, including the addition of a computer scientist, a database administrator, and a test engineer.

The DLMS system oversight is the responsibility of a governing body called the Stakeholder Review Board (SRB). During this time, SRB membership was made up of each of the Branch Managers in the division, the Division Head, Technical Project Managers with projects associated to DLMS, and the DLMS Project Lead. The SRB acts as a steering committee, making directional decisions for the system, approving funds appropriation, and overseeing project execution.

The SRB delegated the authority to approve change requests to the DLMS system through another committee called the Enhancement Review Board (ERB), which was comprised of the SRB plus the addition of the other divisional Project Leads and DLMS power users. The

ERB had the authority to approve or deny enhancement requests. This created a close working relationship with customers where customers had a large sense of ownership in the DLMS system.

Identifying there is a Problem

At this point, numerous new functionalities were developed in the DLMS system that the user base did not desire. As the development team tracked enhancements utilizing User Stories, little detail of what the customer needed was communicated to the developers. Instead, the individual developer met with the enhancement proposer to figure out what the customer desired. This collected data was normally kept handwritten in notebooks, which were scattered about the workspace. There was no electronic record or notetaking associated with the development. This led to a system where the developer thought they understood what the customer desired but did not have a standardized way of documenting this or communicating it back to the customer. No modeling was performed. SharePoint tickets were updated but no tracking system was used.

Due to the lack of formal work tracking and communication methods, customers rarely received the changes or projects as desired. As no estimation tool was utilized, it was common for customers to hear that it would take “two weeks” to complete a change, regardless the size or complexity, so much so that it became a running joke within the customer group. Change requests routinely were delivered with functionality that was not desired by the customer or that did not quite meet the requirements of the customer needs. With no formal work tracking or system for workload estimation, developers were left to their own devices, often spending time working on pet projects or “fun little tweaks” to the system, adding new color schemes for the interface or small changes to the system that customers did not request.

At this time, projects often started without consistent User collaboration and with little understanding of the proposed system context. An example of this is the DLMS Watch List functionality. The Users desired to be able to track risk status of various parts, adding them to a watch list style function. In DoD logistics, watch lists are "a report that captures the status, mitigation plans, and trends for all high risks and those medium risks that need to be elevated to senior managers" [98]. Customers attempted to communicate this idea to the DLMS development team in a series of meetings. The team did not understand the problem domain and instead came to the assumption that the Users desired to be able to create a list of parts to be flagged for follow up on a scheduled basis. A function in the system was developed allowing users to mark parts with a flag, a recurrence period, and criticality. These parts were then viewed in two methods. The first was as a pop-up reminder upon logging into the DLMS system. The second was through a menu item displaying a list of all parts with their associated watch list meta data. During the development of the DLMS Watch List, Users were not consulted past the initial meetings nor were any prototypes demonstrated. When the functionality was finally presented to the requesting customers, one User stated, "I could see how this could be useful, but this isn't anywhere what I had in mind." Due to a failure in communication resulting in a misunderstanding of the basic needs of the customer, a subsystem was designed and implemented into the DLMS system that was not desired. It took another year of tweaks in functionality through User input to make the watch list perform the developers intended function. Users never received the actual logistics watch list functionality they desired.

Within six months of joining the team, it was clear that some form of software development rigor was required. Customer dissatisfaction was high, and projects were not delivered within cost or schedule. A system was put in place to incorporate "User Design

Teams" (UDTs) into the development process. UDTs consisted of users of the DLMS system representing each of the branches of the division with a member of the software development team facilitating the meeting. The UDT members would sign a charter for the associated project and agree to availability for project reviews and to answer questions of the development team.

Applying the User Design Team Process

Utilizing the UDT process, the team increased the success rate of project delivery. Work items associated with projects were now tracked in a formal system, as User Stories kept in a spreadsheet. Engineering rigor was still low, as there was no modeling, no complexity or workload tracking, and little schedule projection. However, customer engagement was at an all-time high and satisfaction with the delivered software product was increasing. While the User Stories were not detailed enough to capture all the customer requirements, they were more detailed and tracked than prior to the UDT process.

Issues still occur using UDTs, mostly due to a lack of communication and understanding of customer needs. An example project where this occurred is the Supportability and Forecasting Tool (SFT) subsystem. The SFT started as a spreadsheet utilized by an obsolescence management group in the DoD. It used the built-in functionality of Excel to leverage the Poisson distribution, allowing logisticians to forecast part availability and assist in identifying risks associated with the supply chain. This was tied directly to submarine schedules, based upon hull designations. The spreadsheet had the full functionality desired by the logisticians but lacked a live connection to the DLMS database. Data was stored over multiple spreadsheets, causing issues between logisticians. Customers desired to transition spreadsheet functionality to the DLMS system, allowing for a common workspace for the SFT users.

The development team used the UDT process to develop the SFT. A UDT team consisting of users of the SFT spreadsheet was assembled and a charter was signed. The team worked together to create User Stories to represent the SFT system. The spreadsheet was given to the development team to reverse engineer. Complexity was high in the system, as discrete probability curves had to be calculated utilizing the Poisson method and the limited resources with experience in discrete mathematics. The logisticians that utilized the SFT spreadsheet did not understand the math behind what they were using, relying on common logistics knowledge to know to apply the Poisson distribution.

The project was successfully delivered on time and within budget but with a limited number of the functionalities that the users required. The limitations of User Stories and a lack of requirements traceability with no task planning led to many assumptions being made by the development team. The product looked and functioned similarly to the spreadsheet based SFT but was not user friendly. The required schedule tool was hard for users to use and had no way to automate or upload schedules, requiring users to manually input dates and hulls. Due to limits in the coding language used and the hardware which DLMS was hosted, open-source software packages were required to calculate the Poisson distribution to a level that did not cause crashes in the system. This required the developers to go line by line through the software, purging out what they felt was unnecessary. Even with the purges, the system ran much slower than the Excel spreadsheet version and took longer to input and manage data. This resulted in the customer not using the DLMS based system and instead continuing the use of the spreadsheet version. Five years later, the test team found that not only had zero users logged into the SFT system in the last two years, but that the major functionality had ceased to operate due to changes

in the code base versions. With permission from the Stakeholder Review Board, the SFT subsystem was purged from DLMS.

Transitioning to Agile Scrum

The development team continued to grow, adding four more developers. With a larger group, it was determined that the team needed to work in a more organized manner, as schedule slippages were common and there was no visibility into what each individual developer was working on. It was decided that the team would transition to utilizing the Agile Scrum Framework.

As required by Agile Scrum, one of the developers was assigned as the Scrum Master [9]. The Scrum Master's duties were to write the DLMS Scrum Handbook, manage the Product and Sprint Backlogs [9] [41], and train the development and customer teams in the Scrum methodology [9]. A large whiteboard was assigned as the Sprint Backlog for the team and an area cleared out around it so that the team could hold daily Scrums in front of the board. The Product Backlog was still managed in the same SharePoint as before, but Story Points were now assigned to work items. Story Points are not part of the core of the Agile Scrum Methodology, but instead are drawn from the eXtreme Programming method and are commonly utilized by Scrum Teams to assist in estimating workloads for Sprints and Product Increments [97]. The Scrum Master started to track Sprint Velocity and held Backlog Grooming and Sprint Planning [9] [41] meetings. Sprint Reviews [9] [41] were held with stakeholders and customers invited. The Scrum Master would facilitate Sprint Retrospectives so that the team could "inspect how the last Sprint went with regards to individuals, interactions, processes, tools, and their Definition of Done" [41].

The DLMS Project Lead was assigned as the Scrum Product Owner [9] [41] role. He took on the responsibility of representing customer interests, accepting work completed during Sprint Reviews, assisting customers during UDTs, and ensuring that the Product Backlog was well formed, managed, and the User Stories were within compliance of the Connextra [18] method. He stopped working as a front-end developer and instead focused his development time to work on research projects with the University of Washington and perform database administration and development, reducing his impact on Sprint execution.

The development team worked to bring more clarity into work items associated with Sprints. During Backlog Grooming, the team would break down large Story Point User Stories into smaller ones that they felt were more easily accomplished. Any story with a value over forty Story Points was flagged as an Epic [98] story. Once a User Story had been adequately reduced to a value that a team member could accomplish within a day or two, Tasks were associated with the Story. Tasks were written in plain English and did not follow the Connextra format. This created a system with more traceability than was present by utilizing only User Stories to develop new features and functionalities. This was accomplished through Epics being broken down into User Stories which were then broken down into Tasks. User Stories and associated Tasks were printed out on sticky notes and put on display on the Sprint Board. Status was denoted by the column in which the sticky note was present. The status columns were Backlog, In Process, Ready for Testing, and Done. Developers claiming a work item signed the sticky note, taking ownership of the work item.

By tracking Velocity and work item complexity through Story Points, the team was better able to plan how long each feature effort would take to complete. This allowed customers to make better decisions on the prioritization of software enhancements, creating a more organized

backlog with a clear structure for customer needs. The Project Lead was better able to plan out how much work could be accomplished in a fiscal year and how many developers were needed per effort. This brought more rigor into the engineering and project management efforts.

During this time, an effort was made to create a more formal and rigorous test management plan. The smoke test was broken up into larger sections, demarcating the various subsystems of DLMS. The test engineer then went into each subsystem, detailing functionalities in an "as built" manner and noting them in test cases. These test cases were then assembled into a spreadsheet to keep track of them. Prior to each production publication, the test cases were walked through and signed off by the test engineer and the development team, ensuring that if a developer had worked on the subsystem, they would not be a tester on that specific page. While the broken-out test cases were a vast improvement over the single smoke test, there was still a distinct lack of traceability to requirements, which created an environment of operating by tribal knowledge and arguments with customers over what functionalities should be present and why. Though not traced to requirements, the breakout of individual test cases from the smoke test increased the capability and confidence of the team in delivering a working product to the customer.

Moving to a Requirements Driven Development Method

The DLMS team continued to grow, transitioning to be housed in their own branch in the division. With the move to a new branch came new leadership in the form of a new Branch Manager. The Branch Manager had previous work as a systems engineer, with expectations of more engineering rigor for the DLMS project. A new team of contractors also joined the branch, performing commercial off-the-shelf (COTS) vendor survey activities. The team was now at 30 members, having grown from the previous five members over the course of a 5-year period.

With rapid team size growth, the need for structure became increasingly necessary. The previous Project Lead and Product Owner transitioned to be the Staff Engineer. The Scrum Master transitioned to the Project Lead and Product Owner position. The rest of the staff were broken into two teams, the Development Team, and the Survey Team, with a Scrum Master assigned to each one and sub-teams having assigned Technical Team Leads. With the branch structured in a more hierarchical manner, decisions were easier to manage and team members better understood individual responsibilities.

The introduction of the Staff Engineer allowed the team to start the transition to a requirements-driven software development methodology. The role of the Staff Engineer, filling the role traditionally staffed by a Systems Engineer, included ensuring "customer and stakeholder requirements definition, requirements analysis, architectural design, implementation, integration, and verification" [99]. As the Staff Engineer was close to retirement, the Product Owner pursued formal education in Systems Engineering. The Staff Engineer and Product Owner drove the team towards implementing systems engineering principles, including requirements management.

The team also used this time to implement better knowledge management and project management controls. Instances of JIRA, Confluence, and Git were stood up, with use policy documented. The Product Backlog was transferred from SharePoint to JIRA. The team used the built in JIRA functionality to manage Sprints and Sprint Planning. At first, the team used both a physical Scrum board and the digital JIRA Scrum board, but eventually moved solely to the JIRA Scrum board. While this took adaptation, it did allow for more knowledge management with regards to ticket history as developers were able to more easily annotate comments and pertinent files to the JIRA ticket items. Customers were also able to easily comment and provide

feedback while watching progress status on enhancements and defects that they submitted, giving a level of visibility into the development process that was obfuscated prior.

The ERB was transitioned to a Change Request Review Board (CRRB). Enhancements were renamed to change requests to be more in line with industry standards. The CRRB functioned very similarly to the way the ERB did, but the membership was reduced to sole users. Each customer branch was allocated two voting seats and for a CRRB meeting to be held, at least one member from each branch must be present. The COTS Survey Team Lead held a voting seat representing the Vendor Survey efforts. The Product Owner was in attendance and the Scrum Master facilitated the meeting. The CRRB took greater responsibility in the ownership of the change requests, having the authority to not only approve or turn down, but also being the only personnel allowed to submit change requests. All change requests were submitted directly through branch CRRB members, ensuring that change requests were written in a consistent manner. The CRRB also had the authority to determine the priority of the backlog of change requests, creating a "top 10" system where the change requests they wanted worked on would be marked as High priority, those that were in queue to be worked on in the future marked as Medium priority, and those that were considered too low for scheduling but to be done if the opportunity presented itself, referred to as "low hanging fruit", were marked as Low priority. By having the customer approve and prioritize proposed changes to the system, the development team directly benefited from knowing exactly what the customers wanted. The customers also had to directly weigh the pros and cons of the impact on development for each change request, leading to higher quality and more important changes being approved.

A repository in JIRA was stood up to hold requirements. The team utilized the R4J plugin, allowing for ease of traceability from a requirement to a JIRA work item, as it was

readily accessible to them and a no cost solution. Using R4J increased understanding of work items to the team members and allowed the Test Engineer to better understand the requirements and the traceability to test procedures.

For requirements approval, the team approached the Stakeholder Review Board (SRB) for direction. Being primarily made up of management, the SRB decided to delegate requirements approval to a new board, the Requirements Review Board (RRB). The membership of the RRB was initially made up of the same membership of the CRRB, as the CRRB was seen to have the most knowledgeable users in the division with regards to DMSMS and DLMS use. The RRB held the authority to approve, deny, or modify requirements for the DLMS system, with the SRB retaining veto authority and tie breaker vote if needed.

With the repository in place and the development team trained in basic requirements generation and management, the team proceeded to the task of converting functionalities present in each subsystem of DLMS to an "as built" requirements document. Twenty-three functional areas were initially identified, with more being added as the effort was executed, ending up with over fifty functional areas. Examples of the functional areas can be seen in Figure 13: Example of DLMS Functional Areas and Requirements Structure.

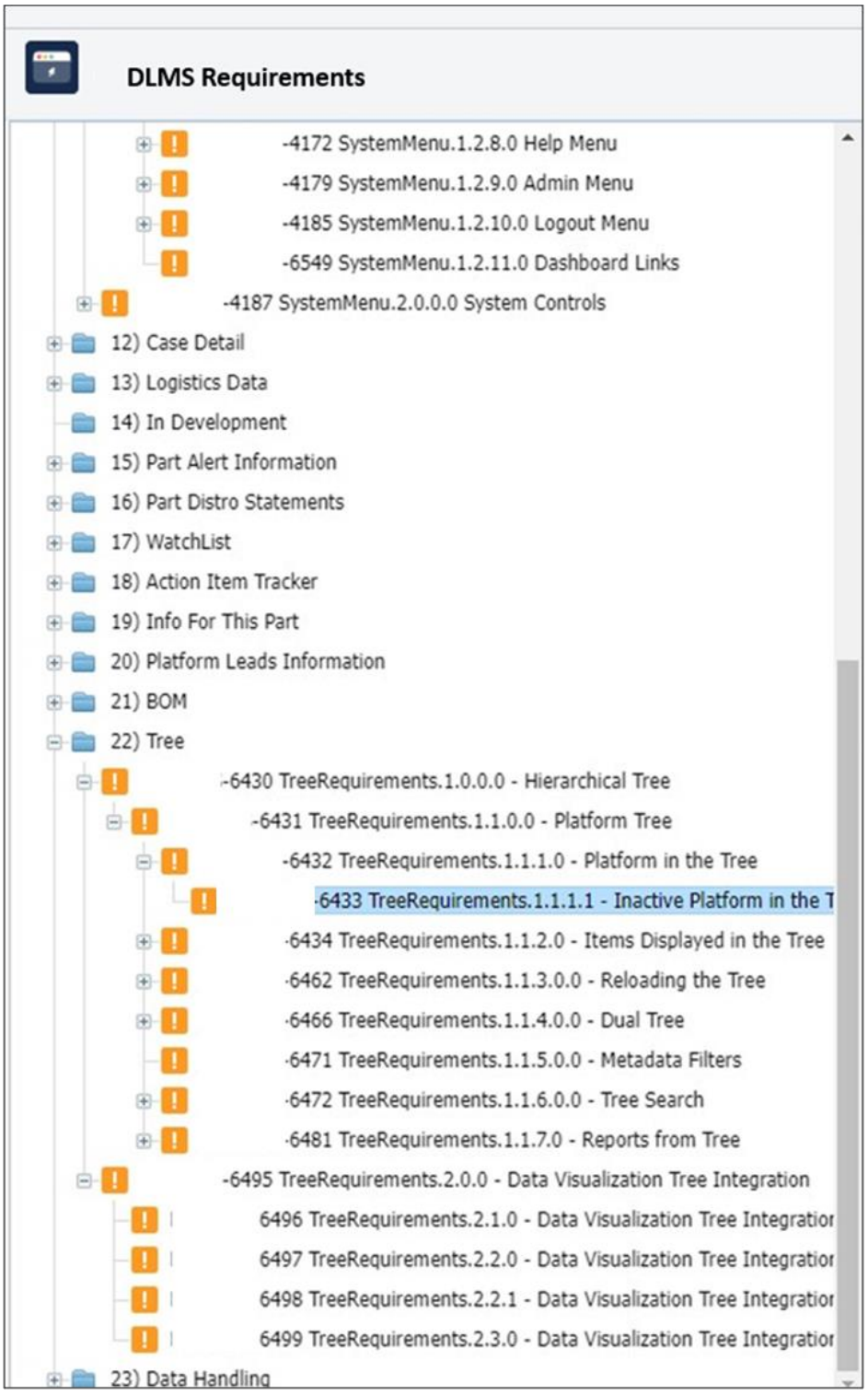

Figure 13: Example of DLMS Functional Areas and Requirements Structure

INCOSE requirements standards were followed with requirements written in systems engineering requirements management shall statement format. This was decided as most of the customers that the developers interacted with were either engineers or logisticians. Engineers are trained in shall statement style requirements management and many logisticians have at least a surface level understanding of the style. INCOSE standard shall statements are also easily understood, using a parent-child traceability and breakdown structure, which makes the format very easy to train personnel in. Customers can be trained in writing Connextra-style User Stories, but DLMS customers found that shall statements were much more familiar and easier to communicate their needs. See Figure 14: DLMS Requirements Sample for an example of how these requirements were written and stored in R4J.

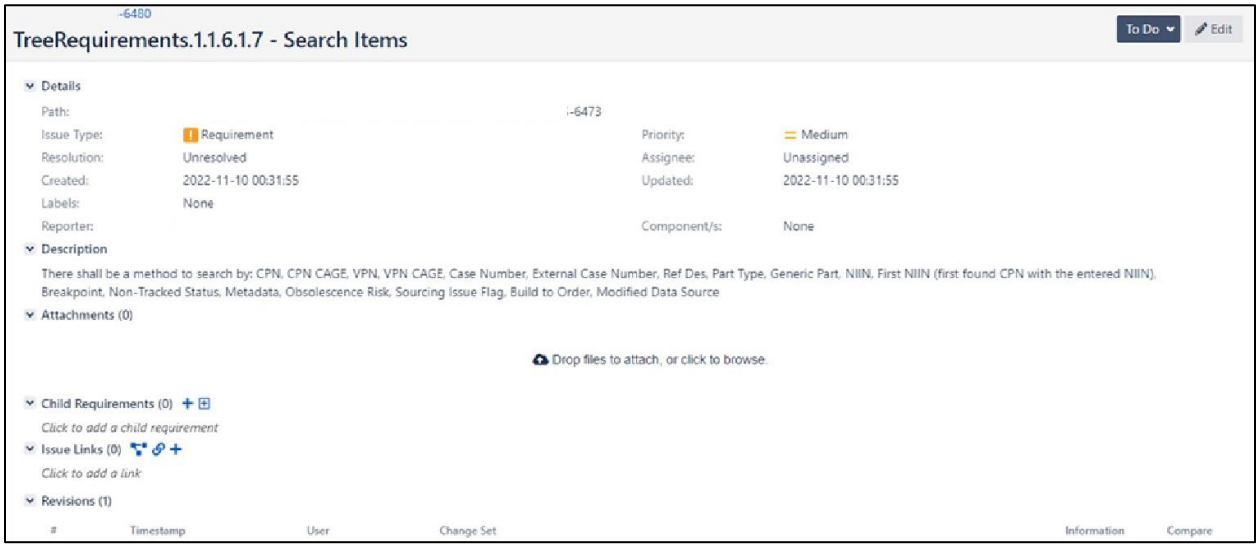

Figure 14: DLMS Requirements Sample

The team used a common spreadsheet template to document the requirements. A tracking spreadsheet was developed to note the status of each functional area with the following information tracked: Requirement Area, Requirement Numbering, Sub Requirements, Assigned To, Needs Reworked (Y/N), Ready for Review (Y/N), Initial Review Done (Y/N), Requirements

Review Board Approved (Y/N), Uploaded to R4J (Y/N), Notes, Time to Generate (in weeks), Time for Initial Review (in hours), Time for Final Review (in hours), Number of Requirements (see Figure 15: DLMS As-Built Functional Area Requirements Tracking). This gave a good insight from a project management perspective into the amount of effort required to develop the as built requirement set and also as to how much more effort would be required to complete the set and create an initial baseline.

	Requirement Area	Requirement Numbering	Sub Requirements	Assigned	Needs Rework	Ready for Review	Initial Review Done	RRB	Uploaded R4J	Notes	Time to Generate (in weeks)	Time for Initial Review (in hours)	Time for Final Review (in hours	Number of Reqs
2	Data Dashboard	DataDashboard.x.x.x		Ron	Y	y	n	n	n	Security Updates and overall rework	0	0	0	107
3	Login	Login.x.x.x		Kristi	n	y	y	y	y		0	0	0	13
4	Part Detail	PartDetail.x.x.x		Connor	n	y	y	y	y		0	2.5	0	120
5	Reports	Reports.x.x.x	Admin, Customer, Metrics, Utility	Kristi	y	n	n	n	n		6	6		400
6	RC2	RC2.x.x.x		Connor	n	y	y	n	n		0.5	0.5		25
7	Vendor Survey	VendorSurvey.x.x.x	Conduct Survey, Performance, Load Balancer, Log Viewer, Group Management	Rashad / Eric	n	y	y	y	y		0	0	0	130
8	RC1	RC1.x.x.x		Connor	n	y	y	y	y		1	1		61
9	Admin	Admin.x.x.x	Import External Data, Manage Application, Security Management, Logs, Developer Help Files, Admin Mode	Dennis	n	y	y	y	y		1	1		50
10	System Menus	SystemMenus.x.x.x	System Menus and Controls	Dennis	n	y	y	y	y		0.5	1		50
11	Security	Security.x.x.x	Data Access, Primary Roles, Secondary Roles, System Access, Meta Roles (Excluding STIGS)		n	n	n	n	n		0.5	1		50
12	Webservices	Webservices.x.x.x		Kristi Anna	y	n	n	n	n	Previously reviewed. Updated changes.				
13	Data Management	DataManagement.x.x.x	Import/Export Data, Process BOM, Train Part Type Model	Rashad	y	y	y	n	n	Security Updates	0.5	0.75		32
14	Metrics Utility	MetricsUtility.x.x.x		Rashad	y	n	n	n	n		0.5	0.75		42
15	ADHoc Reporting	ADHocReporting.x.x.x		Connor	n	y	y	y	y		0.5	1		57
16	Tree View	TreeRequirements.x.x.x	Tree Search, Tree Filter, Dual Tree View	Dennis	n	y	y	y	y		0.75	1.5		75
17	Part Supportability Information	PartSupportabilityInformation.x.x.x			n	n	n	n	n					
18	Part Alert Information	PartAlertInformation.x.x.x		Jon	n	y	y	y	y		0	0.1		10
19	Part Survey Information	PartSurveyInformation.x.x.x		Jon	n	y	y	y	y		0	0.2		14

Figure 15: DLMS As-Built Functional Area Requirements Tracking

While in the process of creating the As Built baseline requirements set, the project team still had the responsibility to continue previously scheduled projects. To accomplish this while maintaining good requirements management practices, the development team ensured that all new projects had baseline requirements generated prior to the execution of the project. This required some work to fully understand how to accomplish this, as normally requirements are very rigid. Many times, customers view requirements as something that is chiseled in stone and that should not change. Not wishing to give up too much of their Agile mindset and execution policies, the development team treated requirements sets much as they would a set of User Stories. They were prioritized and linked directly to User Stories for Story Point allocation. This allowed the team to better estimate the level of effort required to accomplish a requirement. The

team also frequently revisited requirements with the SRB and RRB for projects, adding, removing, and adjusting requirements as necessary to reflect the current state of the software development effort and customer desires. A true baseline was not snapped until the project was considered finished, final delivery was made, and the documentation closed out.

The development team members and customers responded very well to the increased engineering rigor associated with good requirements management. Customers reported increased satisfaction with the development efforts of the DLMS team. Having a clearer understanding of customer needs allowed the development team to focus on creating exactly what the customer wanted rather than focusing on non-essential but perceived desires. The test engineer found it much easier to execute his duties and he developed an increasingly mature test management plan.

Utilizing Wireframe Models to Facilitate Communication

To facilitate further communication with the customer, the development team started utilizing models during presentations in the form of rapid prototyping. Rapid prototyping is the use of creating models, drawings, or sketches to assist in determining how a User will utilize a user interface [100]. The developers took customer requirements and transitioned those into mockups of how they envisioned creating the system pages. This allowed customers to better understand what the development team planned and suggest adjustments if needed while also engaging customers earlier in the design process. Paper prototyping activities also took place where a User walked through menu options and use cases with the models. This directly increased the developers understanding of how the User would use the page. Rapid prototyping using wireframe models in this method “costs nothing in the long run” and increases developer

and customer communication [100]. Wireframes are visual models widely used to represent graphic user interfaces and system functionality in Agile development [101].

The development team found modeling to be especially useful in rapid prototyping "widgets" for the DLMS dashboard. The DLMS dashboard utilized a sandbox area where Users could choose various small widgets to display data. These widgets were designed in such a way that a software developer could create one within a few hours if given the proper data. Combining widgets together in a dashboard view allowed customers to create custom reports and dashboards without having to request a formal report through the change request process. Customers also found this approach helpful in communicating their needs and often would create their own wireframe models to present their ideas to the development team. An example of a model of a widget can be seen in Figure 16: Wireframe Model of a DLMS Widget.

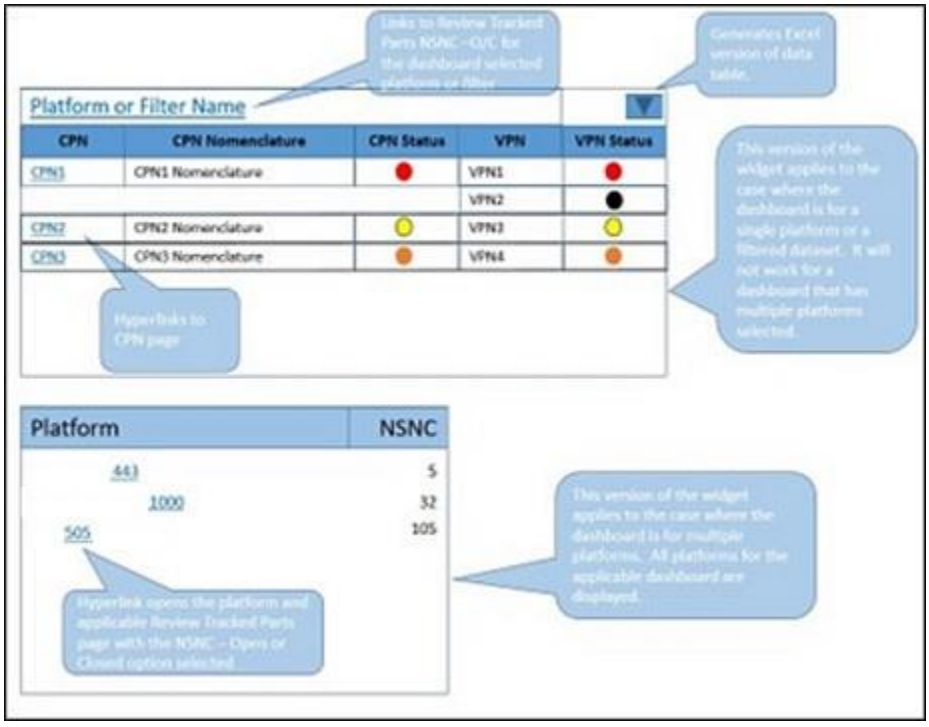

Figure 16: Wireframe Model of a DLMS Widget

While rapid prototyping activities utilizing wireframe models is not strictly a Model Based Systems Engineering (MBSE) activity, it laid the foundation for the DLMS team to start utilizing MBSE methodologies in their software development. As customers became more comfortable with the idea of seeing designs laid out in models, the team utilized other techniques, such as SYSML [102] and the Department of Defense Architectural Framework (DoDAF), to model out future projects. An example of this can be seen in Figure 17: COTS Selection Tool OV-1 Example, an OV-1 DoDAF view created for a proposed Commercial Off the Shelf Part Selection Tool. Models such as this "can visualize and animate a proposed solution in forms that various customer communities can understand and respond to" [99].

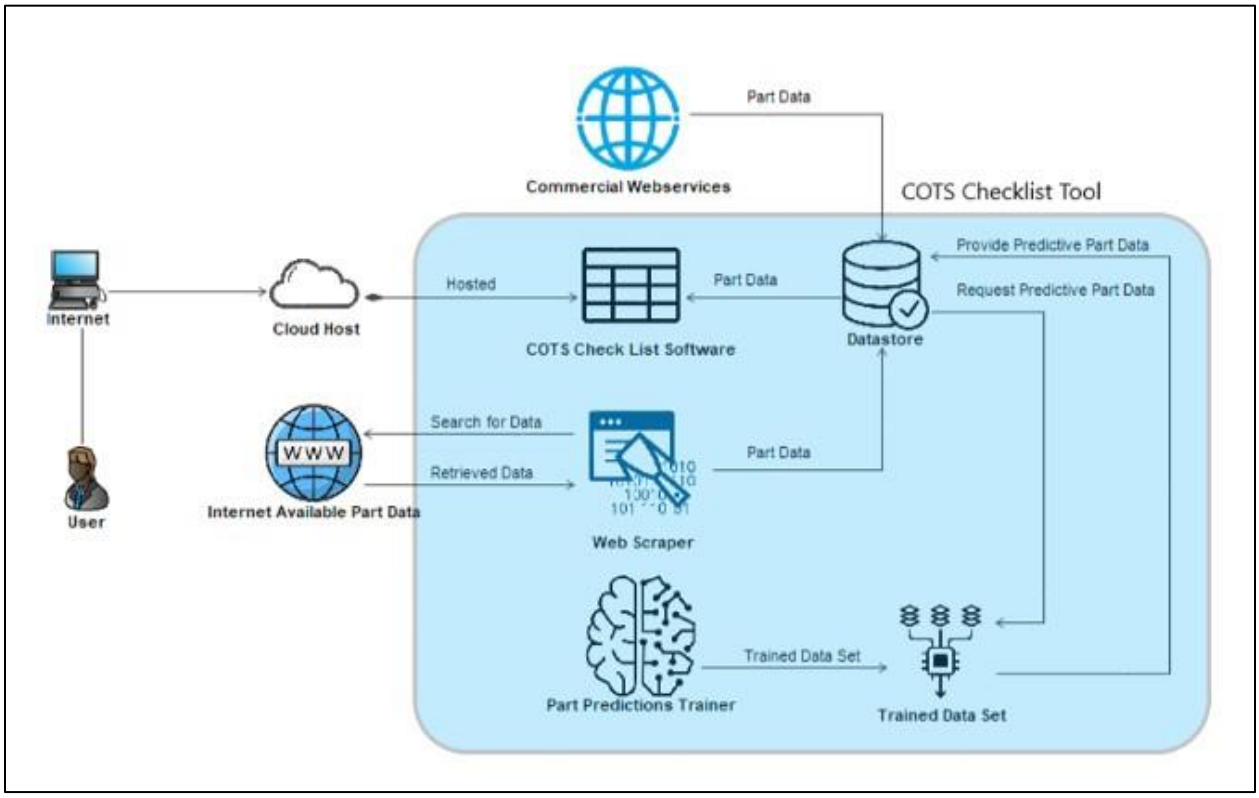

Figure 17: COTS Selection Tool OV-1 Example

Results of Implementing Systems Engineering Rigor in the DLMS System

Numerous positive changes resulted from implementing systems engineering rigor into the DLMS development lifecycle. The DLMS development team had a much easier time understanding customer needs and expectations. Software defects have been reduced in the project and test personnel have a clearer understanding of why and how the software should operate. Reported customer satisfaction is at an all-time high. Overall, the DLMS project is in a more improved state than it was prior to implementing the above changes.

The DLMS development team has experienced an increase in customer communication and collaboration, as well as reporting a better understanding of customer needs and expectations. Developers state that Systems Engineering Requirements Management shall statements "paint a more detailed and clearer picture of what customers want" than when the team was solely utilizing User Stories. By using MBSE techniques, such as modeling new functionality prior to coding efforts, developers can present the functionality in a rapid and cost-efficient manner to customers, gaining much needed feedback during the initial development steps, rather than waiting for Sprint Reviews to get customer feedback. The team enjoys the collaborative environment this method has created, embracing customer created wireframes and drawings as system development enablers. Utilizing a more rigorous engineering foundation has resulted in improved developer job satisfaction, a higher understanding of customer needs, and a more collaborative working environment.

Defects are costly for any system, including DLMS. Therefore, the test team reports the greatest increase in job satisfaction from implementing Systems Engineering Requirements Management policies. Having to rely on a single, large smoke test resulted in many defects being published to the DLMS production environment. Though the work was arduous to create an "As

Built" baseline of requirements, the test team reports much greater ease in tracing customer needs and system requirements to test cases, resulting in a rigorous and robust test management plan and process. Requirements are traced from inception to work item, allowing the test team to know exactly what needs to be tested, why it needs to be tested, and what the results of testing should look like. Defects in the DLMS Product Backlog have decreased by 300% in comparison to prior to implementing Requirements Management. The change to a Requirements Management centric software development method has directly increased the ability of the test team to meet test management requirements and ensure engineering rigor.

DLMS customers report a record high in satisfaction with system and project interaction. The DLMS Project Lead conducted a survey of customers in January 2024. System users and technical managers were asked a series of questions, and the responses were collated. The results of the survey are displayed in Table 1: 2024 DLMS Customer Satisfaction Survey Results.

Table 1: 2024 DLMS Customer Satisfaction Survey Results

Question	Average Result	Notes
Prefer user stories or shall statements?	Shall Statements	83% Prefer Shall Statements, 17% Prefer User Stories
Are you more or less confident in the quality of the DLMS System?	Equal/More	50% As Confident, 50% More Confident
Do you feel prototypes or wireframes have a positive impact in understanding and communicating user needs?	Yes	100% Positive Impact
In the last two years, have you felt you are finding more or less bugs/defects in the system than in years prior?	Less	100% Feel Less Bugs/Defects
Overall, how satisfied are you with the DLMS development process as opposed to years prior? (On a scale of 1-10, 1 being unsatisfied and 10 being very satisfied)	8	Average calculated based upon all respondents
Do you feel communication and development status has become more transparent or less in the last two years?	More Transparent	67% Improved Communication, 33% Equal to Previous Years

The implementation of systems engineering rigor into the DLMS software development lifecycle created a dramatic positive effect on the project. The development team has increased throughput of their development product, having increased confidence in customer needs and desires. The DLMS Product Backlog has seen a reduction in defects by a remarkable 300%. The test engineer and development team report reduced rework from customer dissatisfaction and a reduction in unmet needs. Surveyed customers have reported much higher satisfaction with the product and an increased feeling of ownership in the system, with respondents averaging a score of 8 out of 10, resulting in a "high" satisfaction rating. Overall, implementing requirements management, system modeling, and stringent test planning and management has had a positive impact on the DLMS system.

Case Study 3 – Project History of the DoD Process Automation System (DPAS) 3.0 Client/Server to Web Application Server Conversion Project

Introduction

The DoD Process Automation System is a process control system built and sustained by the Department of Defense. DPAS was not the first process control system the DoD developed and sustained. In 1986, the DoD deployed a previous inventory and document system referred to as the DoD Control System (DCS). After ten years in service, DoD leadership recognized the need for a more robust system than DCS could offer and, in 1999, fielded the DPAS 1.0 system.

DPAS was deployed as a three-tier client/server-based system, as was common at the time [103]. Each build was managed by the development team and pushed from the server to the

client. For an NMCI seat to have DPAS installed, the DPAS software system must be manually installed by a system administrator. Once the DPAS system is installed, client updates are checked for each time a user logs into the client system. If there is an update pending, the server initiates a "pull," downloads the update, installs the update, and then restarts the application, if necessary. In 2010, DPAS negotiated to include the DPAS software client on the NMCI gold disc for installation on all NMCI seats, but this required Legacy Application Deployment Readiness Activity (LADRA) testing and was phased out within a few years.

It is widespread practice in the software industry to move away from client/server-based systems and towards web-based applications. Client/server-based systems require installation on each asset that a customer wishes to use the application on. This can cause access issues if there are problems with specific assets and customers need to use "loaner" assets. As discussed earlier, clients require updates which causes a delay in use of the system whenever an update gets posted to the server as clients have to download, install, and restart the application. All processing is carried out on the client's asset and speed is directly related to the asset hardware configuration [104]. Finally, servers must be taken offline to install updates, which requires late night or weekend work from the server management team. Web-based applications do not require updates to be pushed to a client as these types of applications are accessed directly through a web-browser, such as Google Chrome or Microsoft Edge [105]. Access via a web-browser also removes the installation barrier that client/server-based applications face. Updates can be made swiftly to the application, many times without any downtime accrued by the server. Web-based application servers deliver dynamic content in much the same way that the client/server model operates [106] with the benefit of a high control of hardware performance, as the performance of the system is not restricted by the asset that the customer accesses the system from. Web-server

applications come with the added benefit of access control and dynamic presentation built into the system, allowing sites to restrict access only to authorized users and to present only the data that a customer is permitted to view [107]. With these points in mind, it is clear why the DPAS stakeholders desired to move to a web-based application versus the client/server-based architecture.

DPAS 2.0 to 3.0 Transition

In 2017, it was decided that to overcome the challenges presented with a client-based system, the DPAS system would transition to a web-based system. The development team proceeded to work on a prototype, transitioning the client/server based DPAS system to a sustainment only state. The proposal to transition as presented to the prime stakeholders of the system and approval to develop requirements was given in 2019, though prototype development began as early as 2016.

Managing a system upgrade while the current system is in place requires a heavy effort on software developers. Not only does new architecture have to be designed and deployed, but the previous obsolete client has to still be supported. This caused the development team to split into two sub teams. One senior developer was allocated to maintain the current client-based system. The remaining senior developer, journeyman developer, and new college graduate took on the effort to design the web-based system. Other team members included a Database Administrator, the Project Manager, a Customer Service Support Specialist, and a Software Tester.

The team decided to transition to an Agile Scrum project execution. This was due to the team attending an Agile Scrum training session and seeing the benefits that Scrum can bring to a development effort. The team stood up the initial Scrum Team with the Software Tester taking

on the role of Scrum Master. The software developers joined the team. It is important to note that there was not a Product Owner designated, though the Project Manager would have been the natural selection for this role. Having a Product Owner is essential to the Scrum process [9]. This is a crucial point to understand as it is one of the key drivers of the issues that the team was soon to experience.

To transition to Agile Scrum, the team had to create a repository for a Product Backlog, as is necessary for Agile Scrum development [9]. The team chose to use GitLab for their backlog management. In GitLab, teams can create and use dynamically generated issue lists, populated with work items [108]. Epics were created as GitLab epic work item types and User Stories were created as GitLab tasks. Sprints were setup as milestones in GitLab, as is standard practice [108]. The Scrum Master owned and managed the backlog. The backlog was set up and displayed in a comparable manner to Figure 18: GitLab Backlog Example.

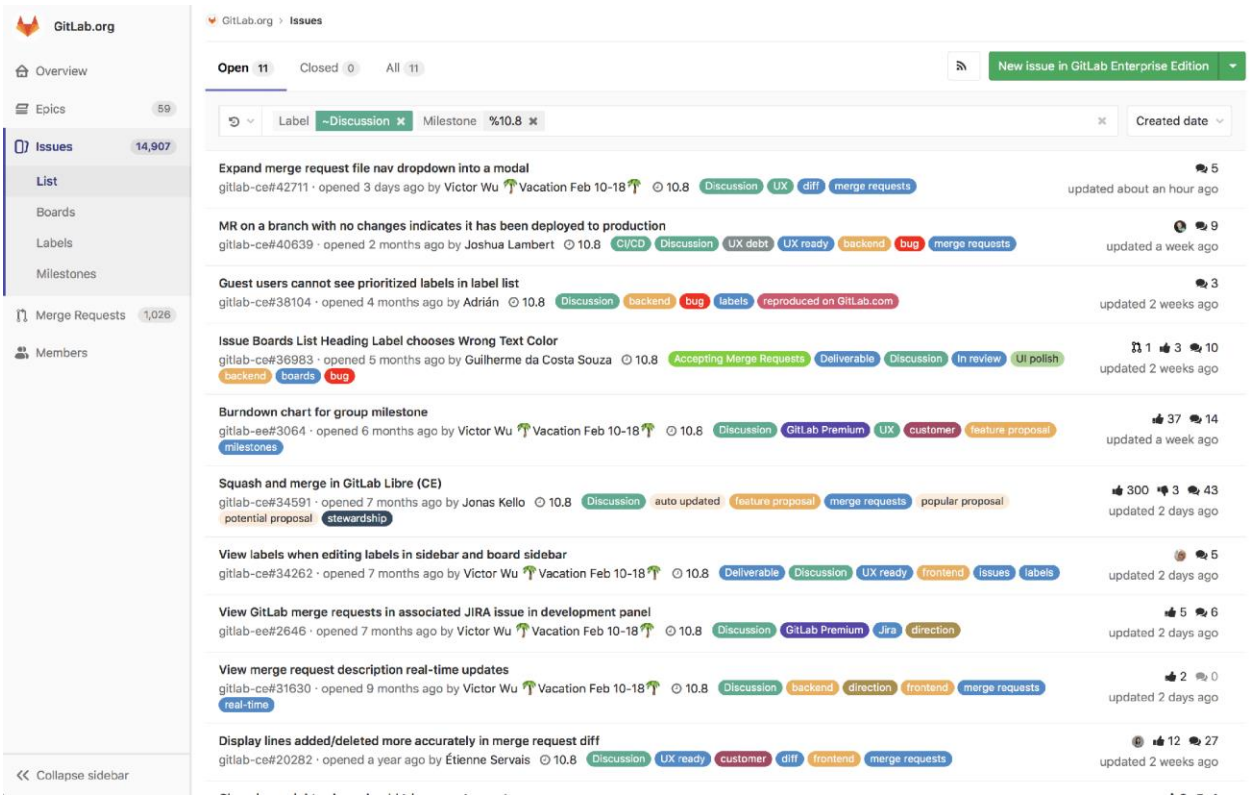

Figure 18: GitLab Backlog Example [108]

After populating the project backlog, the team began planning and executing Sprints. The backlog was prioritized according to the team's view of customer needs. This differentiates from pure Agile Scrum, as the team should have had a Product Owner that prioritized the backlog. Instead, the team used their own opinion of customer needs and desires, as well as developer needs and desires, and prioritized the backlog accordingly. At each Sprint Planning, the team took whatever work items were on the top of the prioritized backlog, counted down the backlog until they reached what could be done within their velocity, and added those items to the Sprint Backlog. By restricting themselves fully to a 100% prioritized backlog, the team lowered their capability to execute what was needed in development, instead focusing on the artificial prioritization represented in the backlog.

During the execution of the Sprints, work items were claimed by developers and moved across the Scrum Board, being marked as “In Progress”, “In Review”, and “Closed” similarly to what is displayed in Figure 19: GitLab Issue Board. A Scrum Board is an Agile Scrum project management tool that allows Scrum Teams to have a visual representation of the planned work for a Sprint to increase communication and transparency during project execution [109]. As the team was focused on performing in a self-managing method, this allowed the team to track current work item status and have transparency to the project stakeholders. The board also allowed the team to increase collaboration through ticket comments, tagging, and other matters related to the work [110]. Testing and code review was performed as work items were populated into the “In Review” column. Code review required another developer to review the code related to the ticket item for potential bugs, “cleanliness,” and adherence to policy standards. Testing activities were performed by a dedicated tester.

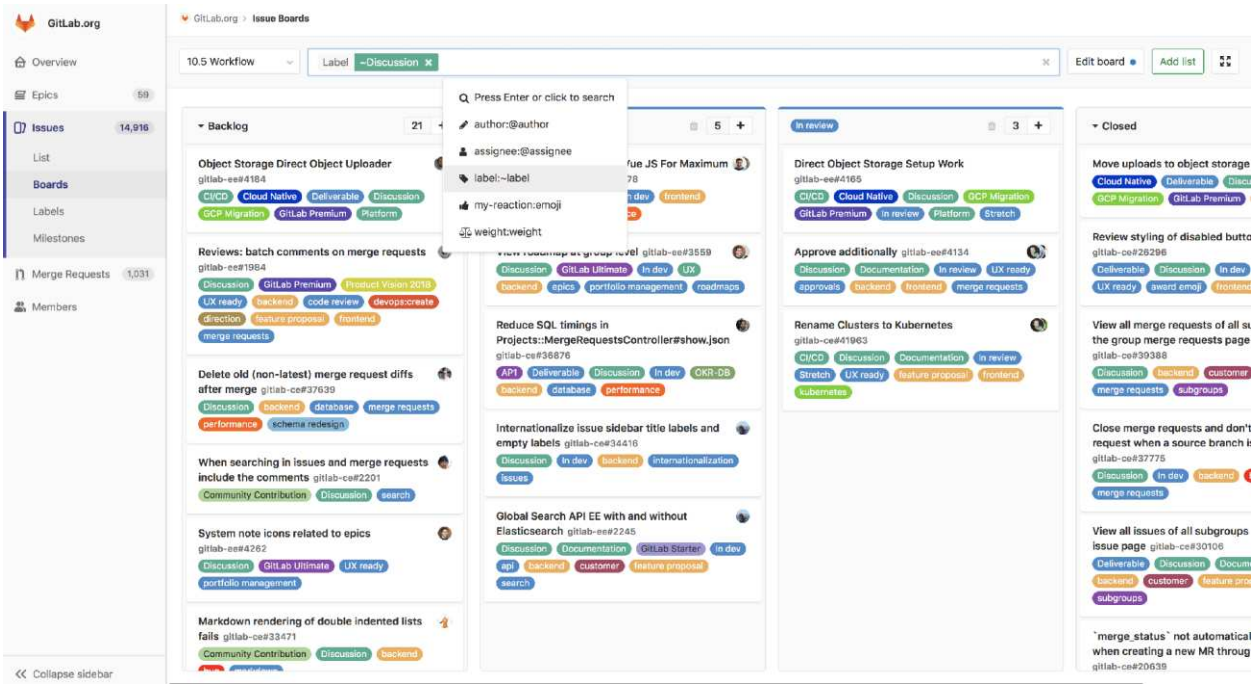

Figure 19: GitLab Issue Board [108]

The Problem with Too Much Agility

From 2016 to 2017, the DPAS team made little headway towards project completion. During this period, the team was able to only accomplish the creation of a login page and a landing page. The Project Lead and Branch Manager were concerned with the progress of the project as the majority of the development team had been allocated to the development of the new iteration of DPAS. An analysis of the cause of the deficient performance by the team is detailed below, but it should be noted that these issues are not unique to this team and are a common occurrence in software development projects, especially in teams that are new to Agile Scrum.

For Agile Scrum to be successful, the Scrum Team must have the dedication to follow the process rigorously and through a standardized methodology [111] [112]. Agile Scrum is the industry standard as it has had decades of adjustment to meet changing customer needs and has a proven record of success when applied thoroughly. This is not to say that all Artifacts and Events meet every single goal of the Scrum Team, but without a thoughtful and reasonable assessment of the Scrum methodology with regards to a specific project, every effort to deviate from the norm should be avoided. Implementing Agile Scrum does not require the abandonment of long-term planning, documentation, and structured project management. Rather, a culture of embracing change in a reasonable and thoughtful manner should become part of the corporate culture [112].

The DPAS team took some of the tenets of the Agile Manifesto to heart and, in many ways, quite literally while ignoring others. After their initial Scrum training, the team had a large poster of the Agile Manifesto printed out and hung on the wall of their cubical space. This was to reinforce the idea that being Agile was of utmost importance. Unfortunately, the team did not

have a member onboard who had experience in the successful implementation of Scrum and therefore were novices in their adaptation and execution. Certain team members used this time to assert their ability to work, however they wished and disregarded any form of engineering and technical rigor. It was this last part that led to the lack of deliveries and project execution.

By focusing too much on the letter of the law, specifically with regards to "Responding to change over following a plan", the team entered a cycle of constant change without any successful deliveries [8]. The team was in a constant state of change due to the literal interpretation of this statement. If a team member found a new coding standard or methodology, even a new coding language, they would refactor all the code base to work in the new way. This resulted in the DPAS code being refactored multiple times. The team decided to convert from using JavaScript to a more succinct version called CoffeeScript, which utilizes a structure more akin to that present in Python coding rather than being presented as a C derivative [113]. By implementing this new language in the middle of the development effort, all the developers had to spend time not only refactoring the entire software solution, but also learning the new scripting language conventions. This had a direct and negative impact on the project's cost, schedule, and performance. This was not the only occurrence of too much change in the project execution but is notable as it reflects the norm in execution.

Another example of leaning too much into the Agile Manifesto in a literal sense is the team's approach to "Working software over comprehensive documentation." The team took the approach of adding no internal software documentation, such as JavaDoc or code comments, under the assumption that any code written should be done in such a way as to be "self-documenting." The core idea of self-documenting code is that no internal description of the functionality present in a method is required as the variables and structure are written in such a

way that the intention of the author is revealed through descriptive names and careful structuring [114]. The team disregarded common self-documenting guidelines, such as external documentation as defined as a header above a class definition that describes various internal components to the class, such as the title, functional description, coding author, date modified, etc. [114]. Comments are still needed in self-documenting code, though not necessarily as verbose as presented in a non-self-documenting way. Example topics of comments include the need to describe organization, insights into complex methods and algorithms, assumptions or constraints, and examples of use, when feasible and reasonable [114]. The team disregarded all these factors and instead commented and documented nothing in the code, resulting in a code base that had no historical documentation of developer thoughts or explanations, limiting the ability of other developers to gain swift understanding of the function of the code being reviewed.

"Customer collaboration over contract negotiations" is an integral part of Agile Scrum which the DPAS team did not adhere to. Agile Scrum focuses on creating an environment where stakeholders and developers work closely together to create a working piece of software in an efficient manner while maintaining the ability to pivot to changing needs [9] [115]. The Product Owner holds the critical role of representing and facilitating customer interaction for a Scrum Team [9]. As stated earlier, the DPAS team did not have a Product Owner and therefore had little to no representation of customer needs during critical Scrum events. This is especially relevant when reflecting upon the Product Backlog and Sprint Backlogs. Without a Product Owner prioritizing the Product Backlog with customer needs, the backlogs are left to be prioritized by the development team and their understanding or assumptions of what the customer desires most and without the benefit of the customer collaboration feedback loop [116]. This leads to a

situation where developers are performing software development and major product decisions with little true insight into customer needs, goals, requirements, or priorities. Without direct customer interaction, developers and engineers are left to guess what a customer needs, often resulting in a less-than-ideal end state.

Finally, the team misunderstood "Individuals and interactions over processes and tools." The DPAS team focused on the desires of the individuals internal to the team and not communication and interaction with those outside of the team. They also put an undue emphasis on the tools and Agile process that they were using. This took the focus off of the delivery of a high quality and working product that would meet customer needs. Instead, the solution was treated more or less like an experimental sandbox with frequently changing tools and processes. This directly led to the equivalent of negative velocity. To clarify, the team appeared to be making forward progress as they were creating and testing code, but the code that was created and tested was often rewritten solutions, implementing new ideas, concepts, and coding standards, resulting in almost constant rework. Rework such as this is difficult to track when solely looking at velocity. One must understand that the code being delivered to the integration servers was reworked and not representative of positive momentum towards final product delivery.

As stated prior, the DPAS team's failures to adopt Agile Scrum in a productive manner is not uncommon with development teams. Surveyed customers report projects developed under an Agile umbrella as "unsuccessful" 46% of the time within the boundaries of client benefits, cost control, and time control [117]. Furthermore, for Agile projects to be successful, customers must actively participate in the process and are expected to be "Collaborative, Representative, Authorized, Committed, and Knowledgeable" [89]. When transitioning from a traditional or

predictive methodology, such as Waterfall, it is important to plan out the process transition and stick to the plan. It is equally important to have someone attached to the project to "steer the rudder," keeping the project on track for delivery, normally represented by the Product Owner. Finally, Agile Scrum is a "cooperative social process characterized by communication and collaboration between a community of members who value and trust each other" [118]. Development team members cannot work independently and must instead work closely with the rest of the team, checking in frequently and sharing ideas. When a team does not execute collaboratively working towards a unified product vision, Agile projects fail.

Results of Implementing Systems Engineering Rigor in the DPAS System

In 2019, it was determined that the DPAS team needed more structure. A new project manager was put in place that took on the role of Product Owner. The project manager reviewed the Agile Scrum standards and created a solidified project vision. The team was required to communicate up the chain through the Product Owner to ensure that development standards were being met and that what was developed reflected customer needs. Changes to software execution and coding standards were limited only to those that were required to facilitate delivery of the software. This led to the DPAS 3.0 project making forward momentum towards delivery.

One of the first steps taken by the team was to work collaboratively with the customer to generate a Software Requirements Document (SRD) containing high-level business requirements for the DPAS 3.0 project. The requirements were generated with customer collaboration and concurrence. A formal SRD was created and presented to the project stakeholders with each stakeholder indicating approval of the requirements through a signature. The customers represented a broad range of facilities, including Intermediate Maintenance Activities, In-Service Engineering Activities, the Program Manager, and the Project Manager. While the requirements

are only requirements through a broad definition of the term requirement, those listed in the SRD did solidify the project goals and needs of the customer. The SRD also gave the Scrum Team a starting point for conversations, allowing for ease of clarification and further requirement discovery.

By creating and executing software development in a more structured Agile Scrum method, the team was able to work together more effectively. Previously, the team embraced far too much agility, taking the definition of agility to mean doing whatever they wanted, whenever they wanted to. While having the ability to quickly respond to changing customer needs and requirements is necessary for an Agile Scrum project to be successful, a lack of any form of structure almost certainly leads to the failure of a project. The lack of a form of structure and instead promoting a complete focus on fluidity in changing methods can be defined as an anti-pattern in Agile when considering an anti-pattern to be defined as a "practice that initially looks like an appropriate solution but ends up having bad consequences that outweigh any benefit" [66].

Close team collaboration is essential to the success of Agile Scrum [111] [119]. Moreover, the lack of unified team alignment and a focus on individual software development has been identified as a cause of failures in Agile projects [119]. The lack of close collaboration is readily apparent in the DPAS project. Team members would take Scrum work items on their own and then independently develop them. Pair programming was essentially non-existent. Senior developers actively avoided sitting with junior developers, leading to a lack of essential knowledge transfer and the ability to quickly overcome challenging programming problems [120]. Another consideration of pair programming is that while pairing, developers do not necessarily output more lines of code than individually working, the code is of a higher quality

and less defects are introduced into the general production system as by having a second set of eyes watching the code being developed, there is a built in quality check that would otherwise be absent [121] [122]. While it might not be readily apparent that close collaboration with customers is a systems engineering focus, it is one of the key tenets of the systems engineering discipline. The goal of systems engineering is to create a holistic system that meets customer needs while bringing together cross-functional and multi-disciplinary teams. By reducing the instances of developers working in individual silos, the DPAS team was able to better develop code and deliver a higher quality product.

Finally, the project manager implemented frequent customer check-ins. These check ins took the form of "demos" where the development team would present modules and subsystems that were close to ready to be deployed. This allowed customers to give feedback prior to a publish to production. By demonstrating the working software to the customer prior to release, new requirements were generated, and flaws were identified with time to fix any issues prior to final publication. This has led to a closer and more successful collaborative environment with the DPAS team and their customers.

The path from DPAS 2.0 to DPAS 3.0 has been a difficult one for the DPAS team, but through the embracing of structured Agile Scrum and systems engineering methodologies, such as requirements management, the team has overcome the previous conflicts and struggles and has a clear path forward to final publication. This required a firm hand on the rudder to steer the team to success and ensure compliance with policy, process, and standards. The team had to learn to work together instead of focusing on individual contributions. An understanding that the perfect solution is not the goal, but rather a solution that makes the customers happy and can be delivered in a timely manner, was adopted by the DPAS development team. With these changes,

the team is closer than ever to the final publication of DPAS 3.0 and closing out this almost 10-year development effort.

Case Study 4 – How Too Much Rigor can Stifle Innovation as Shown in the DoD Qualification Management System (DQMS) System

Introduction

The DQMS application is a web-based training-support software application containing code, datastores, and tables that are utilized to track technical training throughout servicemembers' careers. The system is used to validate knowledge and skill sets through short- and long-term individual or group training plans, qualification tracking, and testing. DQMS provides a feedback system so qualification supervisors can provide feedback to qualification candidates. The hosting of the system is done through the utilization of COTS hardware and support software. DQMS currently supports more than 250,000 user accounts, both military and civilian. The system provides tracking for over three million training tasks and qualifications, certifications, and licenses (QCL).

Approach to Systems and Software Engineering

The DQMS project is managed through a very rigorous, detailed, and documented system and software engineering approach. As is required for DoD Acquisition programs, the process is laid out in a Systems Engineering Plan. The plan requires the system to follow a tailored Defense Unique Software Intensive Model. Six Systems Engineering Technical Reviews are held for each planned software release. The team utilizes a modified Agile Scrum methodology. Customer interactions are through the project leadership and formally held during Integrated Product Team

(IPT) meetings. DQMS system configurations are tracked through Product Baselines. Each version is planned out, presented, and approved by the Program Office through a Software Requirements Specification (SRS) which is revised at a Software Requirements Review (SRR). The project utilizes a very rigorous systems engineering methodology, which is unusual for most software projects.

The DQMS SEP states that the DQMS application utilizes a tailored AAF Defense Unique Intensive Model modified to support a project in a sustainment status. The model used is not actually from the Adaptive Acquisition Framework Software Acquisition pathway, but rather a modification of the Major Capability Acquisition. This is most likely due to the author of the SEP not fully understanding the changes presented in the AAF update and rather utilizing the older Software Intensive pathway. This acquisition pathway requires a large amount of technical and engineering rigor, as shown in Figure 20: Model 2: Defense Unique Software Intensive Program. Milestone Decision and SETRs are held before any publication can occur. Versions are fully planned out prior to implementation. The DQMS project performs six SETR events during each version execution cycle, a Software Requirements Review (SRR), Critical Design Review (CDR), Functional Configuration Audit (FCA), Physical Configuration Audit (PCA), Production Readiness Review (PRR), and Project Performance Reviews (PPR) in place of the standard Release Backlog Reviews (RBR).

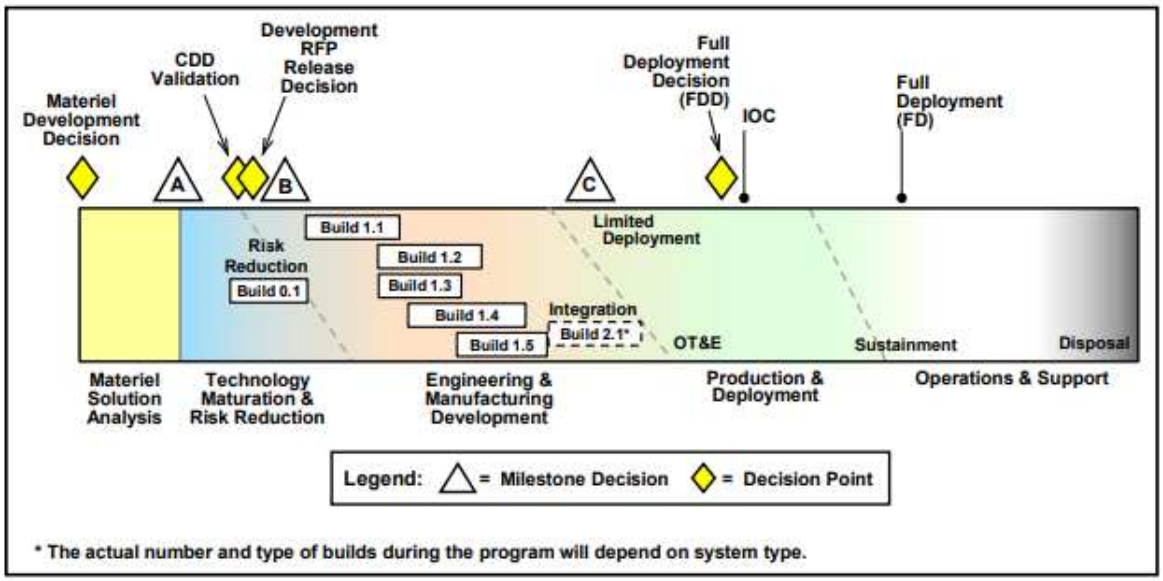

Figure 20: Model 2: Defense Unique Software Intensive Program [123]

At the PPRs, risk assessments are performed as well as communicating Earned Value Management (EVM) metrics and status and Size/Stability, Cost/Schedule, Quality, and Organization metrics. These metrics are important, as breaks in thresholds for these metrics are triggers to take action to update or change the project and reflect a very rigorous attempt to keep the DQMS project within cost, schedule, and performance thresholds, as shown in Table 2: DQMS Software Programmatic Performance Metrics . Of note is the Organizational metric, which has a threshold of "Any deviation from the planned burndown". This requires an explanation to the stakeholders as to why the Predictively planned burndown has been deviated from and how the development team will recover to stay on track for scheduled publication.

Table 2: DQMS Software Programmatic Performance Metrics

Technical Performance Measure	Objective	Threshold
Stability/Sizing – Requirements Variability	Zero variability	< Ten percent of planned requirement increase
Cost/Schedule – PPR EVM	1.0	Lower Threshold = 0.9 Upper Threshold = 1.1
Quality – Defect Open vs Closed	New Defects = Closed Defects	When defects created outnumber defects closed.
Organizational – Full Project Sprint burn down	1.0	Any deviation from the planned burn down.

The DQMS SEP details a procedure for documenting and controlling software enhancement requirements. These requirements are generated at IPTs and documented in an SRS. The requirements are reviewed and accepted by a stakeholder team consisting of a customer, Technical Project Manager, Customer Advocate, Configuration Management Manager, and Systems Engineer during an SRR. If a program office is involved or affected, the specified Program Manager must also sign the SRS. The signatures in the SRS represent acceptance of the requirements. These are the requirements that the software development team will use to perform changes to the DQMS system.

Specific design reviews are required by the DQMS SEP. A Preliminary Design Review is an “independent assessment that informs the Milestone Decision Authority (MDA) of the technical risks and the program's readiness to proceed into detailed design” [124]. This review is used to ensure that the initial design and architecture are adequate to meet the requirements of the stakeholder and that the Milestone Decision Authority believes that the system can be instantiated within the constraints of cost and schedule goals [125]. A Critical Design Review is “an independent assessment that informs the MDA of the technical risks and the program's readiness to proceed into fabrication, system integration, demonstration and test” [124]. The goal of the CDR is to ensure that the system is stable, operates within proscribed requirements, and meets mission goals in preparation for publication to the warfighter and provides stakeholders the confidence needed to proceed with final delivery [125]. These reviews are event driven and occur a single time in the lifecycle of the system enhancement project. They are technical reviews used to grant permission for continuance of a project.

The Problem with Too Much Rigor

The DQMS SEP states that the system is designed and developed using a modified AAF framework. This is untrue, as the AAF Software Acquisition Framework does not utilize the same pathway as laid out in the Major Capability Acquisition Pathway. It appears that the DQMS Systems Engineer did not update the SEP to reflect the changes that make the AAF “adaptive” rather than very static and predictive, as the old Defense Acquisition System operated. This approach utilizes a Predictive methodology, requiring formal customer approval for any changes to deliverables. Planned versions are large efforts and the system is unable to pivot for minor changes. There is little capability for the project to use iterative deployments or adapt to changes in scope and requirements. As can be clearly seen in Figure 21: Adaptive

Acquisition Framework Software Acquisition Pathway , the current AAF Software Acquisition methodology focuses on iterative development with active user engagement [30] [31]. While design architectures are mandated, it is done in an incremental and iterative manner, allowing for rapid changes due to customer needs, goals, and requirements being updated through stakeholder communication and engagement. When planned and executed in a Predictive manner, such as presented in Figure 20: Model 2: Defense Unique Software Intensive Program, this capability is lost, as any changes break the model.

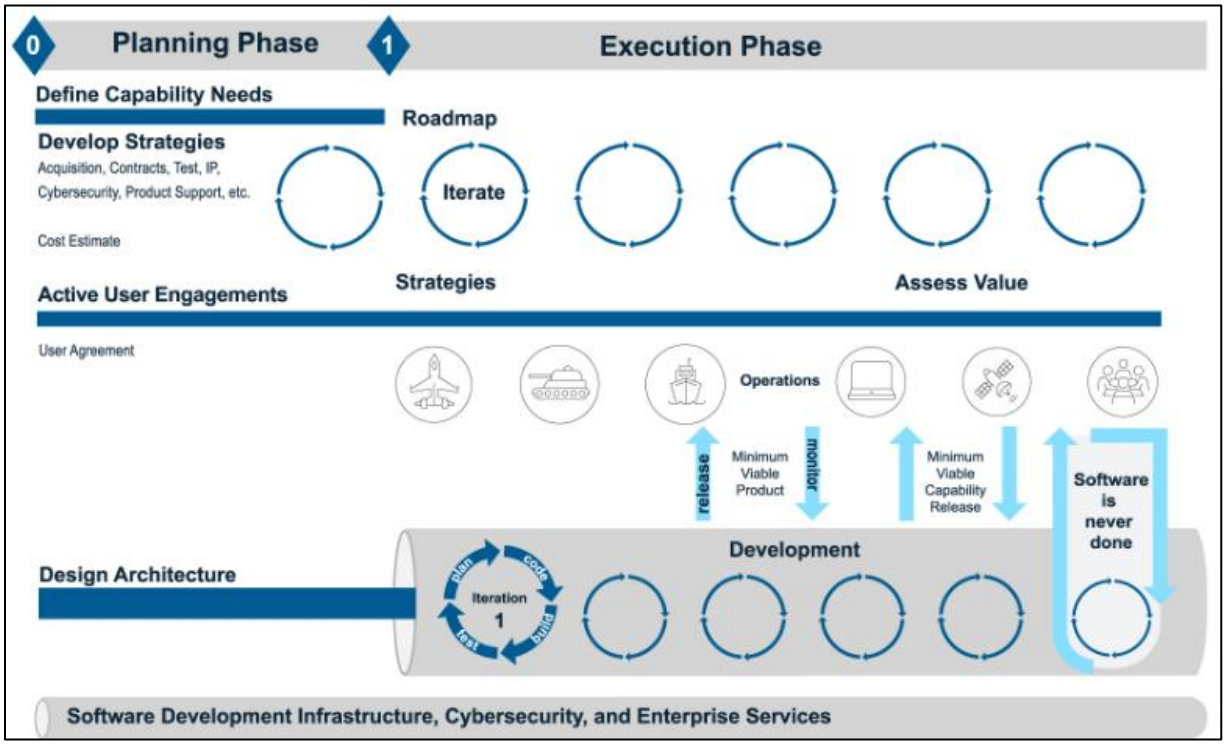

Figure 21: Adaptive Acquisition Framework Software Acquisition Pathway [30] [31]

The approach the DQMS project team takes to track project performance and health is very predictive and onerous. It is difficult to track project performance utilizing metrics such as EVM, as EVM requires the use of the Agile anti-pattern “Big Modeling Up Front” to create a comprehensive model of the requirements of a system, analysis of the requirements, an architecture to execute the requirements to, and a detailed design prior to executing the

development of the system [126]. This is not an Agile or iterative approach, but rather a Predictive approach to design which has been proven to be ineffective for software system development. Peterson et al performed a survey of over 400 waterfall projects and found that software developed under a Predictive, or Waterfall, methodology was either not deployed or ended up not being used as these projects lacked the ability to pivot and adapt to changing customer needs and requirements [127]. The larger the system, the harder it is to fully capture system complexity in requirements documents and models, as well as stakeholder understanding of what they need versus what they think they need [127] [128]. Adding to this is a very tight tracking of deviations of a fully upfront planned Project Sprint Burndown. This assumes that all work is known prior to the execution of development and is another anti-pattern in Agile, allowing for no flexibility in delivering early or late, depending on changes to customer needs or desires. As is clear, the DQMS project suffers from too much rigidity in project execution.

The way that the DQMS project team manages requirements through a formal signature process in an SRS can be problematic in software development. Software requirements need to be fluid and adaptable to allow for changes in customer needs and the evolution of the system concept as development progresses. This is due to the complex nature of software engineering. Software engineering is as much an art as it is an engineering effort. Requirements management must be flexible and adaptable when considering that many customers do not know what they need until they see it or change their mind as new material is presented to them. Utilizing systems such as JIRA to document, track, approve, and maintain status of requirements is much more aligned with the Agile principles than the more Predictive methodology of hard documentation through a formal signature process. In the process that the DQMS project team uses, each time a customer wants a new change request, or a change to an existing change

request, the stakeholder team must fully meet, deliberate, create a new SRS, judge the validity of the SRS, and hold a formal signature meeting. This is not only highly inefficient, but also very costly to the government as multiple work hours are exhausted in this effort, usually with prominent level, and highly paid, government employees requiring attendance.

Formalized project execution documentation, such as SRSs, can lead to the stifling of communication between a development team and the customers. Agile projects focus highly on customer and developer collaboration. One of the key principles of the Agile Manifesto is "Customer Collaboration over Contract Negotiation" [8]. The reason behind this is that the goal of any software project is to deliver what the customer needs when the customer needs it, not what the business team negotiates to build, regardless of customer need. Communication between individuals, be it stakeholders, customers, business managers, or developers, is an evolutionary act and has many obstacles that can stand in the way of full understanding [129]. Communication needs cover many facets of a project, such as goals, objectives, tasks, procedures, constraints, interdependencies, requirements, timelines, prioritization, responsibilities, accountability, models, and even the final system solution [129]. It is seemingly impossible to assess and document the "perfect" system when accounting for all these variables. Therefore, communication through collaboration is required leading to an evolutionary development of a system, rather than a static one. This not only creates a better understanding of the system context, but also builds trust between collaboration partners, "substantially reinforcing the positive influence of reciprocal feedback" [130].

The DQMS SEP requires formalized technical reviews with specified input and output criteria in the form of the Preliminary Design Review (PDR) and Critical Design Review (CDR). This is against current DoD Software Acquisition Pathway intentions. Waiting to demonstrate a

software product to a customer until it is ready for a technical review style meeting is counterproductive and against the collaborative nature of Agile development. DoD Instruction 5000.87, the "Operation of the Software Acquisition Pathway," proscribes close collaboration with stakeholders using frequent product demonstrations, leading to the capability of continuous delivery of capabilities through incremental development [36]. The use of two technical reviews stymies the collaborative capabilities of the project team and does not take advantage of the ability for software to be delivered incrementally. In this instance, smaller releases containing individual change requests can be reviewed and published individually through smaller review boards rather than waiting for large, project sized deliveries.

DQMS Case Study Conclusions

The DQMS system suffers from adherence to a too rigorous engineering methodology. The systems engineering principles that are laid out in the DQMS SEP align more closely to what one would expect in a hardware centric system, rather than a software development effort. The standards taken to meet DoD Acquisition standards are antiquated and do not reflect the updated Adaptive Acquisition Framework policy. Much of the policy laid out in the DQMS SEP requires formalized project documentation and a high amount of technical rigor that is more akin to a Predictive project. While the DQMS SEP does reference daily Scrum meetings, there is no other reference to Scrum or Agile, nor does it lay out management of User Stories, requirements elicitation, or Story Points. The DQMS project suffers from a too Predictive project execution style and would benefit greatly from the adoption of Agile practices, specifically Agile Scrum, utilizing requirements traced to User Stories, closer customer collaboration, and increased developer independence.

The DQMS project team is working on employing more Agile techniques to enable more efficient and flexible development. They are planning to release new versions monthly to ensure rapid deployment to the fleet and enable enhanced communication and feedback from customers. The team is stopping fully planning releases and instead implementing a system closer aligned to Continuous Improvement / Continuous Development (CI/CD). This brings the project closer to alignment with the principles laid out by the DoD Chief Information Officer [131]. The Product Backlog items, specifically defects, have currently all been prioritized, and the team is working on the defects in priority order, bringing the Backlog into compliance with Agile Scrum ideals. As is readily apparent, the team is working diligently to transition to a more Agile focus that can swiftly adapt and react to changing customer needs.

Case Study 5 – Adapting the DoD Work Management System (DWMS) to Modern Software and Systems Engineering Practices

Introduction

The DoD Work Management System (DWMS) is a web-based application initially developed in 1998 as a workload management system designed to manage projects and processes within Department of Defense workshops. The initial system was an Access-based application for internal use in the DoD. In 2002, the DWMS application transitioned to a web-based client server application. In 2004, DWMS expanded to allow for use by external customers. It has since gone through two more versions. The DWMS development team is currently working on an

updated version but has struggled to formulate a suitable and acceptable solution for this fourth version of the software.

DWMS Software Development Process

The DWMS team has done a respectable job of staying up to date on modern software development techniques, implementing industry standards in a thoughtful and impactful manner. The team uses the Agile Scrum framework to develop their software projects. The project has documented standards explaining the implementation of Agile Scrum that is used, as well as workflow expectations and project roles and responsibilities. Quality controls are laid out and defined as well as testing procedures. The DWMS team utilizes a work tracking system, JIRA, to plan and execute Sprints. Various checklists are used as gates to ensure compliance with DWMS development standards. Overall, the documentation is adequate for the DWMS team.

The two core documents used to set policy for software and systems engineering for the DWMS project are the Configuration Management Plan and the DWMS Feature Development Process Standard Operating Procedures. The Configuration Management Plan lays out how the development team sets the standard in which configuration will be controlled, including specified meetings, change requests, requirements traceability, definition of configuration items, test procedure documentation, and software release procedures. The DWMS Feature Development Process Standard Operating Procedures detail DWMS process roles, Sprint definitions, Sprint workflow states, the feature development process, the JIRA management process, the GIT management process, and has appendixes with various guidelines. These two documents set a high-level standard for development efforts. The areas covered are adequate to meet high-level expectations.

The team does a better than average job of knowledge management with the project. There is a repository with documents, operating procedures, and guidance for new employees as well as standard operating procedures for setting up Integrated Development Environments (IDE) and connecting to servers. In depth documentation is present for the deployment process, test case templates, and various miscellaneous information such as technical API guides, user guides, and other data.

The DWMS team has various roles associated with the project, many of which align to traditional Agile Scrum roles. The Code Owner role is defined as the technical leader and is responsible for providing technical direction and vision, code implementation oversight, and setting coding standards. Developers are responsible for formulating software solutions for development efforts, creating and managing test cases, managing merge requests in GIT, and maintaining development item status in Jira. The Project Manager is aligned with the Product Owner with the responsibility of determining features and priorities, setting schedules, and acting as a liaison between the customer and the development team. The Scrum Master is responsible for helping the Project Manager and Code Owner set up and maintain the current Sprint; this role can be filled by any team member. Customer Testers are customers, or stakeholders, who are available to the development team to perform requirement and functional testing for features they are SMEs for. A Deployer is a person responsible for packaging application updates for deployment as well as updating SQL schemas and objects. Reviewers are listed as team members who perform code reviews for other team members prior to a code check in. Information Assurance (IA) personnel ensure that updates are made to application servers as well as ensuring cybersecurity requirements are being met. Finally, Change Review Board (CRB) members are customers representatives who assist in steering the direction of the software development for the

product evolution. As is clear, there are quite a few roles involved in the development of the DWMS product, which feeds into the complexity of customer and stakeholder communication as well as the complexity of the development environment with consideration of hierarchical responsibilities.

The DWMS team utilizes Agile Scrum in a modified fashion. They ascribe to a normalized Sprint schedule, with planned Sprint ceremonies, or events. The Sprint ceremonies performed by the team are the Sprint Retrospective, Sprint Planning, Scrums, which are referred to in documentation as Stand Ups, Sprint Reviews, and the Backlog Grooming. The development team utilizes Story Points to gauge the size of a task, as shown in Table 3: DWMS Story Point Schema. Of note, sizing is directly correlated to days to execute, which is different from normal Story Pointing, as the normal procedure is basing Story Points on complexity, not time of effort. The Jira Sprint Workflow is overly complex, with many steps required prior to deployment. Of note, the Code Owner must review all work prior to submission for final approval.

Table 3: DWMS Story Point Schema

Size	Days	Points
Small	1-2	1
Medium	3-5	2
Large	6-8	3
X-Large	9-21	4
Epic	> 21	5

Configuration items are detailed in the Configuration Management plan, as well as the expectations for configuration management. The Configuration Management plan details the policy for version control. Change requests and defect identification is noted as being generated by users. Test procedures are detailed, along with the management of test cases and where they are stored. Requirements for the creation of design documents, as well as traceability from the documents to functionality, is discussed.

DWMS uses a dirty trunk, clean branch development version control methodology [132]. In this method, the development team pushes all code to a centralized trunk. This code is then baselined and segmented off into a branch which is then used for publishing to a production environment. The main branch represents the most recent version tag along with recent feature and defect solutions since the last baseline. All items that are successfully tested and approved are pushed to the main branch. These changes are bundled, deployed, and then tested in production. After successfully passing a test event in production, the system version is updated, and the main branch is baselined.

Testing is touched upon both in the Configuration Management document as well as in the DWMS Feature Development Process Standard Operating Procedures. The testing policy does not have a formal Test Engineer identified. Rather, the development team performs testing. The procedures state that the originating developer will test their solution themselves and then pass the solution over to another developer for testing in the test environment. User testing comes next and is verified by the project lead. Unit tests are built into the system and are required Sprint Tasks for each Sprint Item. The majority of regression testing is performed through unit tests. Unit testing is “the process where you test the smallest functional unit of code” [133] and is coded into the system for automation, specifically for use in CI/CD pipelines.

Requirements management is not discussed in the DWMS documentation, outside of reference to design documents. The documents do refer to requirements when discussing testing, in stating that tests verify requirements were implemented and working correctly, but there is no reference to what form the requirements take. User Stories are also referenced but no documentation or policy surrounding the creation, verbiage, or tracking of a User Stories. The documentation is written with the appearance of a baseline understanding of what requirements and User Stories are and makes no effort to define the structure or use of either type of statement.

Difficulties with Partially Implemented Agile Scrum and Engineering Rigor

The DWMS project suffers from partial implementation of the Agile Scrum Framework and engineering rigor. While many recommended processes and policies are in place, they are written at a very high level resulting in little detail being communicated. Many of the policies make assumptions of the readers' knowledge and understanding. When viewed as an impartial external entity assumed to have no understanding of a software development lifecycle, structure, or Agile Scrum knowledge, the policies do not function well.

The first problematic part of the policies is found within the roles section of the DWMS Feature Development Process Standard Operating Procedures. Many roles are detailed and are not part of the Agile Scrum framework. Certain roles, such as the Product Owner, are misaligned. The Project Manager can, and often does, serve as the Product Owner. However, the Product Owner should not be responsible for the creation or management of Sprints. That is the job of the Scrum Master and Scrum Team. The Code Owner role is especially problematic as this is not a role proscribed by Agile Scrum. It appears to be a control role that fills duties that are Product Owner duties, such as providing technical direction and vision. Due to the requirement that all checked in code be reviewed by the Code Owner, it also causes an unnecessary

bottleneck in the development cycle. The Team Tester role can be filled by developers, which does not leverage the strengths of having a full-time Test Engineer on staff, but rather uses Developers who are not formally trained in the role. The Scrum Master role is not staffed as an individual, but rather can be filled by any team member. This is not ideal and can lead to serious issues as the Scrum Master is supposed to be the evangelist and the authority on Agile Scrum execution. The Scrum Master is accountable for the Scrum Team's effectiveness and for establishing Scrum as defined in the Scrum Guide [9]. If no training or experience with Agile Scrum is required, then the role becomes inconsequential. The roles need to be streamlined, brought into alignment with the Scrum Guide, and the Code Owner role should be removed entirely with the duties given to the Product Owner.

Another area that needs improvement is the way the team uses Story Points. Story Points are units of measure designed to estimate effort as described relative to work complexity, the amount of work, and the risk involved in the work [134]. The DWMS team has linked Story Point estimation directly to the number of hours required to complete a task. This does not work well in a team of mixed-experience engineers. For a junior engineer to perform a task, it might require four hours. A senior engineer with experience in fixing similar issues may be able to complete the task within an hour or less. In this case, the estimation falls apart as each engineer will estimate the work at a different level. Expanding this out to a project estimation makes the task impossible to estimate as complexity is not considered nor is risk taken into account. Rather than basing Story Points on a set time to perform work, the team should transition to the industry standard of basing Story Points on complexity, amount of work to complete the task, and risk involved.

With regards to the Agile Scrum policy, the DWMS team is mostly following guidance as laid out in the Scrum Guide. One major differentiation is the reference to the Daily Scrum as a Daily Standup. The Daily Scrum is specifically not a stand up, as the Daily Scrum has rules laid out in the Scrum Guide, such as a restriction to 15 minutes, who participates, and what is discussed by the team. A stand-up can be anything used to communicate within a team and can include management and other individuals who are not formally part of the Scrum Team. It is not a status meeting for the Scrum Master, it is a synchronization meeting for the Scrum Team [135]. Beyond changing the name of the meeting to Daily Scrum and adhering to the Scrum Guide rules to manage the meeting, it is common for Scrum Teams to use the "three questions" to focus the discussion had at the Daily Scrum. The three questions are "What did I work on yesterday? What am I working on today? and What issues are blocking me?" [136]. By focusing on what the individual did yesterday, what they plan on doing today, and what is standing in their way, all the team understands the progress the team has made towards completion of the Sprint Goal and any blockers that might be problematic for the team [137]. This is especially helpful as it gives the Scrum Masters notice that there is a blocking issue that needs solved. Clarifying expectations at the Daily Scrum and following the guidance laid out in the Scrum Guide should result in increased efficiency and communication within the development team.

The single largest concern with the DWMS software implementation planning is the lack of formal requirements management documentation. For the DWMS program, requirements are listed in a Software Requirements Document. The requirements are not written in a standardized language format, such as INCOSE style shall statements. Neither are the requirements following traditional Connextra User Story format. Requirements are not tracked in a system that is query-able, nor are they clearly demarcated through a numbering schema. As written, the requirements

do not meet even the most basic requirement writing guidelines proscribed by INCOSE, including testability. The requirements also include extra elements, such as data elements, written in abbreviations with no clear definition of what the elements are. An example of this is a data element named “Prob Factor.” Prob Factor is associated with an element called “Quick Quotes.” There is no explanation of the form of probability that should be shown, the customer driven use or rationale, nor any way to rationally write a test case to ensure the requirement is being met by a system implementation. This is the standard, not the exception, to how the requirements for DWMS are written. A full system “as built” requirements documentation effort needs to be undertaken to build the foundation for future systems engineering work, otherwise the system will continue to fail to meet customer needs, goals, and requirements.

DWMS Case Study Conclusions

The DWMS project has good practices built into the team’s processes and policy, as well as poor practices. It is obvious that the DWMS team has worked hard to implement engineering rigor in the best way they know how. They have made efforts to document requirements in a method that is not too far from standard in software engineering. The team has implemented a form of Agile Scrum, though not as fully implemented as it should be. This is probably due to no one on the team being certified as a Scrum Master or Product Owner. With guidance and work, the DWMS project can swiftly be brought into alignment of a systems engineering focused Agile Scrum development.

As of the time of writing of this case study, the DWMS team is making efforts to implement formalized requirements management. The team has been trained by their Principal Engineer, a formally trained systems engineer, in INCOSE style requirements management. The team is focusing efforts on creating an in-depth requirements document for the new version of

the DWMS project. As the original DWMS program is being sunset once the new version of DWMS is implemented, they are using the time to focus on the next version, rather than documenting functionality that will not be ported over. When the team has this requirements document and a requirements management plan in place, they will have built the foundation upon which to implement increased engineering rigor for the future and more thoroughly understand customer needs, goals, and requirements.

Systemic Causes of Software Project Failure in the DoD

In performing the diagnostic literature review and comparing the findings to real world software project executions, it has become clear that there are three key factors common in DoD software project failure. The first factor is clear communication, both between the customer or stakeholders and the development team and within the development team itself. The second driving factor is the ability to adapt to changing customer needs within a swift period. The final factor is the development team having access to well-written requirements for the software project. These three factors have been shown to drive the success or failure of DoD software projects.

The first influencing factor of software project success is clear communication. Clear communication is essential for software development success as without it, requirements will not be communicated, changes in needs will not be understood, and questions regarding the problem domain will not be answered thereby lowering the work item quality present in the project documentation and backlog. There is a reinforcing causal loop, named “Answered Domain Questions Loop” in this research, present in this influencing factor wherein development teams

must understand the problem domain and reach out to customers. Customers then communicate their needs to the development team. The development team documents the needs in the software documentation or backlog. The development team will then draw from that same documentation to develop the system, thereby resulting in more questions for the customers. There is a similar balancing causal loop, named “Unresolved Domain Questions Loop” in this research, in this process where the customers do not communicate well with the development team and therefore the work item quality present in the backlog is negatively affected. These loops presented in Figure 22: The Importance of Communication on Work Item Quality In this way, poor communication adversely affects the quality of the documentation and software backlog, causing defects or failure in the development process and system of interest.

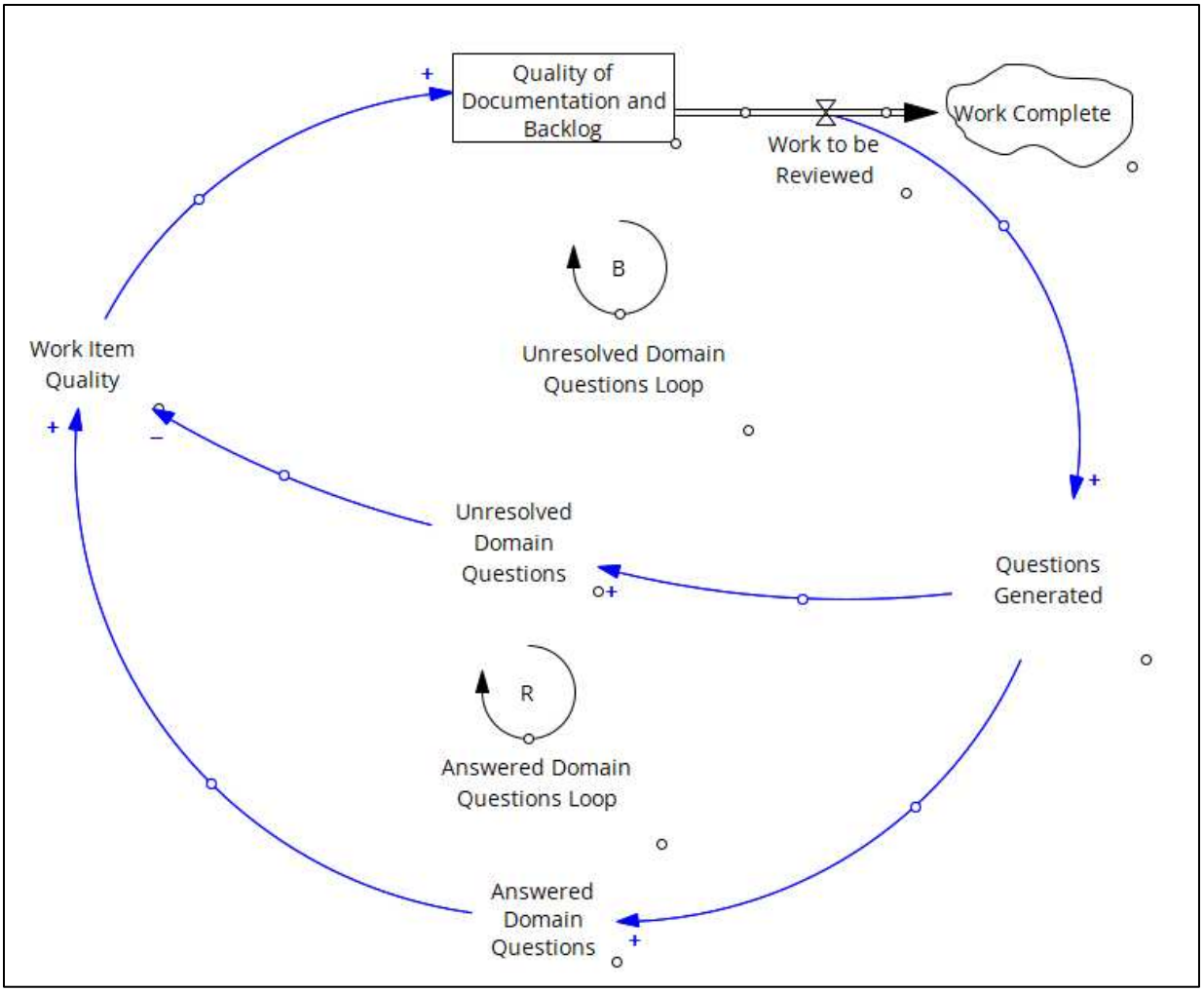

Figure 22: The Importance of Communication on Work Item Quality

The second influencing factor of software project success is the ability to adapt to changing customer needs. In software development efforts, customer needs and requirements often change during the execution of a project. There are many factors as to why customer needs and requirements change, but the fact that they often do change is relevant to the quality of work a development team can produce [17] [138] [139]. There is a causal loop in this process, named "Customer Goal Adjustment Loop" in this research, in that as work is completed and presented to customers, customers will identify changes to needs and requirements that they would like to update or add. The ability for the development team to adapt to these changes directly affects the

time to execute the new plan. As the time to execute the new plan increases, the work item quality produced decreases, as work is still being accomplished while the changes to the project planning and backlog are being finalized. As work item quality decreases, so too does the documentation and backlog quality. This causal loop is shown in Figure 23: The Importance of the Ability to Adapt to Change on Work Item Quality. In this way, the ability to adapt to changing customer needs directly affects the quality of work items present in the documentation and product backlog.

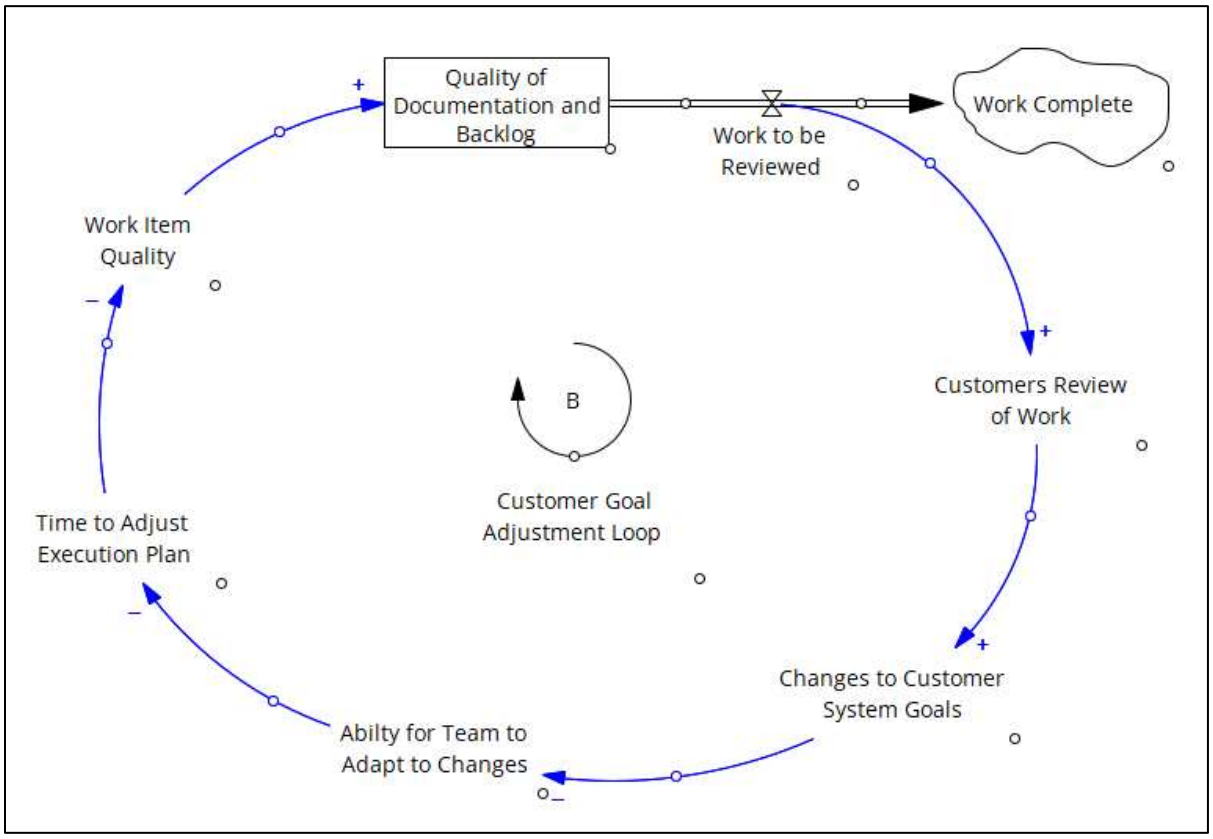

Figure 23: The Importance of the Ability to Adapt to Change on Work Item Quality

The final influencing factor of software project success is the development team having access to well written and high-quality requirements documentation. As work is completed, customers will often communicate changing needs and desires to the development team. The

software development team must then codify their understanding of this new work into requirements documentation. If requirements are not documented well and in a high-quality manner, understanding of the work that needs to be done will be incomplete, lowering work item quality in the backlog. If the development team can document a good understanding of the new work in the requirements documentation, then the work item quality present in the documentation and backlog will increase. This forms a reinforcing loop, named "Requirements Impact on Quality Loop" in this research, as shown in Figure 24: The Importance of Requirements Documentation Quality on Work Item Quality.

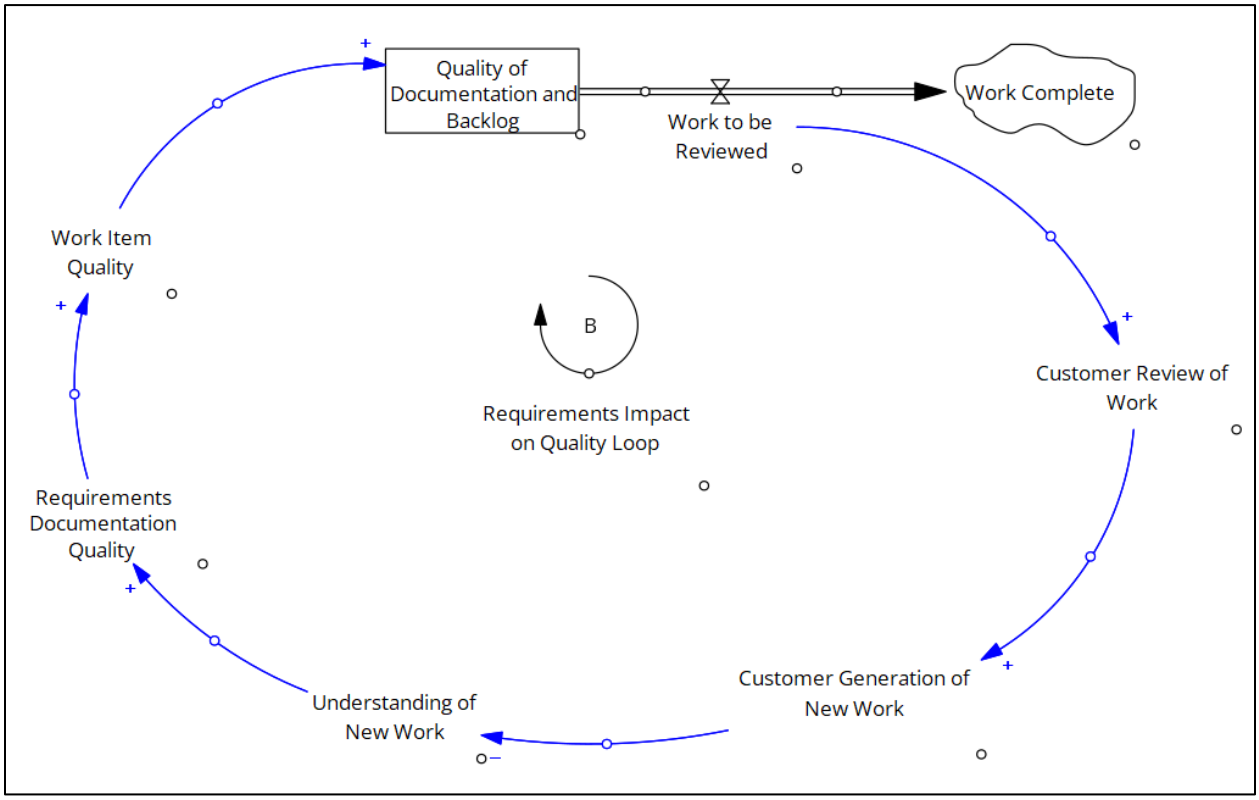

Figure 24: The Importance of Requirements Documentation Quality on Work Item Quality

As shown, communication, requirements documentation, and the ability to adapt to changes all have direct influence on work item quality. Unresolved domain questions have a negative impact on work item quality as the development team is left to engineer a solution

without a full understanding of the problem domain. The inverse is true in which the ability to get domain questions answered has a positive impact on work item quality. The ability to adapt to changes in customer goals can affect work item quality directly as if the team is not able to adjust their backlog and project execution plan swiftly, work that is still in development while new planning is being performed has the potential of being wasted effort. As work is delivered to the customer, oftentimes customers will request new work items to be added to the project. If the developers do not fully understand the new work to be completed, then requirements documentation quality will suffer and therefore drive down work item quality. The effects of these three influencing factors are shown in Figure 25: The Combined Effects of Communication, the Ability to Adapt, and Requirements Documentation on Work Item Quality.

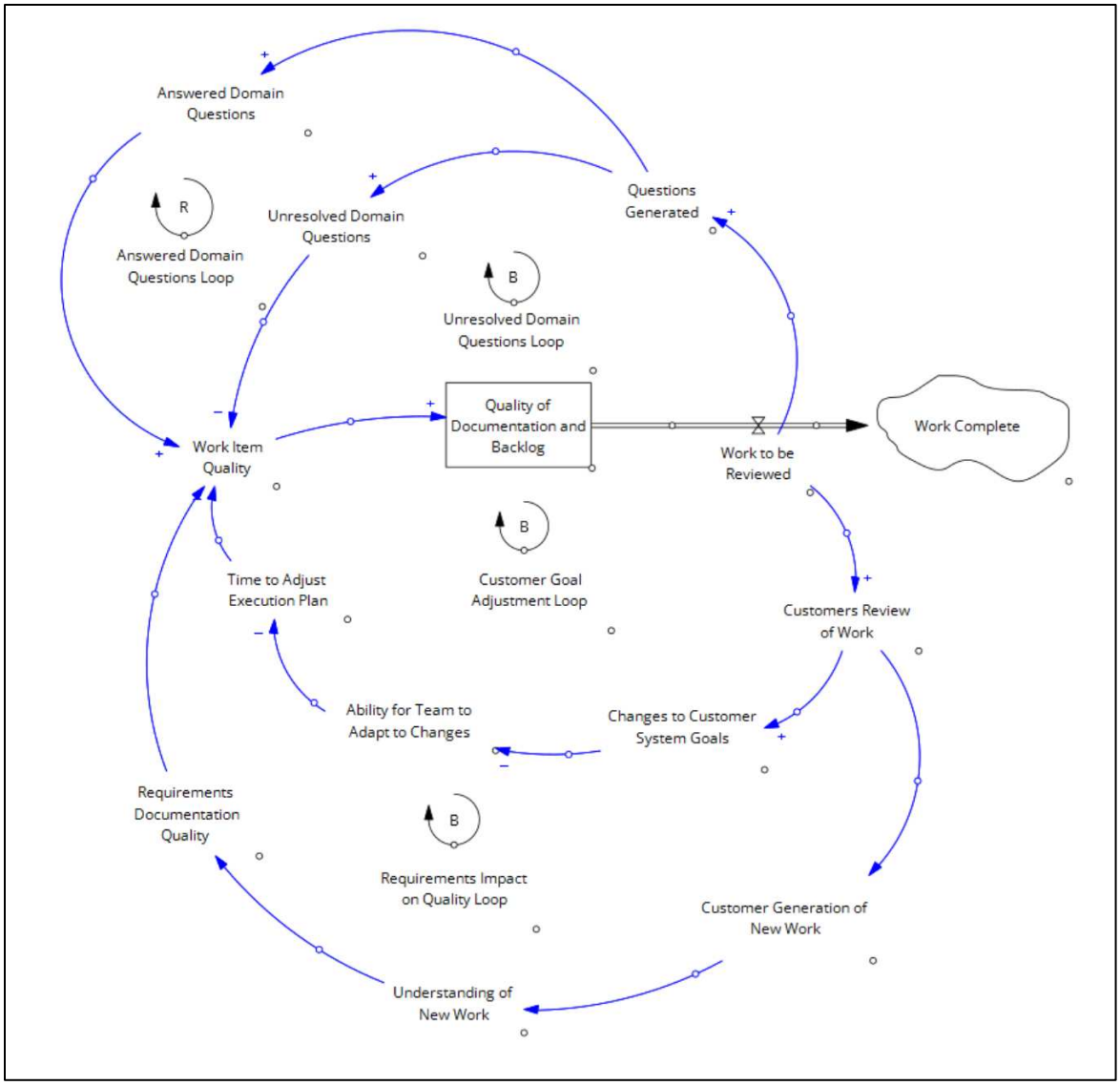

Figure 25: The Combined Effects of Communication, the Ability to Adapt, and Requirements Documentation on Work Item Quality

It is clear to see that the three influencing factors and four causal loops discussed above are systemic causes that affect the quality of work items in software development. The ability to communicate directly affects the understanding of the problem domain and system of interest. The ability to adapt to changes in customer needs and requirements is essential to software developers staying on track for delivery of what the customer actually needs, rather than what the

customer previously needed, and therefore affecting the quality of work items. The documentation of well written and high-quality requirements directly affects the documentation and presence of quality of work items in the documentation and product backlog. While these are not the only factors that can affect the success of software development projects, they are the key influencing factors that affect the quality of work items that software developers utilize to produce the system of interest. Other factors such as complexity of the project, schedule, and cost are not currently explored and are recommended additions to this model.

Case Studies - Conclusions

What are the systemic causes for success and failure of DoD software projects? After reviewing multiple DoD software project executions and performing a diagnostic literature review, it is readily apparent the problem that defense development teams face. Software is often developed without good requirements management and is grown from smaller, proof of concept type projects, or are systems that have been supported for decades and have little documentation. With DoD employees across all US time zones and of differing levels of government, communication can be difficult. Finally, the ability of software development teams to adapt to changing customer needs and requirements can have a major impact on the ability to produce what the customer needs, rather than what the customer needed. These three factors are the key influencing factors behind the success or failure of DoD software projects.

Due to the nature of software development, teams can fluctuate in size as projects go through growth and diminishment, especially during resource droughts. DoD software projects can last years, with developers leaving and returning projects. As opposed to Predictive Projects, many of these projects are executed in an Agile Scrum style project management and there is little documentation of requirements available. In these cases, knowledge management and

knowledge transfer can be low, resulting in teams operating in a “tribal knowledge” style situation, with senior developers passing on knowledge verbally to junior developers. Without rigorous documentation, as new employees join teams and experienced employees leave, knowledge is lost. With good requirements documentation, this knowledge would be retained and thus a better understanding of the system of interest is gained over the course of execution of the project.

Systems that are decades old, as many software projects in the DoD are, tend towards Predictive methods with lip service given to Agile development and suffer from a lack of ability to adapt to changes in customer needs and requirements. Those developing under Agile Scrum run the gamut for little to no actual Agile Scrum alignment and extreme engineering rigidity bordering on Predictive execution to those who have no requirements, documentation, or any level of engineering or project rigor where developers have no oversight and work on what they want, when they want, with no planning. When software is managed in an extremely inflexible manner, the ability to adapt to changes in customer needs and requirements is severely reduced. As software continues to be developed prior to the changes being made to work backlogs, this causes low quality in the work items as the work being completed is not necessarily needed or work is delayed until documentation of the changes is performed, thereby delaying functionality that customers need to later in the development cycle. Therefore, it is important for software development projects to be able to swiftly adapt to changes in customer needs and requirements.

CHAPTER 4 – MODELING SUCCESS AND FAILURE IN AGILE PROJECTS

This chapter seeks to answer research question 3, which is stated here as follows:

Research Question 3: How does the unconstrained and dynamic structure of Agile projects drive the behavior of project successes and failures in software projects?

Agile Scrum projects are based upon maintaining a high flexibility in development tasking to meet a customer driven definition of done, supporting change in the development process [140]. Agile projects face numerous challenges, including communication problems with customers, poor task effort estimation, only developers being engaged in the process, and many others [141]. Agile projects are also executed in different manners dependent upon the project, leading to developers having different views on how projects should be executed and a lack of understanding for external customers that have had previous exposure to Agile [119]. It has become common for software developers to push back against any documentation or process that would stand in the way of the execution of what they view as Agile [119]. Teams utilizing Agile Scrum can meet roadblocks of over-engineering development tasks associated with User Stories [3]. These reasons, plus many others, can lead to Agile Scrum projects failing or suffering scope creep. Understanding this project development space will assist in the formulation of a merged systems engineering and Agile Scrum methodology.

- *Task 3.1: Utilizing understanding gained from Research Question 2, create an Agile Scrum Project Simulation using system dynamics modeling to determine impacts of rework, lack of planning, and underestimation of workloads.*

  Gathered and created reference mode data from previous years' execution with regards to the DLMS project. Two reference mode sets were generated, one representing

DLMS development team execution prior to implementing systems engineering methods, such as requirements management and MBSE, and a set reference mode set showing the changes in execution post-implementation.

Utilized Vensim to create an Agile Scrum project simulation. Influencing factors include Sprint Item Quality, which was found to be determined by requirement, or User Story, clarity and completeness, the availability of customers for swift feedback and requirements clarification, defect discovery time, and the velocity of the development team.

- *Task 3.2: Utilize the model to validate and replicate the conditions of failure noted in Research Question 2.*

  Utilized the Vensim model to validate and replicate the conditions present in the reference mode data sets. Of note, it was found that with a higher systems engineering rigor, less work was needed to complete a project, less defects were generated, and a stable velocity was gained. This is opposed to previous years' executions with lower Sprint Item Quality, which resulted in more volatile Sprint Planning sessions and variable Sprint executions. More "heroic efforts" were required to complete the project that did not implement systems engineering methodologies, as opposed to the project that did.

  This research is further detailed below in the text entitled "Simulating the Effects of Applying Systems Engineering Rigor in Agile Scrum Software Projects.

- *Task 3.3: Utilize the Agile Scrum Project Simulation to gain understanding of the strengths and weaknesses of the Agile Scrum Framework.*

  Utilized the Vensim model to simulate and understand how changes in Sprint Item quality directly drive and affect the overall execution in an Agile Scrum

environment. Lowering Sprint Item Quality directly resulted in increased defect generation, which resulted in the development team having to accomplish an increased backlog of work items to complete the project. Increased Sprint Item Quality resulted in the development team generating less defects and the project having a more stable backlog, allowing the development team to perform the work as planned.

Simulating the Effects of Applying Systems Engineering Rigor in Agile Scrum Software Projects

Introduction

Applying Systems Engineering rigor in Agile Scrum software projects has the potential to see gains in cost, schedule, and performance. Applying Systems Engineering methodologies, such as modeling and requirements management, has proven to have a positive effect on software projects [142] [143] [144] and through the direct application of the methodologies to multiple Department of Defense software projects. To verify the findings, data from one such project was collected and used as inputs for a baseline Agile Scrum Systems Dynamics Simulation Model of Agile Scrum process execution. The influencing factor of Work Item Quality is presented as a combination of the influencing factors of requirements quality and the capability of the development team and customers to clearly communicate, as well as the ability to adapt to changes rapidly. Work Item Quality is used to determine whether work efforts generate additional work in the form of defects, whether requirements defects or traditional software system defects and represents the three causal loops as identified in the case studies. As specific data related to those loops were not available, Work Item Quality was calculated based upon Sprint work complete versus Sprint work planned. This model was then used to verify the results of applying Systems Engineering methodologies to Agile Scrum software projects.

Current State of the Field

There have been efforts to model the Agile Scrum framework utilizing systems dynamics models by other researchers. These models focus on the execution of Agile Scrum projects, but do not define the causality of work item quality upon system development. The models implement additional complexity that is not directly influential on work item quality and therefore is not relevant for modeling the effects of communication, the ability to adapt to changes, and requirements definition on work item quality. Therefore, utilizing existing models from the literature does not meet the needs of this research effort.

In "Agile Project Dynamics: A System Dynamics Investigation of Agile Software Development Methods", Glail et al present a model of an Agile Scrum project execution [145]. The model describes Agile Scrum project execution but does not account for work item quality. Rather than having a variable representing the influencing factor of quality on work items, the model uses a "Fraction Correct and Complete" variable which is a pure percentage. This then drives additional work being added into the product backlog. Therefore, modeling an Agile Scrum execution in this way does not meet the needs of this research effort.

In "Modeling Dynamics in Agile Software Development", Cao et al allude to the need of quality in software development work items, but only present it under the facet of refactoring [146]. Furthermore, the quality of design that the authors refer is directly related to the ability to perform design activities and not to the quality of requirements, communication, or the ability of the team to adapt to changes. It is also not presented at a project level but instead at the level of an individual change request or work item. Due to this, the model presented in this paper will not meet the needs of this research effort.

In “Using Qualitative System Dynamics in the Development of an Agile Teamwork Productivity Model”, Fatema and Sakib present a detailed systems dynamics model of the productivity of an Agile Scrum development team [147]. This model utilizes many variables representing key influencing factors of the success of Agile Scrum projects. Unquantifiable factors such as motivation and working environment are present. In this model, work quality drives the need for rework, but the influencing factors behind work quality are presented as “Experienced Workforce”. The model does not account for the quality of requirements. Communication drives “Team Effectiveness”, which in turn drives “Team Productivity”. It is not directly related to work quality, but rather, the amount of work being performed. Finally, the ability to adapt to changes in customer needs and requirements are not addressed. Therefore, this model will not meet the needs of this research.

In “Factors Influencing Productivity of Agile Software Development Teamwork: A Qualitative System Dynamics Approach”, Fatema and Sakib present a causal loop diagram of influential factors on productivity in Agile Scrum [148]. “Work Quality” is present in the loop and drives “Customer Satisfaction” and “Rework”, but never defined in the article. The key influencing factors identified by the research team were “Team Management”, “Team Effectiveness”, “Culture”, “External Factors”, “Skillfulness”, “External Dependency”, and “Motivation”. Some of these variables can be traced to effectiveness in communication, but it is not fully clear that a holistic view of communication between the customer and development team is addressed as the literature only describes internal communication. Furthermore, the ability to adapt is not addressed in the paper. Due to these factors, this model will not meet the needs of this research.

In “Agile Project Dynamics: A Strategic Project Management Approach to the Study of Large-Scale Software Development Using Systems Dynamics”, Firas Glail presents an in-depth analysis of the influencing factors of Agile Scrum project executions [149]. Glail states “undetected requirements defects can wreak havoc on a project if detected late in the schedule”. One can extrapolate that requirements defects are requirements that either do not fully meet customer needs or are requirements that are written incorrectly and not to the standards of the organization. This is presented as one of the influencing factors affecting the “Fraction of Correct and Complete” work. Customer communication is addressed as customer involvement which is negatively associated with requirements churn. In the thesis, customer involvement is not directly associated with higher quality work output, which is an oversight as it is it is common knowledge that direct customer collaboration improves software project quality [150]. The model presented by Glail in this thesis will not meet the needs of this research.

In “Dynamics of Agile Software Development”, Oorschot et al present a systems dynamics model which models the effects of urgency upon Agile Scrum project execution [151]. The model accounts for changes to requirements, but not communication, the quality of the requirements, nor the ability to adapt to changes. The variable “Error Generation Rate” is affected by the “Software Development Rate” and “Schedule Pressure” but does not account for communication between customers and the development team nor clarity in requirements documentation. This model will not meet the needs of this research.

As is clear, there have been many efforts to model Agile Scrum utilizing systems dynamics methods. Some of the models use other influencing factors and associated metrics, such as staff experience, schedule pressure, staff exhaustion, and the effects of automated testing. These factors are not relevant to the focus which is being measured in this research effort and are

not considered in the model. As the models present in the literature do not directly address Work Item Quality, a new model must be developed to take into the account the effect of Work Item Quality on defect generation in the Agile Scrum software execution process. The model must be generated in a simplified manner to directly measure the effects of Work Item Quality without other factors artificially changing outcomes. The reference mode data used to create and verify this model must use real world data gathered from an Agile Scrum project execution with stable funding and no turnover in staff, allowing for the exclusion of those factors in development success. In this way, the model may focus on the effects of requirements management, customer communication, and ability to adapt as presented as Work Item Quality. Due to these reasons, the existing models reviewed will not meet the needs of this research effort and a new and simplified model is required.

Research Approach

The approach taken for this research is twofold. First, Systems Engineering methodologies were applied to multiple software projects. These projects primarily implemented systems engineering style requirements management and modeling efforts utilizing languages such as SysML or simpler methods such as wireframing [101] [102]. Multiple years of data was drawn from these projects for use in this research.

Data from the projects was used to generate reference modes for use in the Agile Scrum Systems Dynamics Simulation Model. The following data points were drawn from the reference data for use in the project simulations:

1. Work Complete: The number of Story Points completed in total at the end of each Sprint cycle.

2. Sprint Work Accomplishment: The amount of Story Points completed per Sprint cycle.
3. Sprint Work Planned: The amount of Story Points planned to be completed per Sprint cycle.
4. Product Backlog: The number of Story Points remaining to be completed at the end of each Sprint cycle.
5. Sprint Item Quality: The quality of the work items per Sprint Cycle.
6. Velocity: The measure of the amount of work that the Sprint Scrum Team should be able to accomplish based upon previous Sprint execution work complete.

Two reference modes were created. One drew from data generated in 2020, which was prior to the adoption of Systems Engineering methodologies in the execution of the project. This reference mode sets the baseline execution for a software team using pure Agile Scrum execution. A second reference mode was created drawing from data generated in 2023 after the same team implemented systems engineering style requirements management and MBSE. This data was gathered from the work tracking software utilized by the software development team which stores historical data aligned with the project.

Sprint Item Quality was generated for each reference model, as shown in

Table 5: Sprint Item Quality Reference Model and Simulated Datasets, and a simulated Sprint Item Quality was created based upon the 2023 execution. Sprint Item Quality was determined by the following equation:

$$\text{Sprint Item Quality} = \frac{\text{Story Points Planned}}{\text{Story Points Executed}}$$

Using this equation, a number between 0 and 2 was generated. The lower the number, the less quality a Sprint Item had and vice versa. This method resulted in a dataset for each Sprint cycle in the 2020 and 2023 execution. A simulated Sprint item quality dataset was generated through a random number generator algorithm with ranges based upon the ranges of Sprint Item Quality in the 2023 reference data. To ensure alignment with the 2023 results, the set of numbers generated was kept within a very similar standard deviation as the 2023 reference mode work item quality numbers, with the 2023 reference mode having a standard deviation of 0.138 and the simulated data having a standard deviation (σ) of 0.139, as shown in Table 4: Sprint Item Quality Data Standard Deviation. The simulated data was created in Excel using the following function: =NORM.INV(RAND(),1,0.138). This function returns a random number bounded by 0.138 around 1.0 [152]. The simulated data quality follows the same growth pattern as the 2023 project data, as shown in Figure 26: Simulated versus Actual Quality Distribution. The reference mode data and the simulated data are shown in

Table 5: Sprint Item Quality Reference Model and Simulated Datasets.

Table 4: Sprint Item Quality Data Standard Deviation

Data Set	Standard Deviation
2020 Reference Mode	0.222
2023 Reference Mode	0.138
Simulated Data	0.139

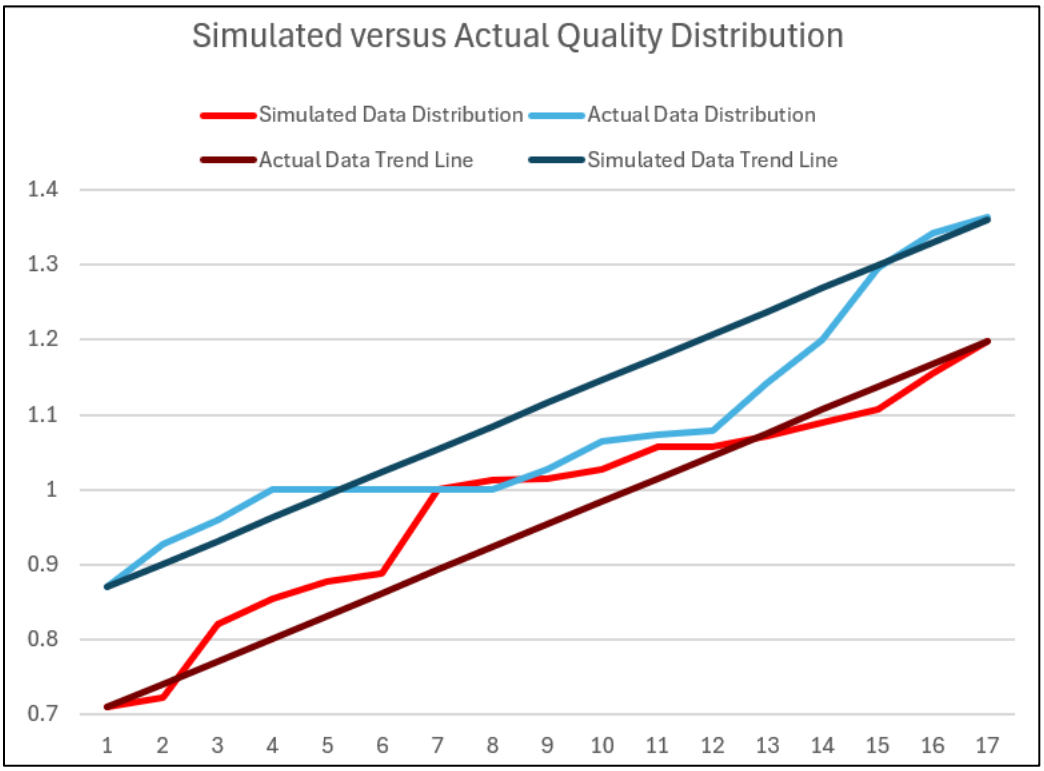

Figure 26: Simulated versus Actual Quality Distribution

Table 5: Sprint Item Quality Reference Model and Simulated Datasets

Sprint Number	2020 Sprint Item Quality Reference Data	2023 Sprint Item Quality Reference Data	Simulated Sprint Item Quality
1	0.9907	1	1
2	0.8667	0.96	1.014
3	1	1	0.821
4	1	1	1.108
5	0.9231	0.87	1.058
6	1.612	1.06	1.089
7	1.094	1.03	1.198
8	0.8308	0.93	0.855
9	0.9142	1.08	1.027
10	0.8	1	0.722
11	0.7573	1	0.889
12	0.7873	1.3	0.71
13	0.5922	1.36	1.058
14	0.6227	1.14	1.156
15	0.7845	1.34	0.878
16	0.7576	1.07	1.072
17	0.8	1.2	1.012

Additionally, data was gathered pertaining to help desk items relating to the software development execution of both 2020 and 2023. Data points collected were help desk items

closed, new help desk items opened, and the total number of help desk items with the status of open in the help desk backlog. The help desk items were grouped by month. The data was then graphed and used to understand the effect of Sprint quality and customer feedback.

The final datapoint gathered for analysis was defects reported per Sprint. This datapoint shows the current number of defects in the software project Product Backlog at the beginning of each Sprint cycle. The number of defects in the Product Backlog per the time of each Sprint cycle graphed against Sprint numbers shows a trendline that is used to determine if defect creation is on the rise or if defect creation is decreasing and if defects created are related to Sprint Item Work Quality.

The Velocity datapoint in the reference models was calculated based upon two other datapoints in the reference model. Velocity is determined by the total Work Complete in a project and by the number of finished Sprint cycles [9]. It is important to note that Velocity is not determined by Sprint Work Planned, but rather the cumulative Work Complete. The following equation is used to calculate Velocity:

$$\text{Velocity} = \frac{\text{Total Work Complete}}{\text{Number of Sprint Cycles Complete}}$$

Methods

The method of analysis validates that the thoughtful application of Systems Engineering methods to the Agile Scrum Framework is twofold. After the initial data gathering and reference models were created, a simulation model was created to show a pure Agile Scrum software project execution, not using any methods outside of those detailed in the Scrum Guide. After creating this simulation model, reference mode data was applied to the model to verify that the

application of Systems Engineering methods to Agile Scrum Framework executing projects has a positive effect, specifically in the measure of Software Item Quality. Finally, qualitative measures were analyzed to understand the influencing factors behind Software Item Quality.

Simulation

Vensim was used to accomplish the task of simulating an Agile Scrum project execution through the creation of the Agile Scrum Systems Dynamics Simulation Model utilizing Vensim simulation software. The simulation has two core loops in the execution and is shown in Figure 27: Simulation Model of the Agile Scrum Execution. The first, and main, loop is the Agile Scrum execution that starts with a Product Backlog which moves through Sprint Planning to Sprint Work Accomplishment then defect generation and discovery which feeds back into the Product Backlog. The second loop is the Velocity loop which moves through the Velocity calculation into Sprint Planning to Sprint Work Accomplishment to Work Complete and which feeds Velocity. There are multiple external inputs to the loops, including the following:

1. Initial Work to Do: The initial planned Story Points needed to complete the project.
2. Time for Defect Discovery: The average amount of time it takes to discover a defect in published software.
3. Quality of Sprint Items: A quality measure that determines the ability of the software developer to formulate a solution that meets customer needs without generating rework or a defect. In this case, defects and rework are a portion of Story Point complexity that was not accounted for or accomplished due to not having perfect understanding of the solution and problem domain.

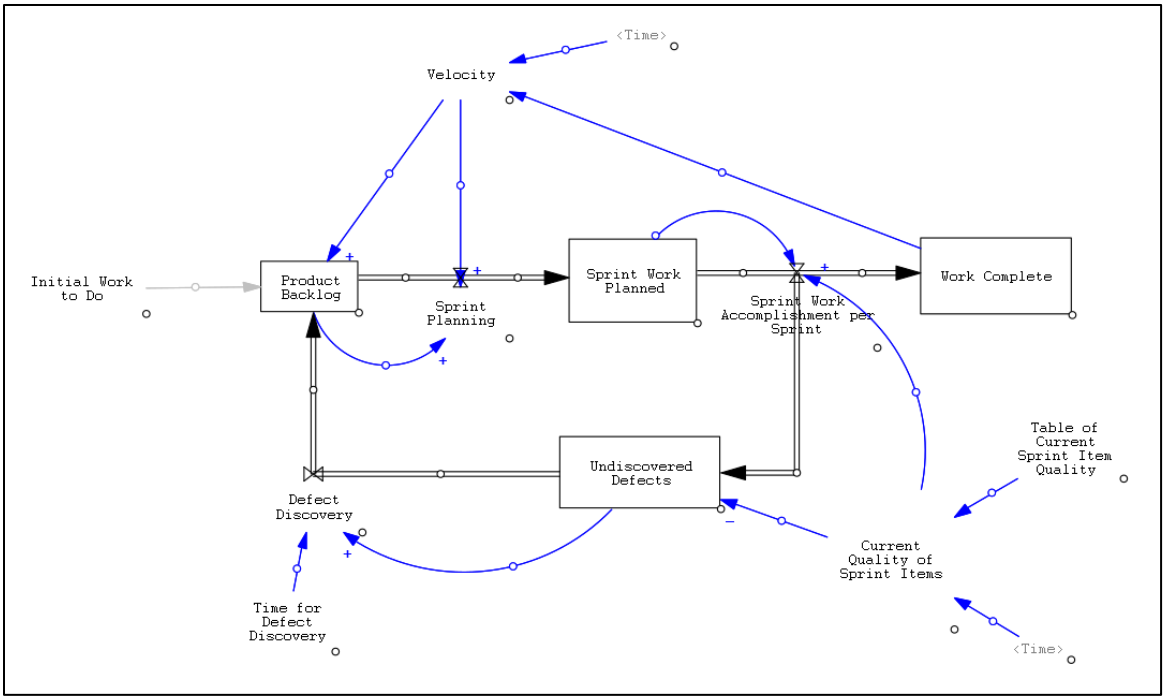

Figure 27: Simulation Model of the Agile Scrum Execution

The main loop execution draws Story Points from the Product Backlog and moves it through to Work Complete. To do this, Sprint Planning subtracts Story Points from the Product Backlog stock based upon the Scrum Team's current Velocity and places it into the Sprint Work Planned stock, which represents the Sprint Backlog. The Sprint Work Accomplishment per Sprint draws the Story Points in the Sprint Work Planned stock and based upon Sprint Item Quality, outputs into the Work Complete stock and Undiscovered Defect stock. The Undiscovered Defect stock is based upon the Sprint Item Quality and the work that has been completed. This stock is drawn from the Defect Discovery calculation, which uses the average time for defect discovery to pull Story Points representing rework and defects from the Undiscovered Defects stock into the Product Backlog.

Qualitative Measures and Analysis

Sprint Item Quality is the highest influencing factor of success in an Agile Scrum software execution. With higher quality in Sprint items, planning is much easier and stable. The Scrum Team does not have to rush, or perform heroic feats, to finish a project. Rework is reduced significantly, and customers do not have to report as many bugs, nor do they have downtime associated with "work arounds" that are required when defects in a production system are present. Scrum Team members also can spend more time working on the solution rather than following up with customers and stakeholders regarding vague, or misunderstood, requirements.

One way to measure customer satisfaction is to track both help desk tickets opened and closed, as well as defects reported. An increase in help desk tickets shows an increase in system destabilization and a misunderstanding of requirements. Customers will always submit help desk tickets as there is always work to be done, such as account creation. Projects are most concerned when ticket numbers are high or on the increase, especially after the release of an updated version of the software to the production environment. When Sprint Item Quality is low, understanding of the needs, goals, and requirements that customers and stakeholders are communicating is equally low. This leads to an increase in the number of help desk tickets submitted by customers and a more unstable ticket flow, with dramatic increases coinciding with software version publication dates. When Sprint Item Quality is high, the Scrum Team has a firmer grasp on the needs, goals, and requirements that customers are communicating, which leads to a more stable help desk ticket trend. While there is not a mathematical equation that can be used to calculate Sprint Item Quality based upon help desk ticket submission values, tracking the trend line of help desk ticket submissions does allow for an understanding of Sprint Item Quality over the course of a project.

Another key influencing factor behind Sprint Item Quality is the quality of requirements associated with work items in the Product Backlog. Not only must requirements be written in a standardized, textual manner that follows the quality attribute guides as discussed in the INCOSE Guide to Writing Requirements, but also be verifiable and validatable themselves [55]. Requirement statements need to be written in a uniform manner and be specific, measurable, assignable, realistic, and time related [153], which allows the Scrum Team to more easily understand the requirement. The Scrum Team must also perform a requirements validation activity with the customer and stakeholders. Misunderstanding of customer needs, goals, and requirements is a key factor in poor development performance. If a requirement is written to not reflect the needs, goals, or requirements of the customer, then any development associated with that requirement is a wasted effort. Therefore, writing requirements in a well-formed manner, verifying the requirements set is written following requirements guidelines, and validating the requirements with the customers and stakeholders directly leads to higher Sprint Item Quality.

Results

To determine if the application of Systems Engineering methodologies would be beneficial to Agile Scrum software projects, data was gathered for a project team over two distinct years. The first year the project had little to no Systems Engineering rigor and solely utilized Agile Scrum methods. The second year of project data reflects the project team applying requirements management and MBSE methods to their software project. Throughout the project, customers remained responsive to the team and so customer availability was not a positive or negative factor. The team remained steady in the fiscal years and, while some members left and new members joined, there was not a point at which the team was in a phase without senior leadership. Good teaming practices were the standard. The Scrum Team had both a Product

Owner and a Scrum Master. Funding for the projects was stable and team members remained on the project for the duration of each fiscal year. While the alignment of a stable project team and stable funding is not necessarily the norm for a commercial project, this is reflective of software projects in the Department of Defense.

2020 Project Execution Reference Mode

The project effort performed in 2020 by the Scrum Team is a project that was executed using pure Agile Scrum methodology. The team was on a 3-week Sprint cycle and held all the expected Scrum ceremonies. Requirements were tracked using Connextra style User Stories. Customers were available for the development team to ask questions for requirement clarification. The product was delivered when versions were ready for release and not on a scheduled drumbeat cycle. The Product Owner was the primary interface for customer requirements and the Scrum Master assisted the Scrum Team with Jira and backlog management.

Figure 28: 2020 Reference Mode Product Backlog shows the Product Backlog for the execution of the 2020 software project. The Product Backlog started with 2191 Story Points to execute over the course of 16 Sprint cycles. The line is not smooth, which gives an indication of1 defects and rework found by customers or the development team. The development team had to add this work to the backlog for the successful completion of the project. The project ended at Sprint 16 with 17 Story Points remaining in the backlog.

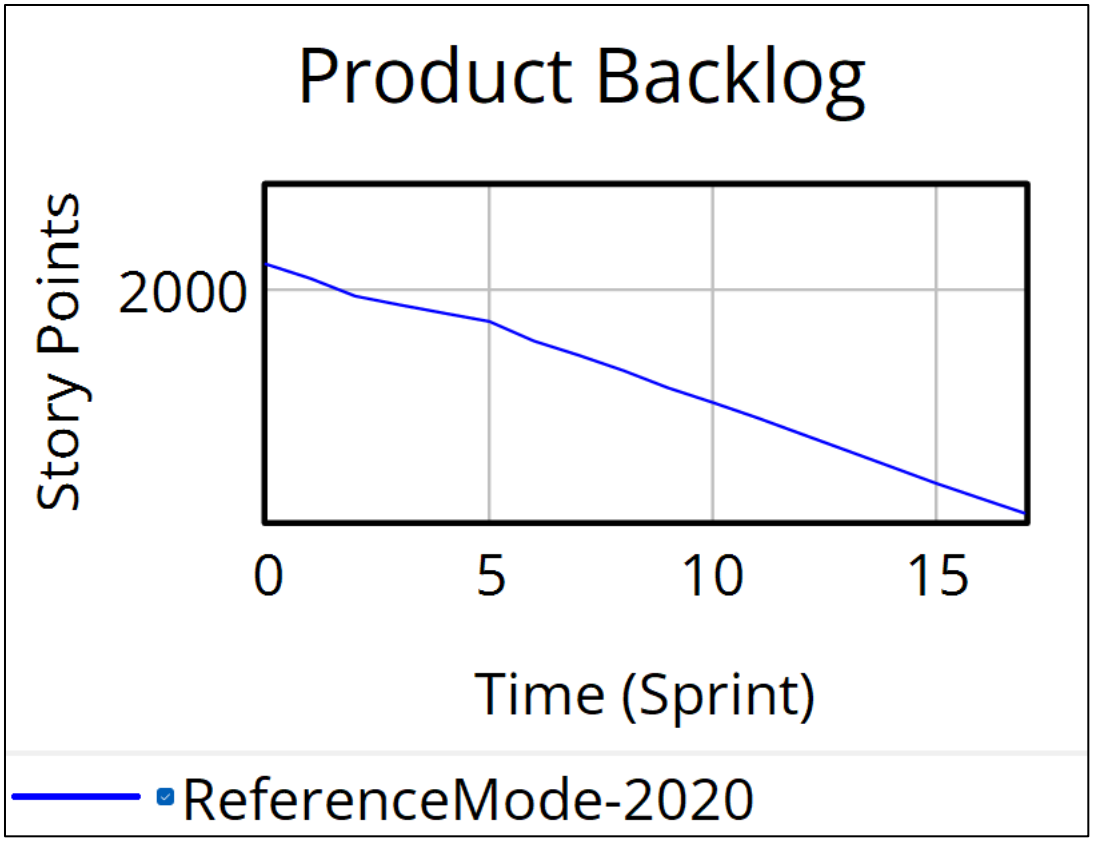

Figure 28: 2020 Reference Mode Product Backlog

Due to the lack of formal requirements management and modeling leading to a lack of understanding with regards to customer needs, goals, and requirements and the project system problem domain, it was difficult for the Scrum Team to have a stable planning process. Oftentimes, the team would plan for less work and have the work be rescoped to a higher number of Story Points. Other times, the opposite occurred, and additional work would have to be added to the Sprint backlog. As additional Story Points were added to the Product Backlog, the team would take on additional work, usually executed in overtime hours, to maintain the project schedule. This had an adverse effect on the cost metric. The volatile nature of Sprint Planning for this project is made clear in Figure 29: 2020 Reference Mode Sprint Planning.

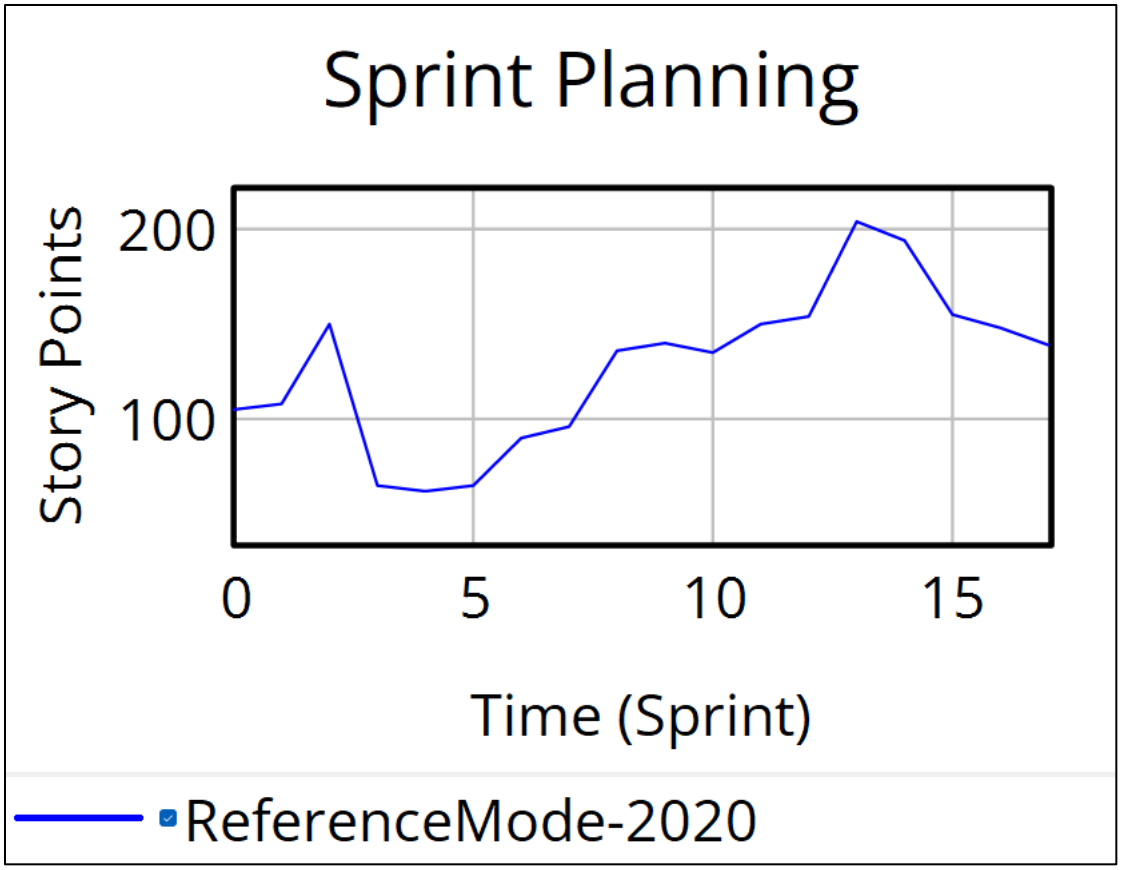

Figure 29: 2020 Reference Mode Sprint Planning

Due to the lack of problem domain knowledge and the constant rescoping of work, the amount of work accomplished per Sprint took an extended time to stabilize. Figure 30: 2020 Reference Mode Sprint Work Accomplishment shows that in the beginning of the project, due to poor Sprint Planning and a lack of understanding of the problem domain, the team was unable to accurately predict and accomplish work through their planning process. An idealized Sprint work accomplishment would be a straight line until the end of the project, at which point it would draw down to a lower number as the project reduces the number of team members or the project is complete. Project execution such as that shown in Figure 24 displays a team that has a poor understanding of customer needs, goals, and requirements and has poor quality work items in their backlog.

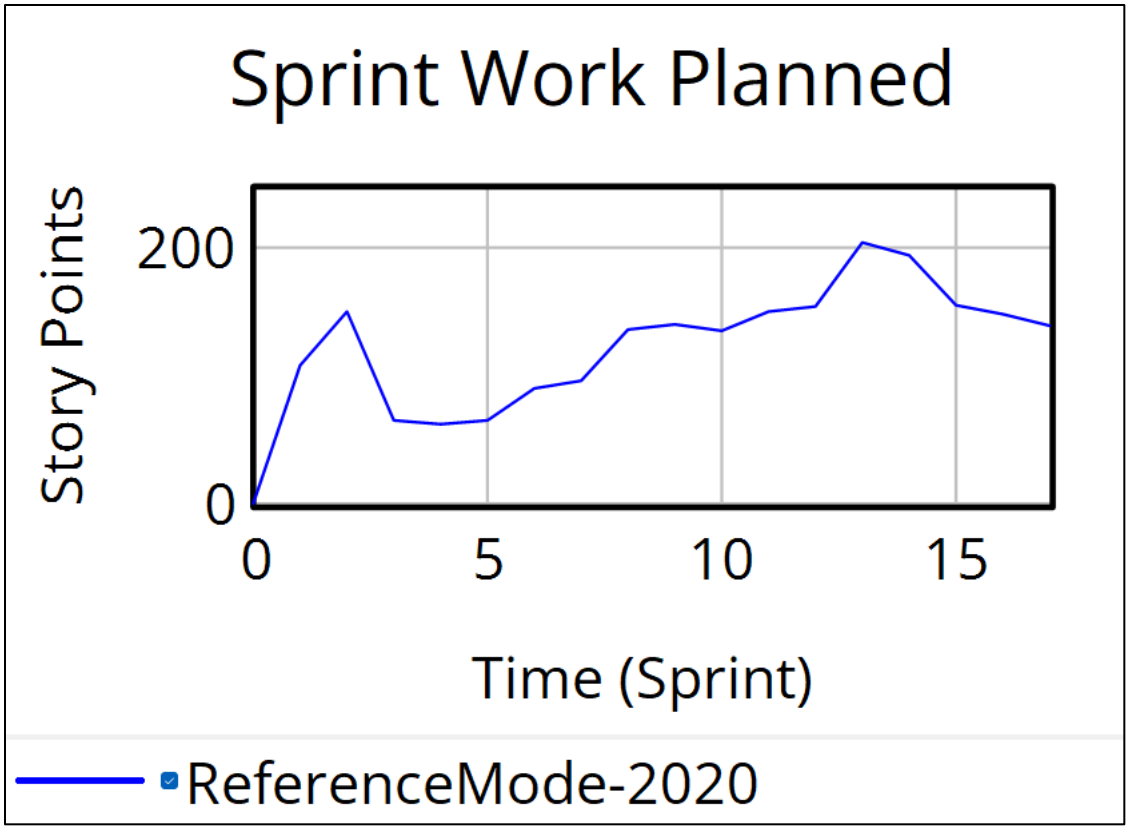

Figure 30: 2020 Reference Mode Sprint Work Accomplishment

In an idealized Agile Scrum execution, a project such as this with a stable and long-standing team should have a straight-line Velocity. When team members work together for extended periods of time, they settle into an understanding of what a Story Point means to their team and have normally established communication patterns and team dynamics such that Velocity tend to be very stabilized [154]. A velocity that has large peaks and valleys is indicative of a team that is not confident, or is overconfident, in their capability of accomplishing work and has a poor level of predictability in their scope of work [155]. Figure 31: 2020 Reference Mode Velocity shows that the Scrum Team had a poor capability of initially planning how much work could be accomplished in a Sprint. The peak and valley at the beginning of the Scrum Cycles showed the team testing their capabilities and, as the Sprint Cycles progressed, the team Velocity

eventually stabilized with an uptick at the end of the fiscal year, which was due to overtime to attempt a heroic effort to complete the project on schedule.

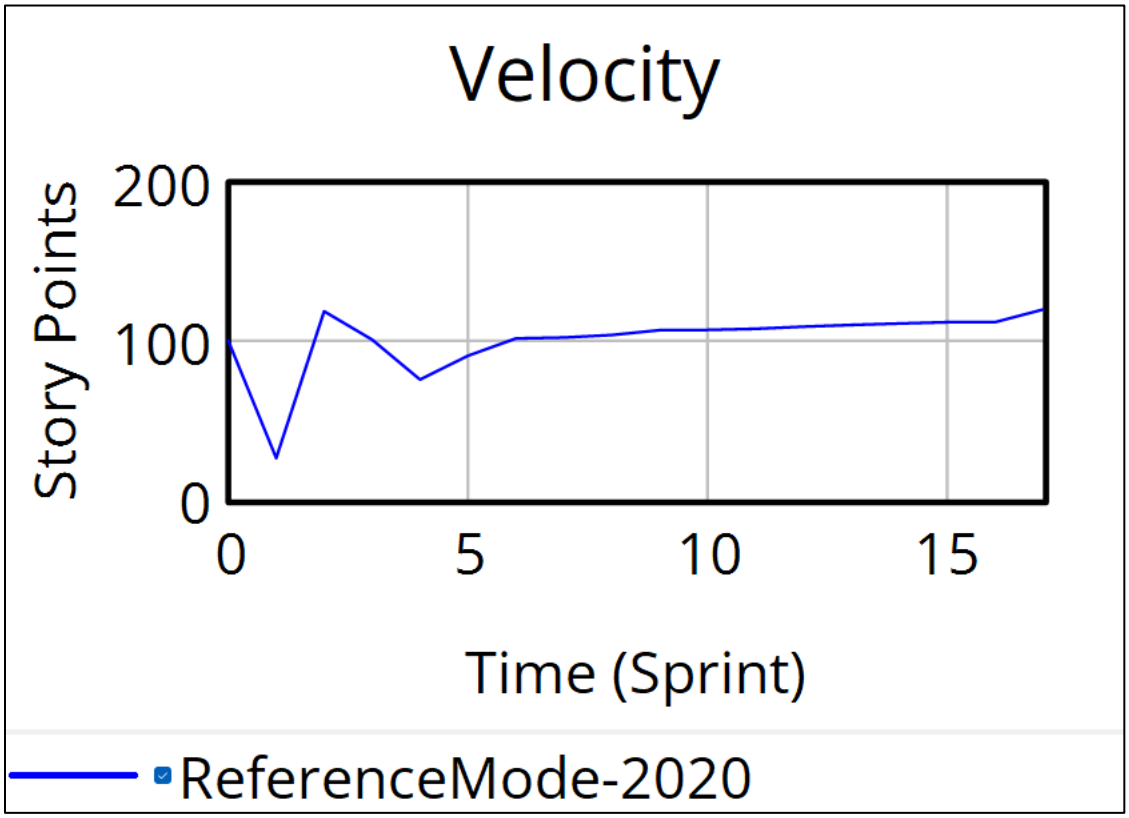

Figure 31: 2020 Reference Mode Velocity

The Scrum Team utilized Connextra style User Stories to track the work items in the Product Backlog. As User Stories are normally not detailed enough to capture the customers' full requirements, the work item quality was, on average, very low. As previously noted, work item quality is based upon planned work versus work accomplished during a Sprint. This is shown in Figure 32: 2020 Reference Mode Current Quality of Sprint Items where the quality of Sprint work was initially high as the team accomplished easy to perform items. As work items became more complex, understanding of the problem domain decreased along with the quality of the customer needs, goals, and requirements being communicated to the Scrum Team.

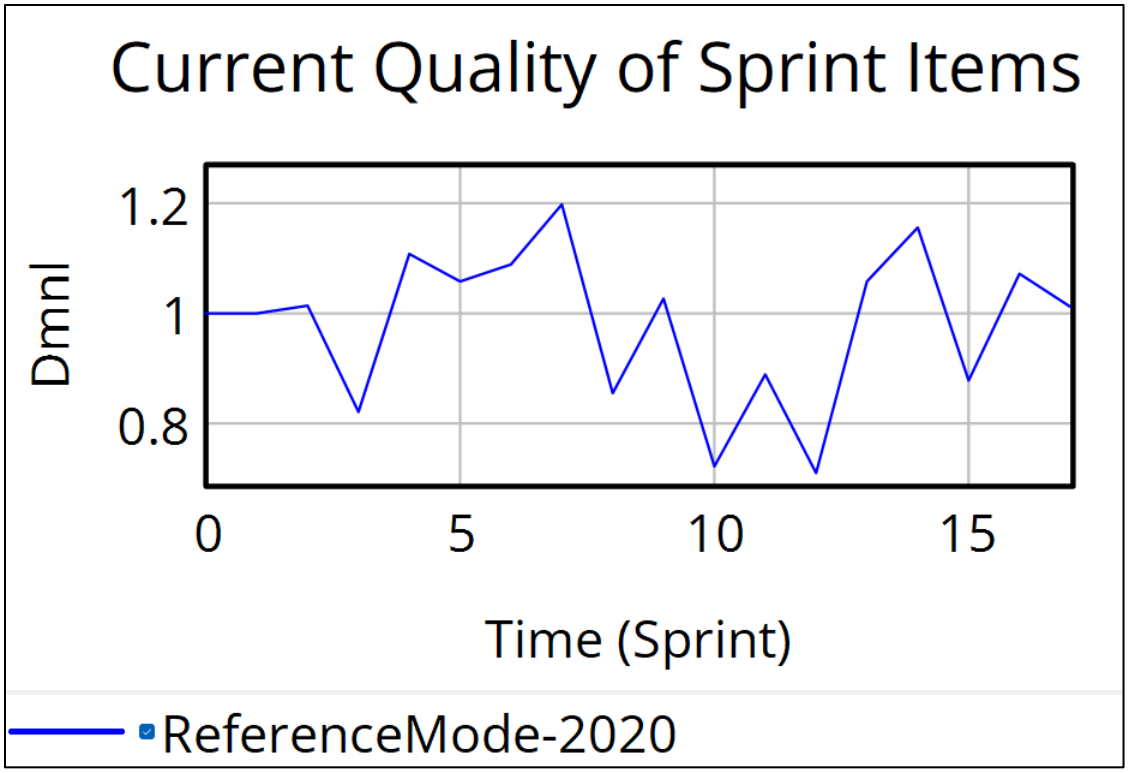

Figure 32: 2020 Reference Mode Current Quality of Sprint Items

Finally, one of the drivers of the need to work overtime and increase Sprint Velocity was the increase in Product Backlog items due to rework and defect generation. As the Scrum Team worked through the Product Backlog, the low level of quality in Sprint Work items resulted in extra work being generated. A lack of understanding in the problem domain leads developers to either refactor, or rework, a product as delivered functionality does not meet customer needs, goals, or requirements. It also results in defects being present in the code. This is especially true of projects where the development team is pressured to deliver the product on time, regardless of increased scope. In this project, Figure 33: 2020 Reference Mode Undiscovered Defects and Figure 34: 2020 Reference Mode Defect and Rework Generation Trend clearly shows Sprint Item Quality decreased, rework and defect generation increased, resulting in an increase in the Product Backlog and production code being delivered that did not meet customer requirements.

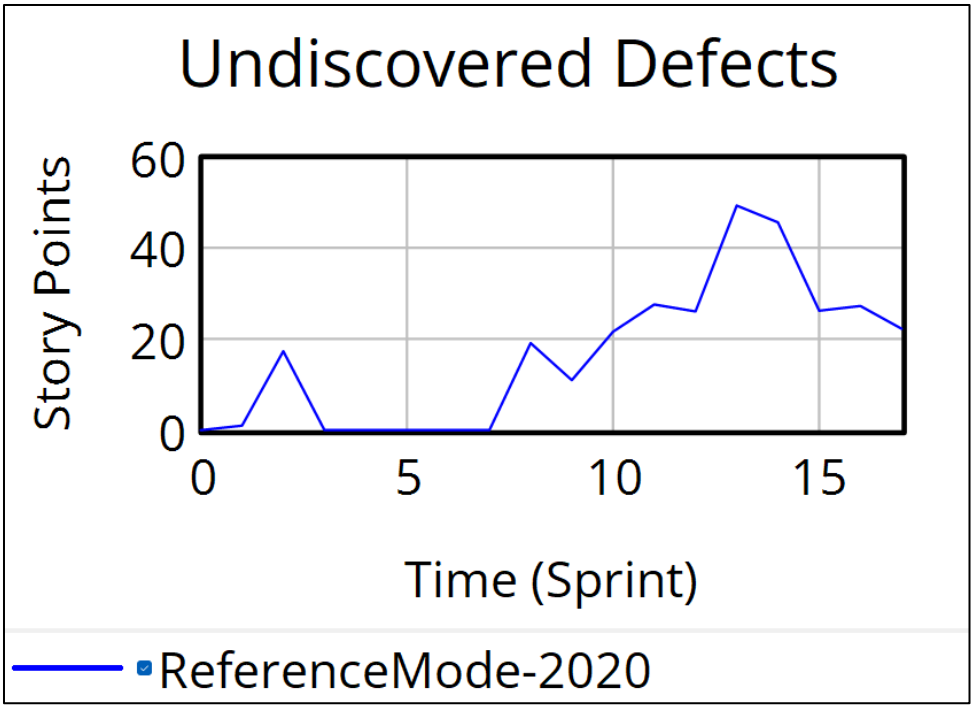

Figure 33: 2020 Reference Mode Undiscovered Defects

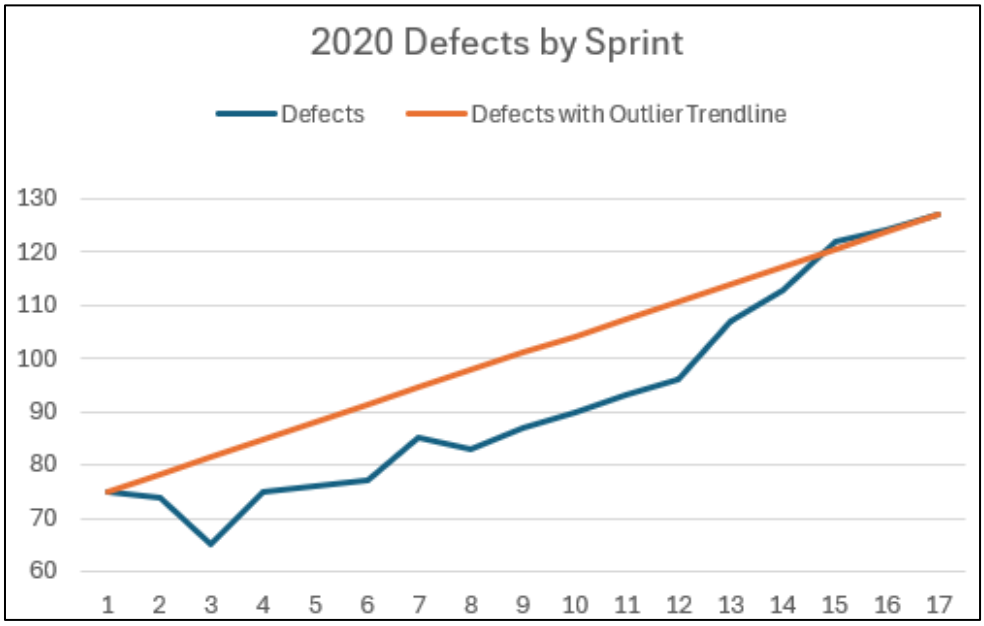

Figure 34: 2020 Reference Mode Defect and Rework Generation Trend

Customer satisfaction with the product is also a influencing factor behind poor execution in a software project. As customers discover defects or functions that do not meet standards,

there is an increase in communication through Help Desk tickets. As new versions of a project are delivered to customers, projects with low problem domain knowledge tend to show a direct uptick in Help Desk tickets. This is shown in Figure 35: 2020 Reference Mode Help Desk Tickets where the team worked diligently to decrease the tickets, fixing bugs and reworking code, but then would have to deal with increased tickets at major releases. Major releases for this software occurred at Sprint 3, Sprint 5, Sprint 7, Sprint 9, and Sprint 12. There was a large push by the team to close out all the tickets at the end of the project cycle, hence the drop in tickets at Sprint 10. In an environment where customers and developers can better communicate, the number of Help Desk items created would be a more stable line, with a normal increase in numbers post-delivery of an updated version of software but lacking dramatic peaks and valleys. The Help Desk ticket backlog for the 2020 project execution shows a project where customers have low satisfaction, and the Scrum Team has an equally low understanding of customer needs, goals, and requirements.

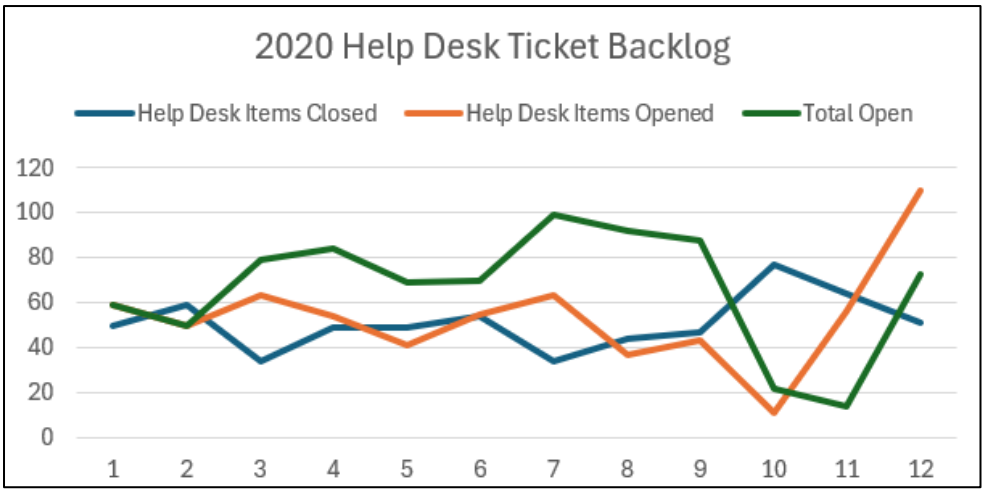

Figure 35: 2020 Reference Mode Help Desk Tickets

2023 Project Execution Reference Mode

In 2023, the software project Scrum Team started applying Systems Engineering methodologies in conjunction with their Agile Scrum execution. Specifically, the team started tracking requirements in the INCOSE style and tracing them directly to User Stories that were stored in their Jira project instance. The team also started using rudimentary MBSE methods, such as wireframes, to assist with communicating understanding of the problem domain with the customers and stakeholders. The team continued the use of normal Scrum roles, such as having a dedicated Product Owner and Scrum Master. Rather than delivering upon work completion, the team was scheduled to deliver a working product every six weeks, or every other Sprint completed. This was a much more structured software execution, but still maintained the flexibility offered through Agile Scrum. It should be noted that the explanation for the large numerical difference in backlog Story Point values and Velocity between 2020 and 2023 is that the Scrum Team changed the baseline valuation of Story Points during this period, resulting in smaller values. As Story Points are a measure of complexity, the numerical value does not matter if the Scrum Team consistently applies a standard growth rate.

Figure 36: 2023 Reference Mode Product Backlog shows the Product Backlog for the project. The project started with 671 Story Points as the initial amount of work to do for the project. It is clear when compared to Figure 28: 2020 Reference Mode Product Backlog that the product execution was much smoother and that there were less defects and rework generated as the Product Backlog is stable with only two peaks noted. This displays a much more refined understanding of the problem domain and that the Scrum Team did a much better job of detailing customer needs, goals, and requirements.

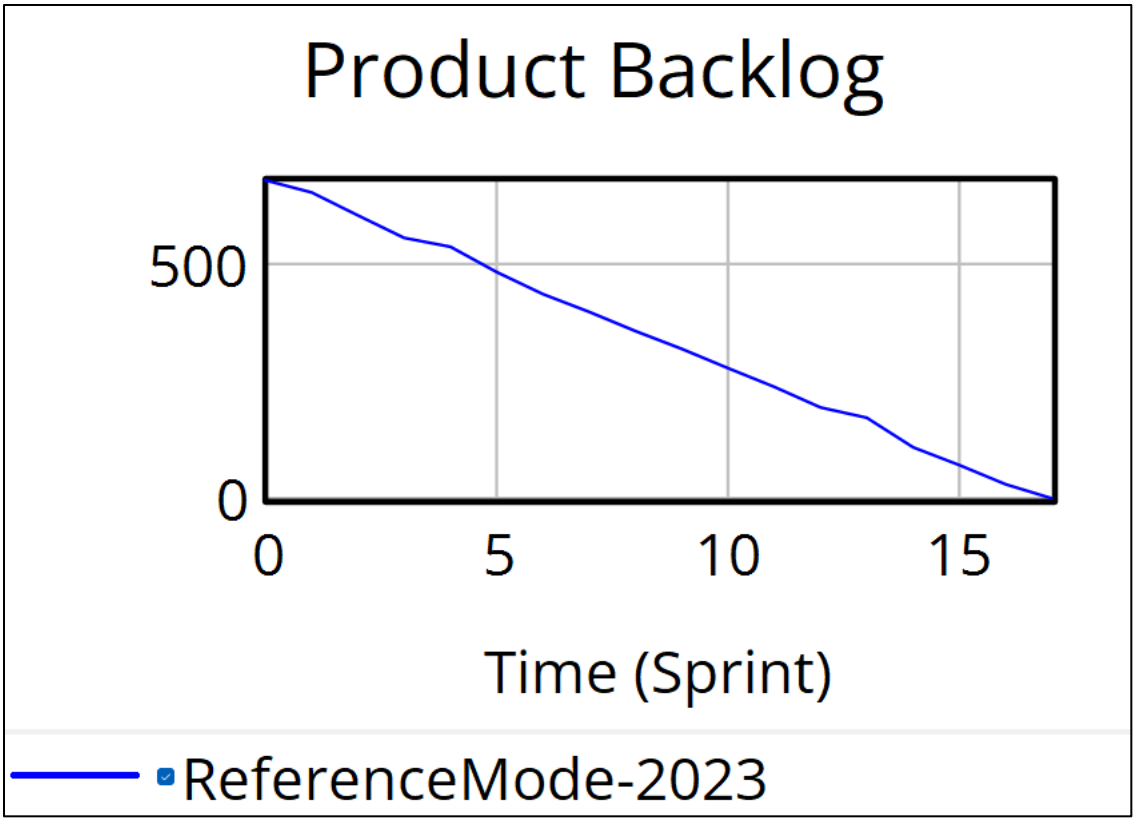

Figure 36: 2023 Reference Mode Product Backlog

Sprint Planning was equally stable, with the Scrum Team frequently executing around an average of 40 Story Points per Sprint. There are peaks and valleys, as is explained by holidays, scheduled and unscheduled leave, etc. The most important understanding to be gained is that there is not dramatic peaks and valleys present, nor a large growth or decline present in the Sprint-to-Sprint execution. The team is able to capably plan an adequate amount of work for a Sprint and then execute that work, as shown in Figure 37: 2023 Reference Mode Sprint Planning and Figure 38: 2023 Reference Mode Sprint Work Accomplishment. The trend lines present in both graphs track with one another which shows that the Agile Scrum cycles are working properly. The Scrum Team understands the problem domain better than in the previous project and therefore can better plan and execute the work necessary to deliver within predicted cost, schedule, and performance metrics.

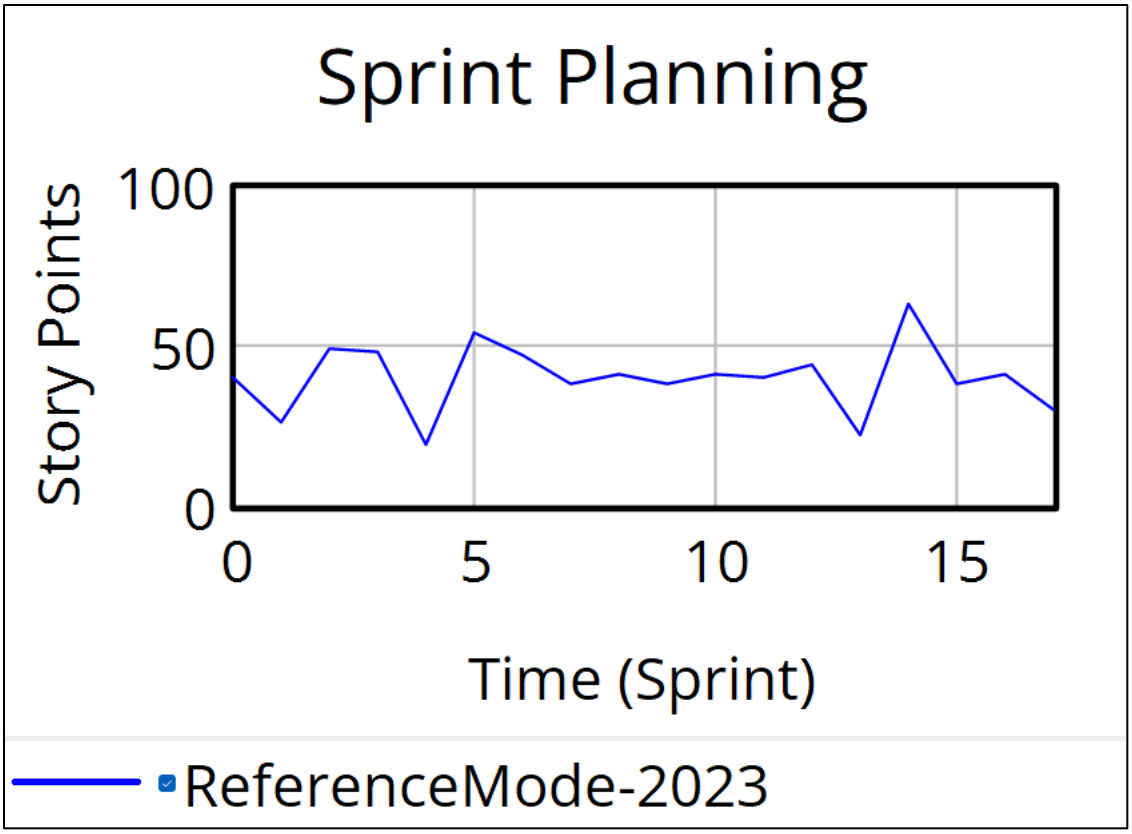

Figure 37: 2023 Reference Mode Sprint Planning

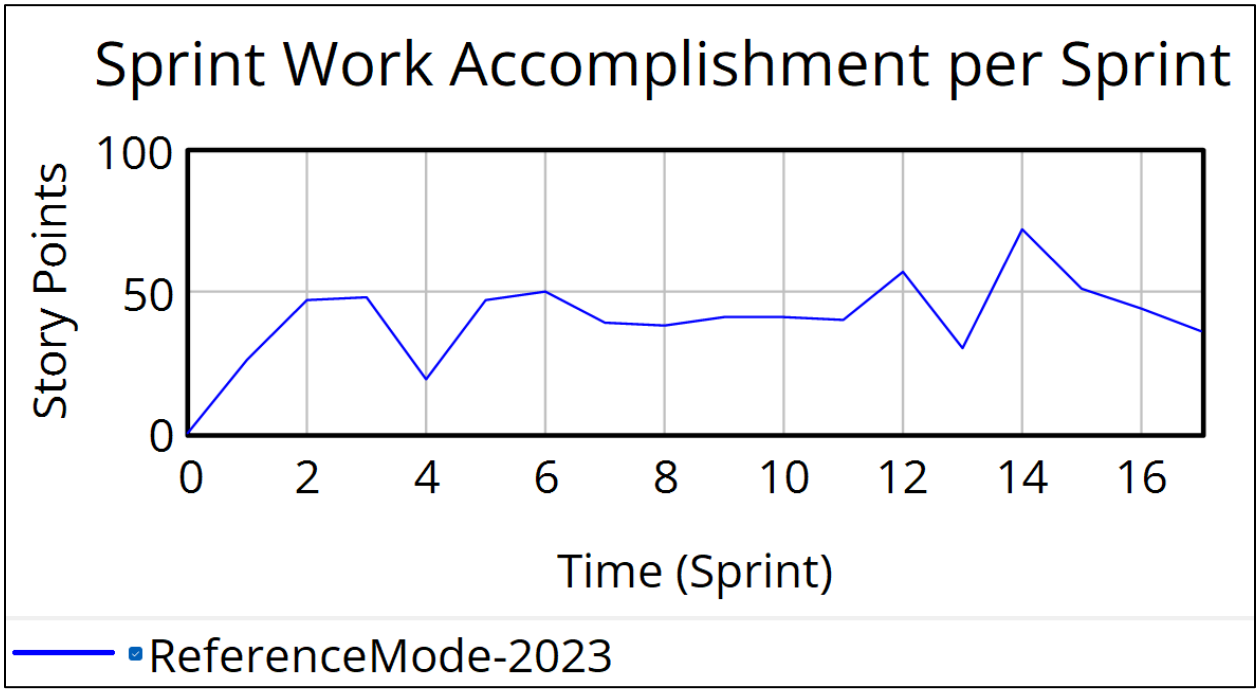

Figure 38: 2023 Reference Mode Sprint Work Accomplishment

As is expected, Velocity also remained relatively stable throughout the project execution, as shown in Figure 39: 2023 Reference Mode Velocity. There were slight variations, which is expected and explained as before by holiday, scheduled and unscheduled leave, etc. While a perfect straight-line execution is described in textbooks, real world applications always have a fluctuation in project team performance. The essential information to note is that the peaks and valleys were small. If this project were to be expanded over multiple years, the line could be expected to become more and more stable.

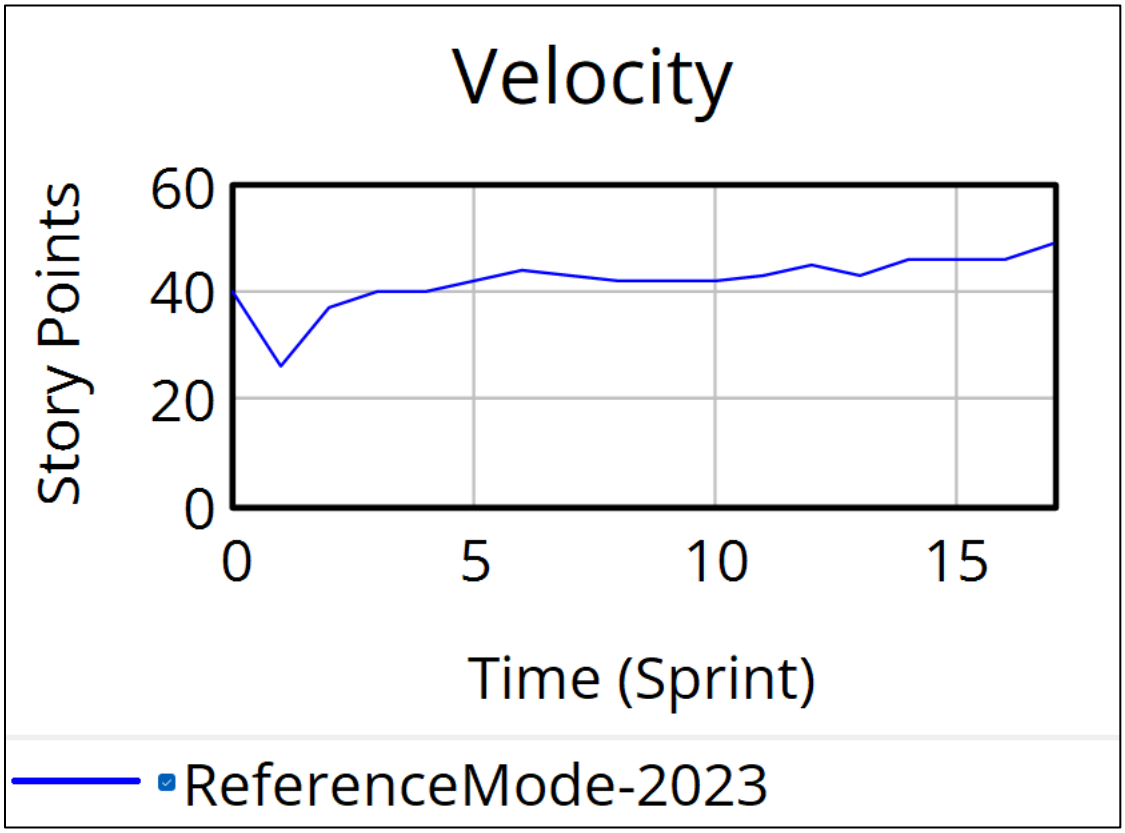

Figure 39: 2023 Reference Mode Velocity

Sprint Work Item Quality was also stable and averaged on the high-quality side when considering a work item with no negatives or positives when considered for quality rated as a 1, as shown in Figure 40: 2023 Reference Mode Current Quality of Sprint Items. The team not only did a better job of writing quality work items at the beginning of the project but showed growth

towards the end of the project in writing exceedingly high-quality items. This is not unexpected, as with increased experience with new methods comes increased capability in performance. It is easy to conclude that the quality of the Sprint Work Items is related to a better understanding of the problem domain through increased communication with the customers and stakeholders. The team accomplished this communication through the utilization of wireframes and INCOSE style requirements. The team ensured that all requirements met the standards of quality as laid out in the INCOSE Guide to Writing Requirements [55]. To simplify their Product Backlog, the team still wrote User Stories but utilized the ability to link Jira items together and created a repository in Jira of all the project requirements and then linked requirements to User Story work items. In this way, the Scrum Team members could have higher level work items with lower levels of detail in the backlogs, but also have full visibility of the requirements driving the work items.

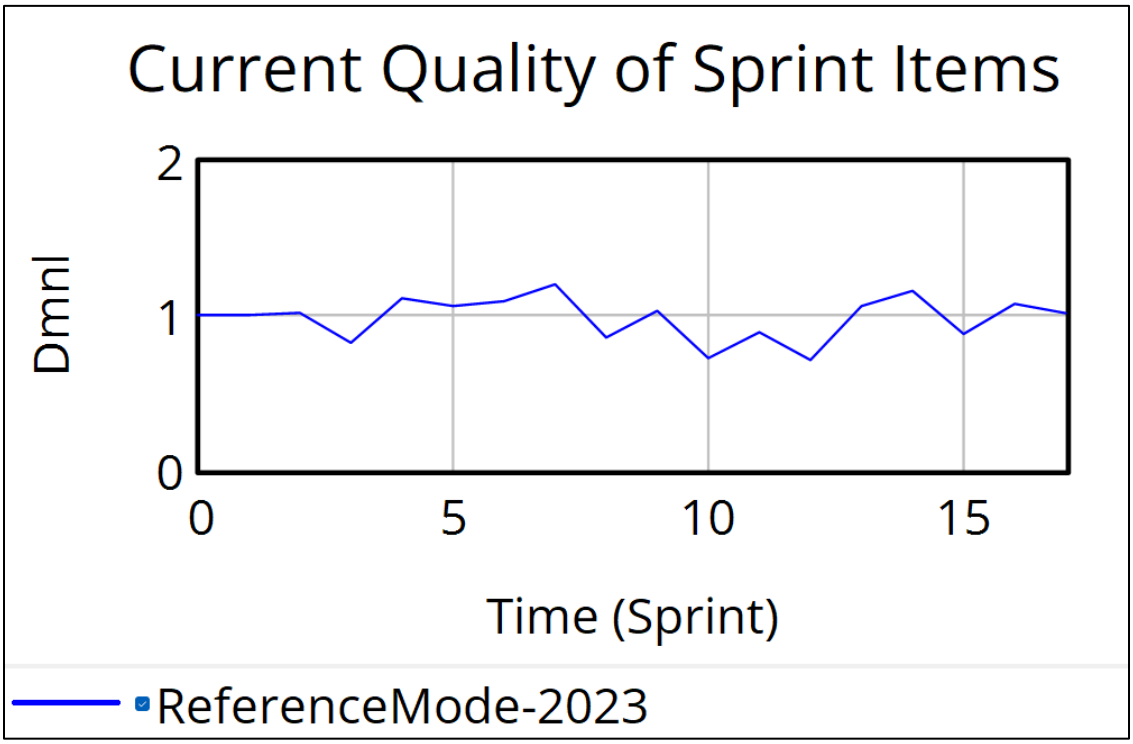

Figure 40: 2023 Reference Mode Current Quality of Sprint Items

Defect and rework generation was low in the 2023 project. There is a high probability that this is due to a higher understanding of the problem domain, but also a result of the test engineers having a better understanding of what work items are tied to specific requirements. As the requirements were written to all be testable, each requirement was easily noted by the test engineers and test management was better performed. The guesswork of functionality was removed from the system. As can be expected from the quality of Sprint items shown in Figure 40: 2023 Reference Mode Current Quality of Sprint Items, there was some defects and rework required in the beginning of the project. The amount of rework and defects generated was quite low in comparison to the previous project. Defect generation is shown in Figure 41: 2023 Reference Mode Undiscovered Defects.

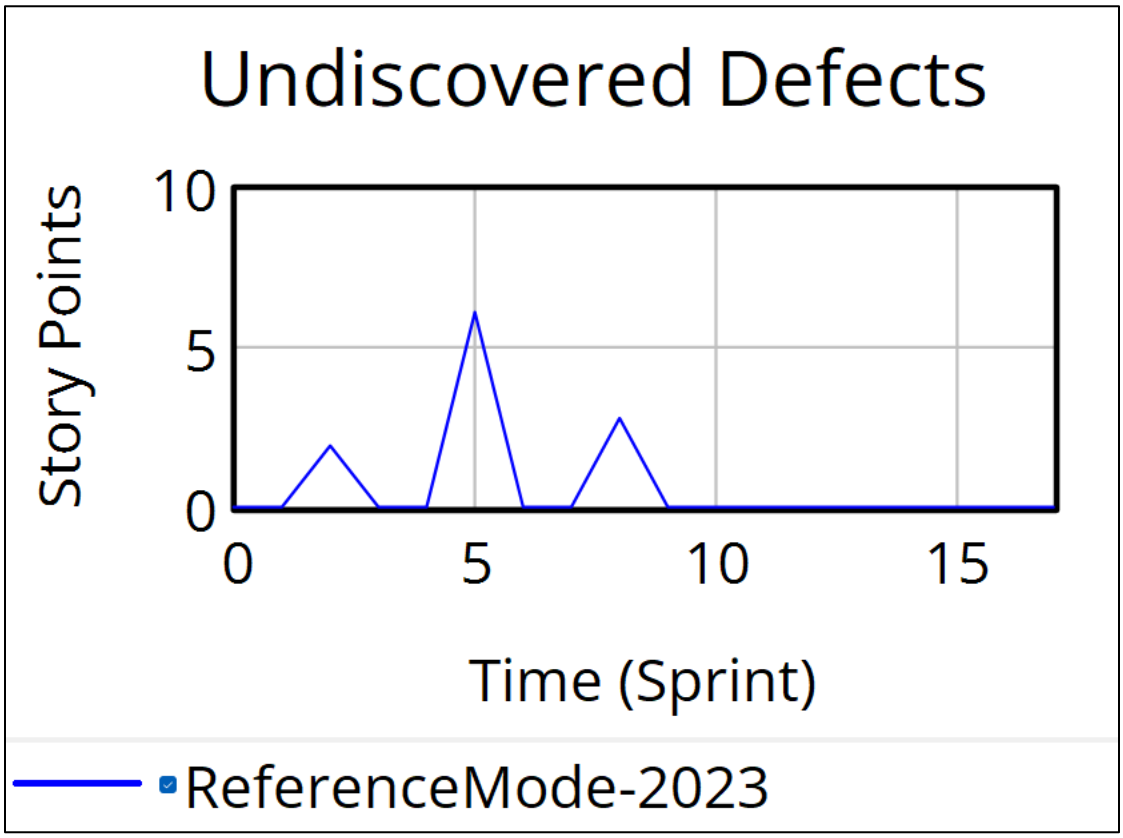

Figure 41: 2023 Reference Mode Undiscovered Defects

It is not surprising that the trend line for defect generation dramatically decreased over the course of the project as problem domain understanding and customer communication quality increased. Figure 42: 2023 Reference Mode Defect and Rework Generation Trend shows the effect that increased Systems Engineering rigor has had on the quality of work produced by the Scrum Team. The red line shows the trend towards decreased defects introduced into the production environment. At Sprint 14, an overhaul of the SQL system and queries took place, increasing defects in the system. Even accounting for this uptick, there was a decrease in defects present. Therefore, it can be expected that applying Systems Engineering methods allows for an increased understanding of the problem domain and customer communication, directly leading to a decrease in rework and defect generation.

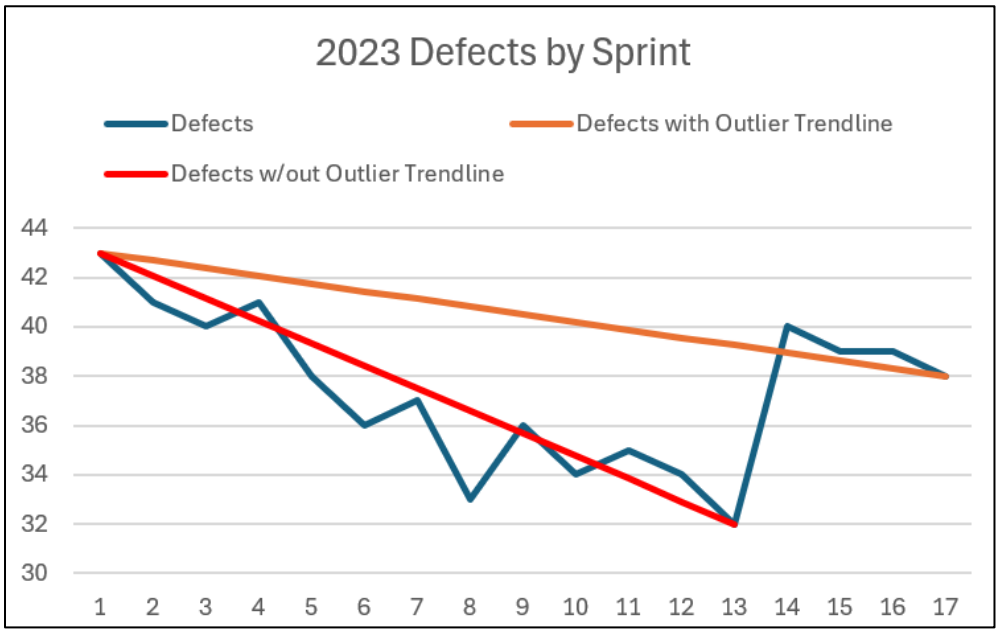

Figure 42: 2023 Reference Mode Defect and Rework Generation Trend

Customer satisfaction with the product delivered increased, or at least remained stable, throughout the 2023 project execution year. The number of Help Desk tickets remained within a 25-ticket threshold. Even after new versions were published to the production environment, there

were no dramatic upticks in Help Desk tickets noted. With less tickets being opened, the Scrum Team appeared to be able to keep up with the ticket load much more easily. Of note in the graph is the large peak associated with Help Desk tickets closed and the valley associated with the total amount of Help Desk tickets open. The team explained this peak and valley as many outstanding tickets for account creation being resolved, which has no effect on the Scrum Team, nor reflects Scrum Team performance.

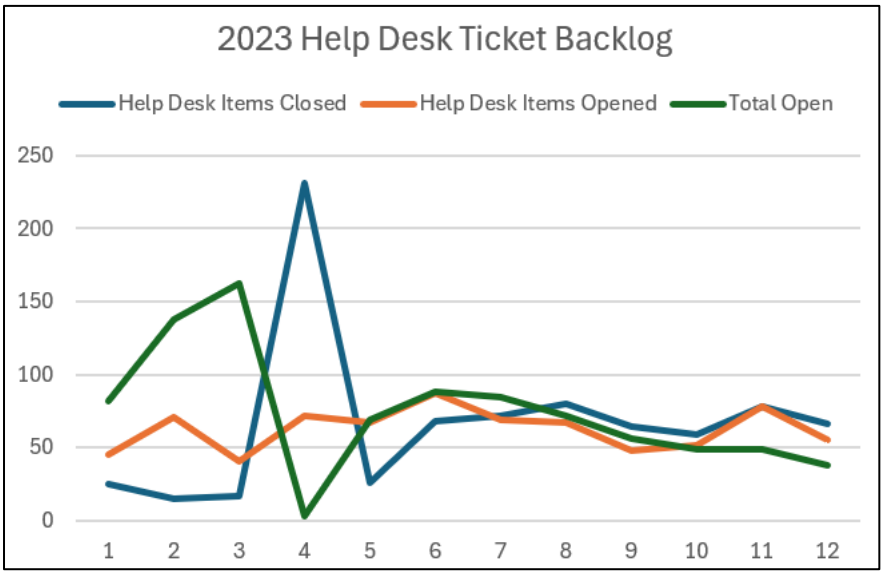

Figure 43: 2023 Reference Mode Help Desk Tickets

Simulating the Effects of Adding Systems Engineering Methods to an Agile Scrum Software Project

To simulate the effects of adding Systems Engineering methods to an Agile Scrum software project, a Vensim Systems Dynamics model, the Agile Scrum Systems Dynamics Simulation Model, was developed. As described above, this model was based upon the previously described Agile Scrum Teams execution. A random number generator was hydrated with seed data based upon the 2023 Sprint Item Quality distribution, resulting in a quality

distribution to use for comparison and simulation purposes. This allowed the input of various reference mode data from other projects to verify the validity of the simulation.

When compared to the 2023 project execution, a similar trendline is present in the Product Backlog. As the simulation reflects a scenario of perfect execution, it is not exact but does perform within a tolerance of 11.2 percent by Sprint 15 with a predicted completion of 1 Sprint early. As the Scrum Team was still in a learning environment during their time and working in the real world where unexpected challenges occur such as sick leave, this is well within expectations.

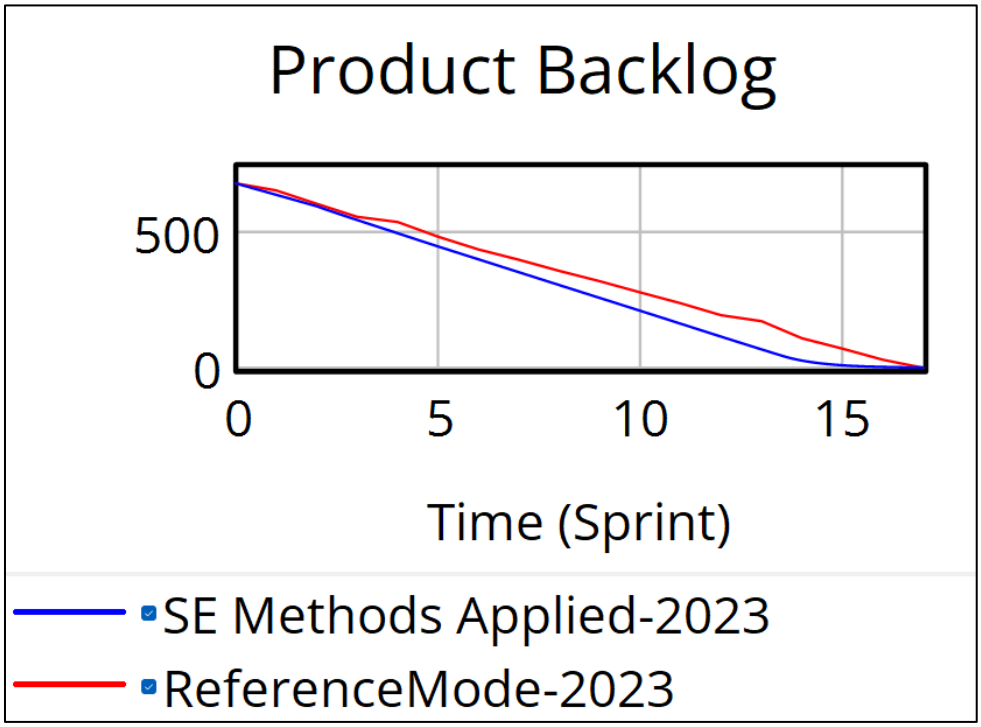

Figure 44: Agile Scrum Systems Dynamics Simulation Model 2023 Product Backlog Comparison

Sprint Planning and Sprint Work Accomplishment was also tracking stable with the actual Scrum Team execution. The planning stayed within the same threshold tolerance around the mid-forties, similar to the 2023 project execution. When the valleys for holiday vacation are

discounted, the simulation is extremely close and normally falls within a 10 percent margin, as shown in Figure 45: Agile Scrum Systems Dynamics Simulation Model 2023 Sprint Planning Comparison. The amount of work accomplished per Sprint follows this same trend, normally falling within a 10 percent margin of error for most Sprints. This is shown in Figure 46: Agile Scrum Systems Dynamics Simulation Model 2023 Sprint Work Accomplishment Comparison. Due to this, the Agile Scrum Systems Dynamics Simulation Model performs well in predicting the amount a team can work as well as the amount of work that will be accomplished.

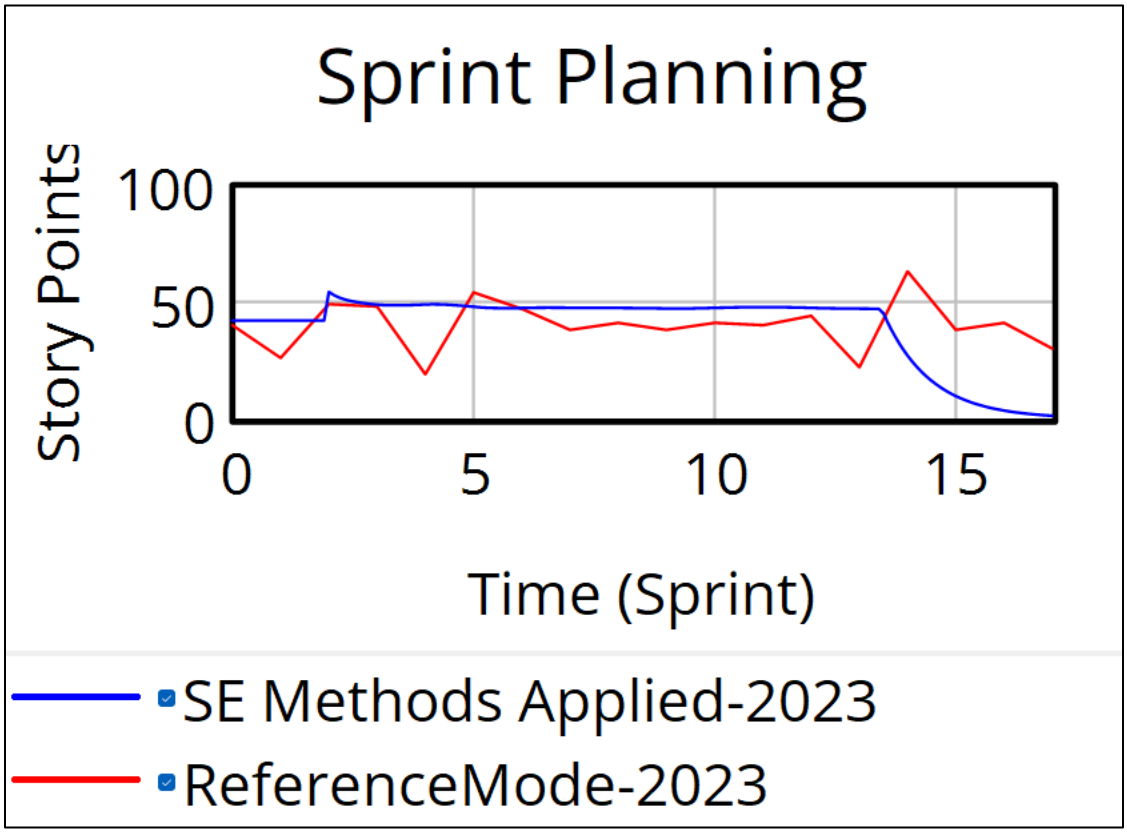

Figure 45: Agile Scrum Systems Dynamics Simulation Model 2023 Sprint Planning Comparison

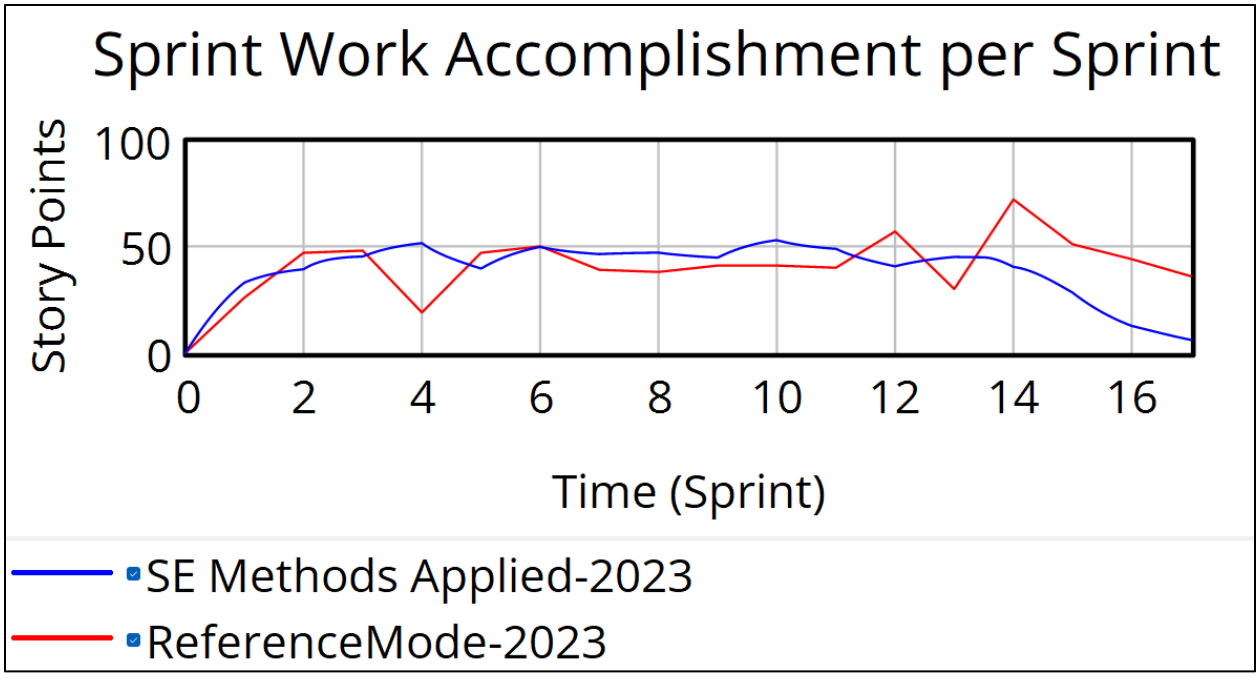

Figure 46: Agile Scrum Systems Dynamics Simulation Model 2023 Sprint Work Accomplishment Comparison

Velocity in the simulation and the 2023 reference mode was consistent. The simulation showed less peaks and valleys but remained within the 10 percent margin of the actual execution. Similarly to the other metrics, external factors affect the Velocity of the actual execution, but with such a close margin of error, this has a negligible effect on the result.

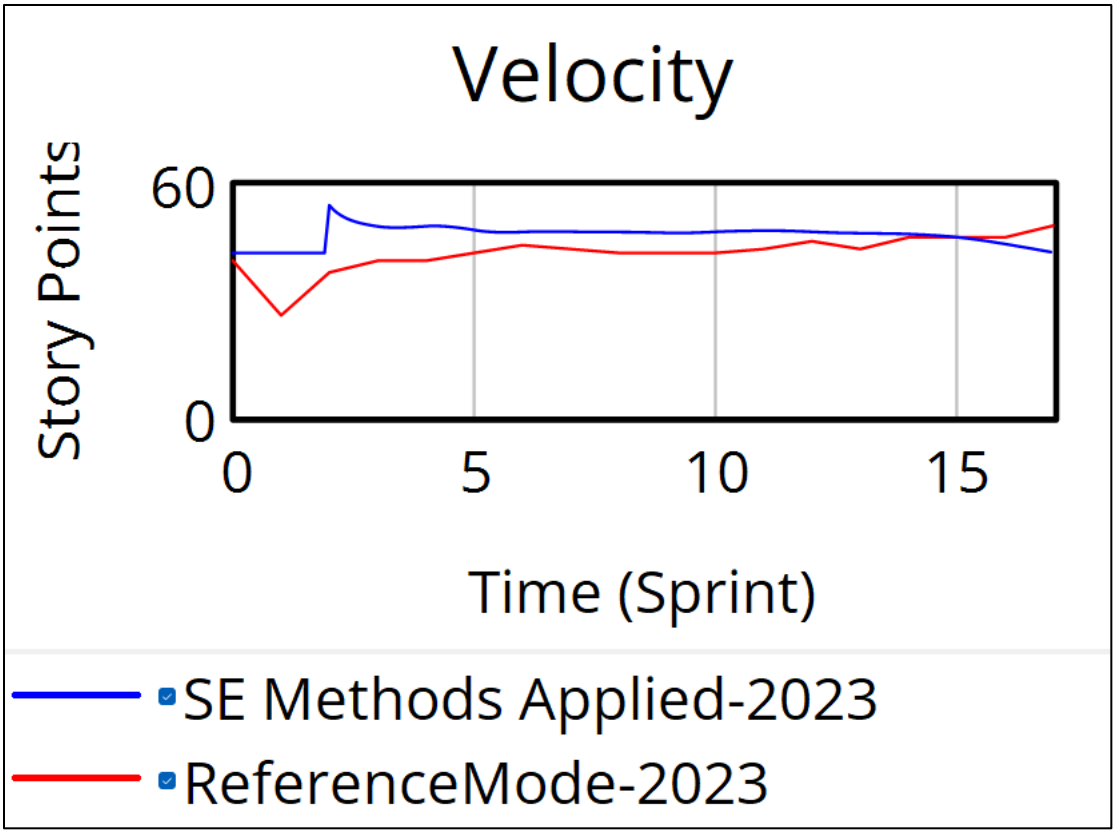

Figure 47: Agile Scrum Systems Dynamics Simulation Model 2023 Velocity Comparison

With a higher quality of Sprint work items as shown in Figure 48: Agile Scrum Systems Dynamics Simulation Model 2023 Quality of Sprint Items Comparison, less defects and rework were introduced to the system, displayed in Figure 49: Agile Scrum Systems Dynamics Simulation Model 2023 Undiscovered Defects Comparison. Of no surprise, this meant that software developers would have to do less work overall to deliver the working software product to the customer. Defects and rework are considered waste when viewed through the lens of Lean [156]. In fact, the Lean Enterprise Institute recommends considering defects to be the worst form of waste, in some situations, as they drive up costs associated with quality and drive down the value delivered to the customer [156]. By implementing Systems Engineering methods, such as requirements management, in conjunction with an Agile Scrum software execution, higher

quality Sprint Work Items are generated, which lowers the overall work necessary to deliver a working product to the customer.

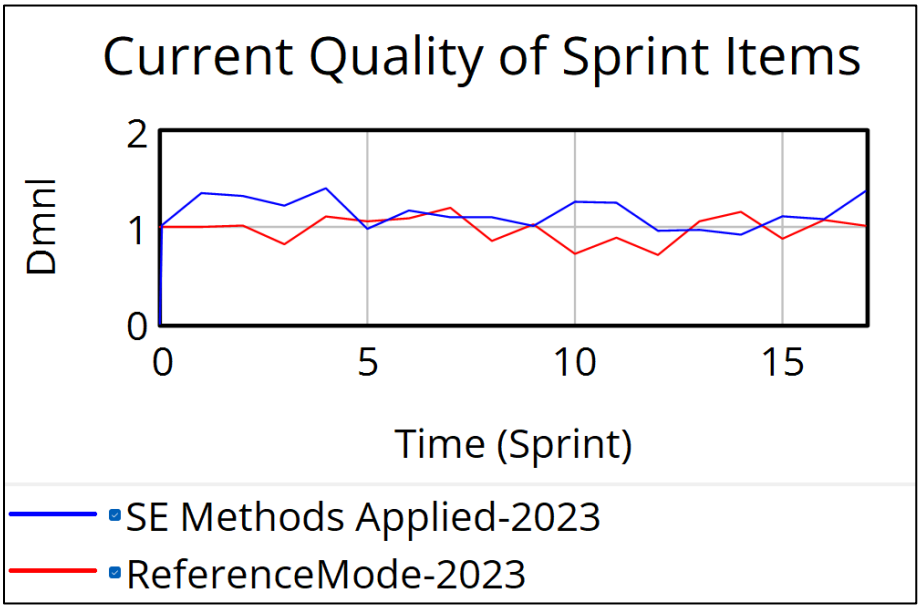

Figure 48: Agile Scrum Systems Dynamics Simulation Model 2023 Quality of Sprint Items Comparison

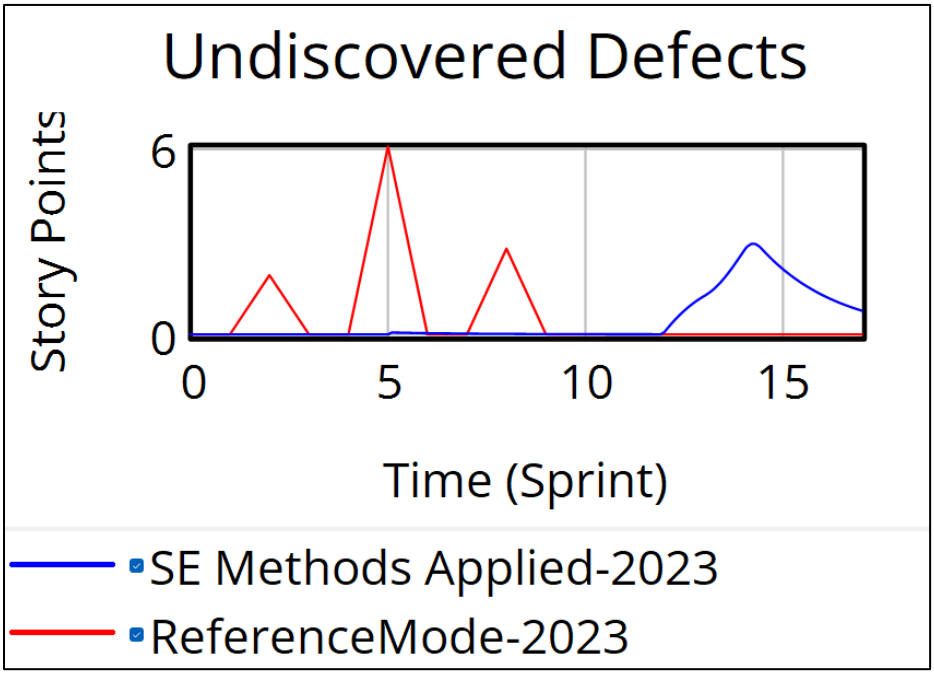

Figure 49: Agile Scrum Systems Dynamics Simulation Model 2023 Undiscovered Defects Comparison

The Agile Scrum Systems Dynamics Simulation Model successfully simulates an Agile Scrum software execution that has added additional Systems Engineering methodologies within a 10 percent margin of error. Utilizing this model, a project manager or Product Owner can predict how long an Agile software project should take and the amount of additional work that would be required.

A Comparison of the Agile Scrum Systems Dynamics Simulation Model and the 2020 Project Execution Data

As the simulation was successful in replicating the 2023 project execution, the next step in research was to compare the Agile Scrum Systems Dynamics Simulation Model to the 2020 project execution data. The hypothesis of this experiment is that given the same software team and the same amount of initial Story Points in the backlog, the Agile Scrum Systems Dynamics Simulation Model should outperform the actual 2020 Scrum Team execution. If the hypothesis is correct, the simulation should show a more stable execution environment, less defects and rework generated, and a more stable Velocity. The following text details the results of the experiment.

When the Agile Scrum Systems Dynamics Simulation Model was executed against the same initial backlog as the 2020 software project, the simulated team not only successfully finished the project on time, but also did so without major additional rework and defects being added to the Product Backlog, as shown in Figure 50: Agile Scrum Systems Dynamics Simulation Model 2020 Product Backlog Comparison. It is important to remember that meeting the scheduled deadline is a success. Finishing a project early would show poor planning and a misunderstanding of team performance. By having close to a straight-line execution, with few

peaks and valleys, the stakeholders can be confident in delivery dates and that costs will not dramatically increase over the course of the project.

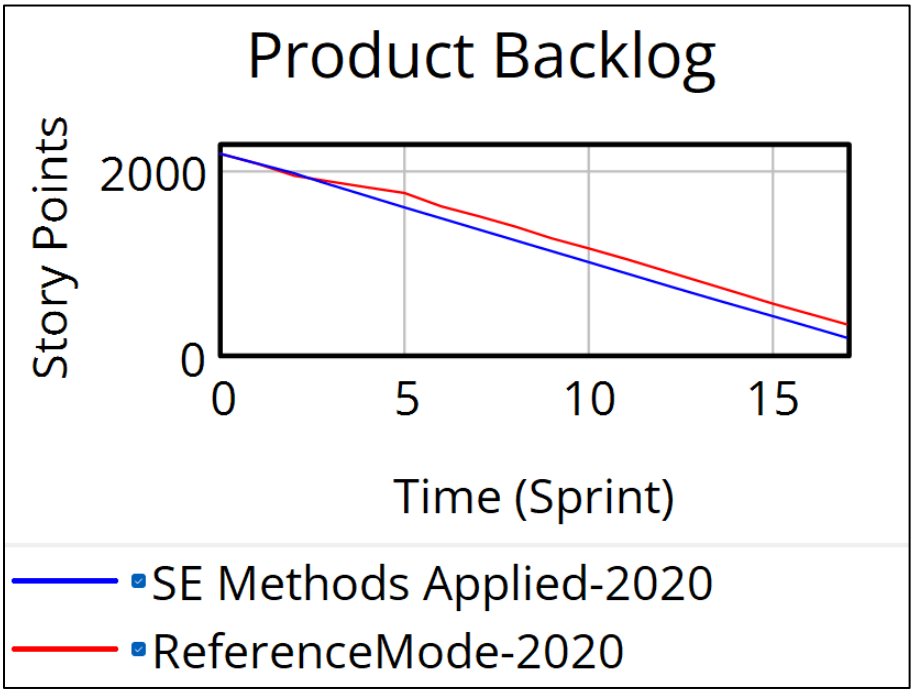

Figure 50: Agile Scrum Systems Dynamics Simulation Model 2020 Product Backlog Comparison

Sprint Planning in the Agile Scrum Systems Dynamics Simulation Model predicted a much more stable Sprint to Sprint software development execution. With a higher quality of Sprint Work Items, the team would have much better knowledge of the problem domain, allowing them to plan for and execute work per Sprints more successfully. The result of clearer problem domain understanding is evident in Figure 51: Agile Scrum Systems Dynamics Simulation Model 2020 Sprint Planning Comparison, where the 2020 reference model clearly shows large peaks and valleys in comparison to the more rigorous engineering methodologies applied in the simulation model. The Sprint Work accomplished increasingly levels out in the simulated model and did not have as prominent a valley as in the 2020 project execution. As shown in Figure 52: Agile Scrum Systems Dynamics Simulation Model 2020 Sprint Work

Accomplishment Comparison, the team is required to increasingly accomplish greater amounts of work per Sprint, as the Product Backlog increases, and the strain of meeting delivery schedules increases. This was accomplished through performing additional work during overtime hours, having a negative impact on Scrum Team quality of life. In the simulated model, the increase in Product Backlog Story Points is not present and therefore the team can continue executing at a comfortable level. Low levels of job satisfaction and general happiness with developer's job execution have a direct effect on the quality of work accomplished by the team as well as a higher motivation and sense of self-worth [157]. The ability to plan for, and execute, work without having to worry about performing rework and defect fixes is therefore not only beneficial to the company paying the developers to work, but also for the developer's mental health and job satisfaction.

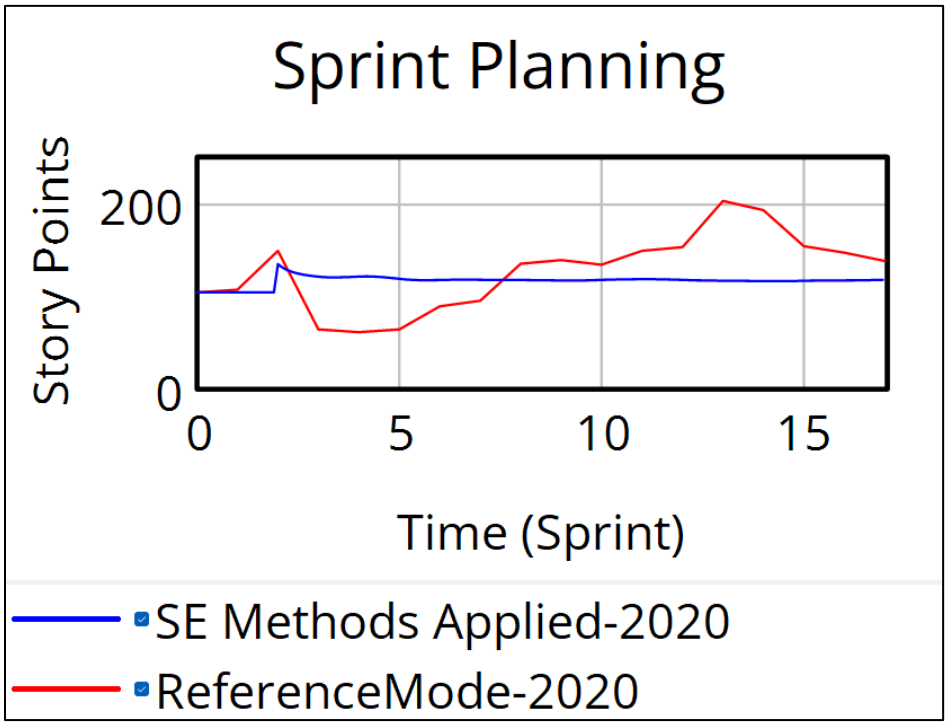

Figure 51: Agile Scrum Systems Dynamics Simulation Model 2020 Sprint Planning Comparison

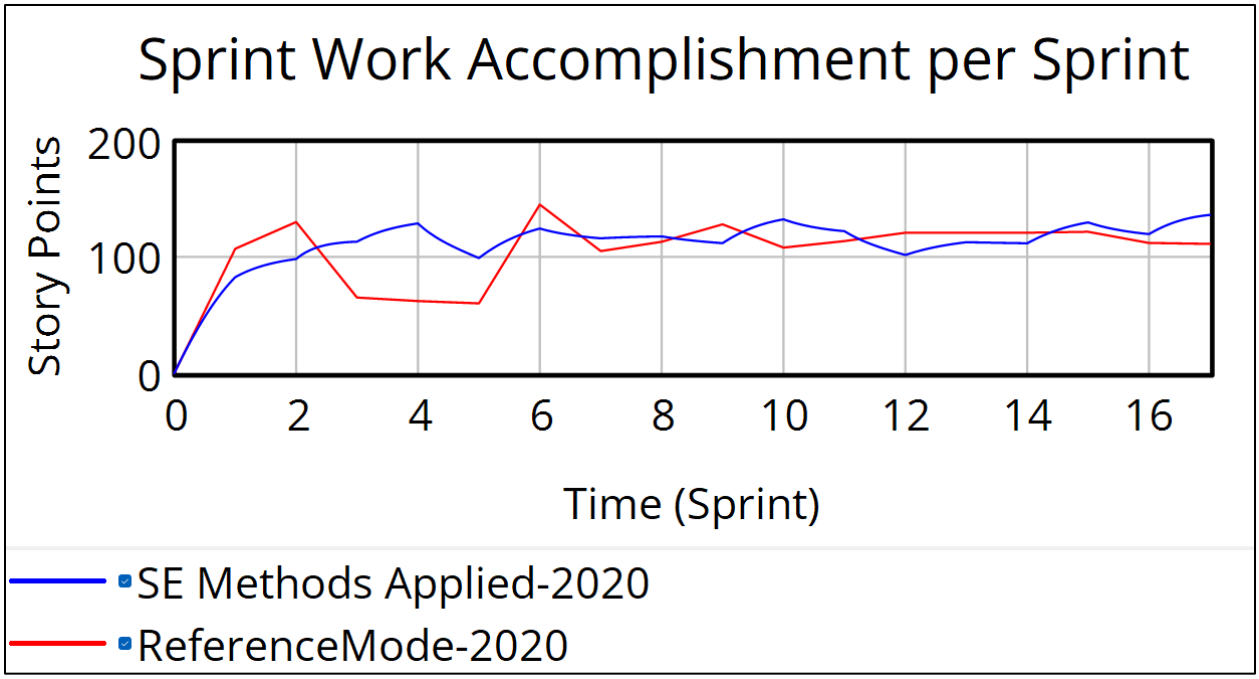

Figure 52: Agile Scrum Systems Dynamics Simulation Model 2020 Sprint Work Accomplishment Comparison

The quality of Sprint Work Items in the simulation was, on average, 21.2 percent higher than the 2020 project execution, as shown in Figure 53: Agile Scrum Systems Dynamics Simulation Model 2020 Quality of Sprint Items Comparison resulting in less defects and rework being generated, as Figure 54: Agile Scrum Systems Dynamics Simulation Model 2020 Undiscovered Defects Comparison shows. This is due to a better understanding of the problem domain and ease of communication with the customer allowing for a more clear and detailed definition of work to be accomplished. This increase in quality results in a decrease in the number of Story Points required to deliver the product. The 2020 project was only able to deliver 1658 Story Points worth of value to the customer as opposed to the predicted 1777 Story Points the simulation would deliver. As the 2020 Scrum Team had to spend time on rework, they reduced the amount of work towards the finished product. Therefore, it can be argued that an increase in Sprint Work Item quality will directly decrease the need for the Scrum Team to

perform rework and reduce the number of defects generated during project execution, increasing the total value delivered to the customer of the course of a project.

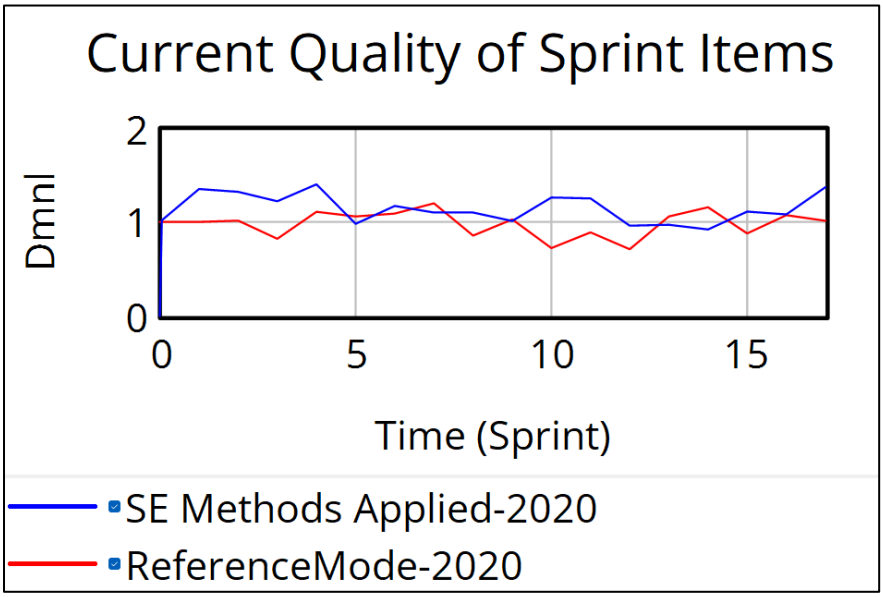

Figure 53: Agile Scrum Systems Dynamics Simulation Model 2020 Quality of Sprint Items Comparison

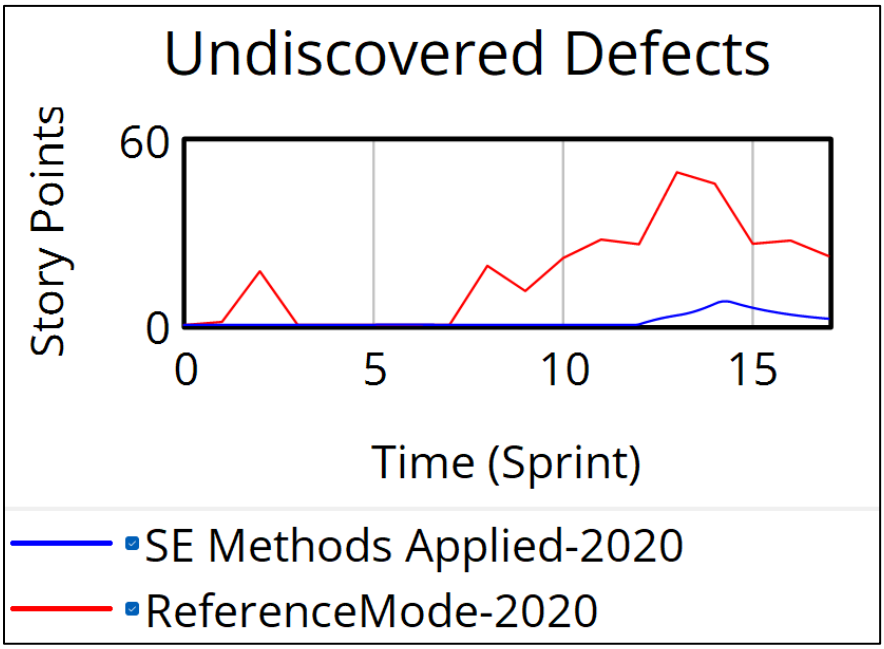

Figure 54: Agile Scrum Systems Dynamics Simulation Model 2020 Undiscovered Defects Comparison

Simulation results predict an improvement in total Story Point completion when applying systems engineering methods with Agile Scrum versus a pure Agile Scrum execution. With the application of systems engineering methods, the simulation predicted a closing Story Point completion value of 1,879 Story Points. In comparison, the development team completed 1,658 Story Point in 2020 working in a pure Agile Scrum environment with little engineering or technical rigor. This is an 11.8% increase in completed software delivered to the customer. The simulation initially underperforms the actual 2020 product execution, but this is easily explained as the developers initially worked on the clearest and understood work items, leaving the more difficult and less understood work items for later in the project. As understanding of the system and problem domain lessened, so too did the amount of completed work delivered, as shown in Figure 55: Agile Scrum Systems Dynamics Simulation Model 2020 Work Complete Comparison.

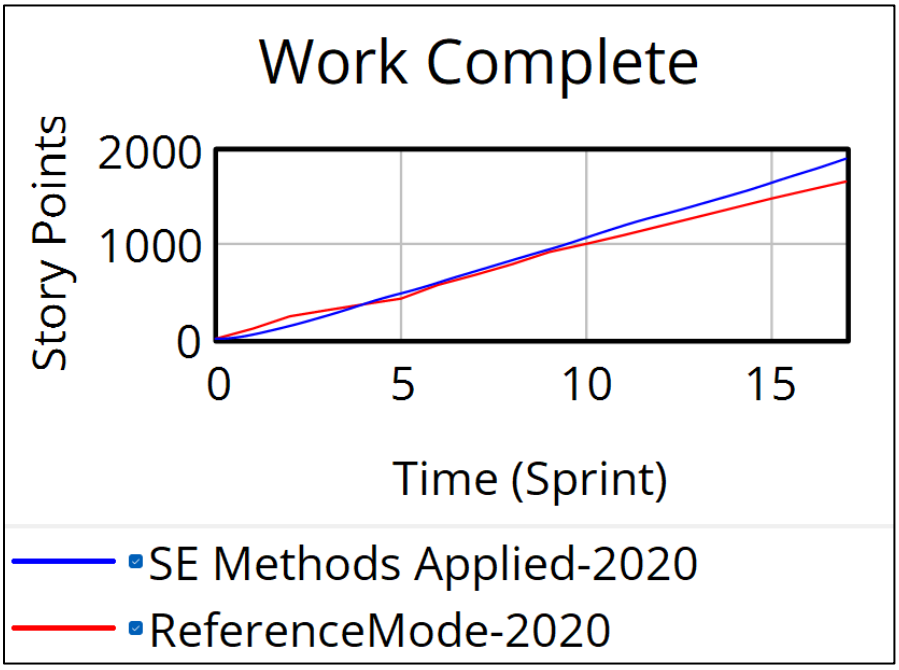

Figure 55: Agile Scrum Systems Dynamics Simulation Model 2020 Work Complete Comparison

The example project data shown above is indicative that applying Systems Engineering methods in conjunction with the Agile Scrum framework will increase software product quality resulting in a reduction of defects and rework during project execution. With the ability to simulate project execution, project managers, stakeholders, and Product Owners can better predict project timelines and effort required to deliver to the customer. Of important note is the decrease in stress reported by software developers executing in the merged Systems Engineering and Agile Scrum environment as there were fewer heroic efforts required to deliver the product.

From this documentation, further studies can be performed to increase the understanding of this effect. By gathering and documenting data for other software projects employing Systems Engineering methodologies and then using that data to create reference modes for the simulation, the Agile Scrum Systems Dynamics Simulation Model can be further refined to create an even higher quality simulation model. With more reference mode data to refer to, the simulation model becomes ever more capable of predicting software project execution patterns.

INCOSE has a large amount of research in Agile Systems Engineering, which is defined as applying Agile frameworks and methods to projects and programs that use a Systems Engineering approach. Applying Systems Engineering methods and rigor to the Agile Scrum framework is a new area of research. Many in the Agile community disregard the benefits of engineering rigor, documentation, planning, and modeling. In fact, the Agile Manifesto specifically calls out against many of those methods and efforts [8]. Agile practitioners often also do not fully understand the complexities of Agile methods, often taking quick training and assuming they know all there is to know in execution rather than understanding that the training is but the first step in a learning process. Worse still are the Agile practitioners who do not tailor their executions to their specific projects and teams. It should be clear that many within the Agile

community are trained to disregard engineering rigor or just lack the desire for work outside of minimal system documentation. This has directly led to a community that creates systems without historical decision-making documentation, framework models, or in-depth requirements tracking. Therefore, research into applying Systems Engineering rigor is next to non-existent in the Agile community. There have been some efforts to introduce Systems Engineering rigor into Agile at Scale methods, but this is simply applying the rigor to a larger program overseeing multiple smaller projects, not integrating Systems Engineering methods into the actual execution occurring on the lowest levels of the Agile at Scale executions [158] [159] [160].

Modeling and Simulation - Conclusions

How does the unconstrained and dynamic structure of Agile projects drive the behavior of project successes and failures in software projects? The ability for Agile projects to change each work iteration is central to Agile project execution. As Agile projects incorporate frequent changes, having backlogs of high-quality work items is key to success. This work item quality is directly driven by customer communication, understanding of stakeholder needs, goals, and requirements, and the ability to conduct rigorous Verification and Validation activities. As in all projects, communication is key and drives a mutual understanding between the customer and the development team. Without effective communication, misunderstandings of needs, goals, and requirements cannot be clarified, nor can they be documented during elicitation. The documentation of needs, goals, and requirements using unambiguous language, in a standardized manner, and as atomic as possible, allows a better understanding of the system and problem domain that developers are performing to. The addition of modeling efforts to clarify requirements, visualize the system domain, increase the ability to understand and predict emergent behaviors, and assist in communication between customers, stakeholders, and

developers increases work item quality. Finally, high levels of work item quality, in the form of clear, concise, and consistent requirements management allows test engineers to better understand what, how, and why work items should be tested and allows for the capability to fully understand, and document test coverage traced to customer needs, goals, and requirements.

As shown in the Agile Scrum Systems Dynamics Simulation Model results, the thoughtful addition of Systems Engineering methodologies to an Agile Scrum project execution has positive benefits for software projects. Sprint Planning and project execution is smoother and less defects are generated. Help Desk tickets are stable without major increases in new ticket creation, even after the release of new versions of software to the production environment. In a qualitative measure, customers and developers also report an ease in communication and understanding of the problem domain. Overall, the benefits of increased rigor are high with few drawbacks noted.

CHAPTER 5 – IMPROVEMENTS TO AGILE SCRUM THROUGH SYSTEMS ENGINEERING FOCUSED AGILE DEVELOPMENT

This chapter seeks to answer research question 4, which is stated here as follows:

Question: How can Agile Scrum project implement Systems Engineering methodologies without losing the ability to take direct customer feedback and requirements changes to meet a customer driven definition of done?

To answer this question, a new methodology is needed which maintains the ability to quickly pivot as customer feedback is given and assessed, considering frequent changes in requirements, while still tracking requirements and documenting system state. The proposed process should focus on technical rigor at the beginning of the project while initial capabilities are generated and requirements gathered, transition to a flexible and highly Agile methodology during software execution, allowing for swift changes in requirements and architecture, and then transitioning back to a rigorous framework to close out the project. This will allow software developers to communicate with cross-functional teams and customers effectively while still working in an environment that can be adapted easily to changing customer driven requirements.

Sub-question 4.1: Where are systems engineering methods most impactful to drive positive impacts to DoD Agile software projects?

Through the data collected in performing the case studies and through the knowledge gained from the Agile Scrum simulation models, it is apparent that the most influential systems engineering methods are the following:

1. Requirements Management: Performing structured, systems engineering style, requirements management increased understanding of the system context as well as facilitating customer communication.

2. Customer Collaboration: Creating, instilling, and ensuring a close and collaborative relationship between stakeholders and development engineers directly increases understanding of the system context and problem domain, leading to a more successful product viewed through the lens of customer satisfaction.
3. Documentation and Technical Rigor: Ensuring documentation and tracking of needs, goals, and requirements, as well as other technical documents, increases understanding of the system context and problem domain, directly increasing the ability for development engineers to execute new projects and sustain existing projects.
4. Modeling: Implementation of Model Based Systems Engineering activities, such as SySML and Wireframe Modeling, increases the ability of stakeholders and developers to communicate ideas and collaborate on functionality, increasing the understanding of systems context and the problem domain.

Sub-question 4.2: How has merging systems engineering methods with the Agile Scrum Framework affected software project execution?

Data collected from software projects executing in an environment that utilizes systems engineering methodologies along with the Agile Scrum Framework has shown that the development team gains a higher understanding of customer needs, goals, and requirements. This directly leads to an increase in system context and problem domain knowledge, as well as facilitating clearer communication with stakeholders. The goal in software projects is to deliver a product with limited scope creep and within budget while lowering stress on the development team and customers [161] [162]. The thoughtful merger of systems engineering methods with the Agile Scrum Framework is an enabling factor for this.

Systems Engineering Focused Agile Development (SEFAD) Process

The following process was developed to create a project execution environment merging the rigor of systems engineering methodologies with the flexibility of the Agile Scrum Framework.

1. OBJECTIVE:

    1.1 To develop a process that can be followed by engineering efforts across multiple programs or projects.

    1.2 To analyze customer needs and translate needs into requirements.

    1.3 To design, develop, and implement solutions to meet the customer's needs and requirements. Solutions will encompass products, product components, and product-related lifecycle processes, either singly or in combination as appropriate.

    1.4 To assemble and integrate the solution into a deliverable, supportable, and well-engineered product.

    1.5 To verify and validate product requirements through rigorous testing.

2. PROCESS BOUNDARIES:

    2.1 This process begins with the project initiating phase to ensure process flow and continuity throughout the product lifecycle.

    2.2 This process ends when the product has been delivered and the project closeout has been performed.

3. BACKGROUND**:**

    3.1 The Systems Engineering Focused Agile Scrum (SEFAD) process is a software development methodology that leverages the rigor of systems engineering methodologies, such as comprehensive requirements management, with the flexibility and adaptability of

Agile Scrum. Systems Engineering is a transdisciplinary and integrative approach to enable the successful realization, use, and retirement of engineered systems, using systems principles and concepts, and scientific, technological, and management methods [163]. Agile Scrum is an empirical process, where decisions are based on observation, experience, and experimentation. Scrum has three pillars: transparency, inspection, and adaptation. This supports the concept of working iteratively [115]. This process includes all stages of the product lifecycle, from initiating phase until sustainment phase. Although it is outside the scope of this directive, it is recommended that programs have a Systems Engineering Plan (SEP).

To better communicate the goal of SEFAD, the development of a system follows a rigid process from instantiation through requirements and architecture development, as shown in Figure 56: SEFAD V-Model. Once the PDR has been successfully accomplished, the process transitions into a flexible, Agile development cycle, which continues until the system has passed the CDR. Once the CDR is completed, all requirements are baselined, and the system process once again becomes rigid. No new requirements will be added. Remaining unaccomplished requirements are developed and defects fixed. This continues until the system is ready for the Production Readiness Review and passing into delivery.

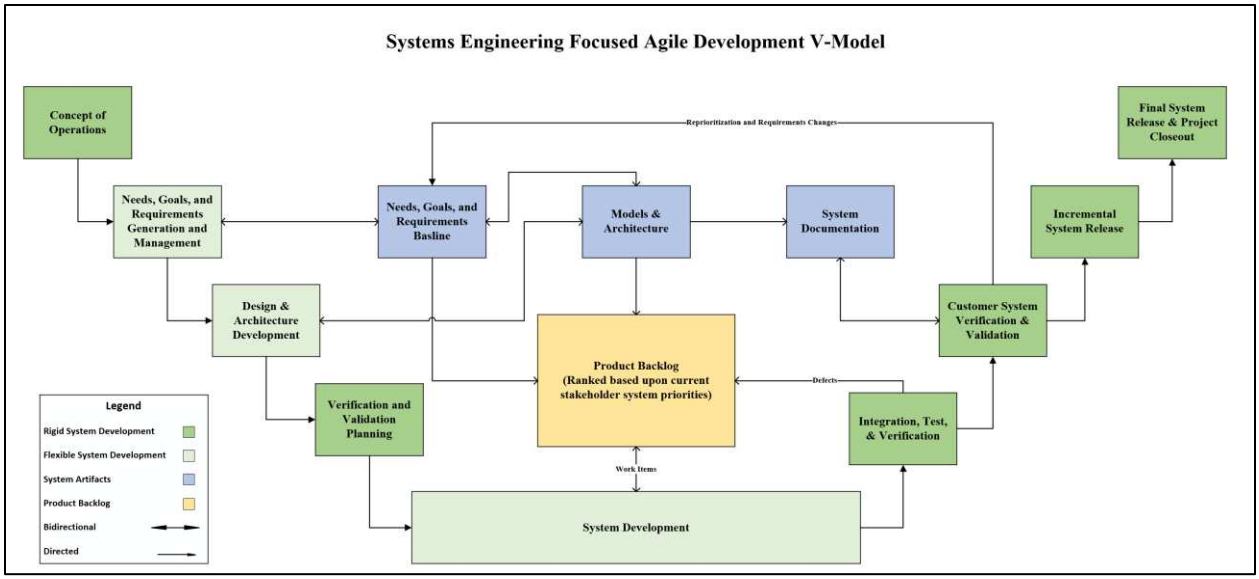

Figure 56: SEFAD V-Model Illustrating Rigidity and Flexibility in Software Development

4. POLICY:

4.1 All major technical reviews will be conducted according to guidance from the customer/Program Office. For example, PEO IWS 5 program office uses the PEO IWS Technical Review Manual for technical review guidance. Other Program Offices may use technical review guidance documents more specific to the design/development work being accomplished.

4.2 Tailoring Guidelines: This framework dictates the minimum effort required for development of systems. Some projects, being more complex than others, may add items, but all items as detailed in this document shall be completed for projects. The entrance and exit criteria for the decision points shall be documented in every case, tailored to fit the project, and shall become a project artifact that is signed off by an "Approving Authority" and retained as a permanent record for the project in accordance with the Configuration Management (CM) Plan.

5. DEFINITIONS AND KEY TERMS:

Table 6: Definition and Key Terms

CCB	Configuration Control Board
CDR	Critical Design Review – Decision Point (System Level)
CM	Configuration Management
Decision Point	Administrative/Technical Review of project performance based upon fixed entrance and exit criteria. Culminates with a decision memorandum to continue to the next phase or iteratively rework the design.
DDP	Design Dependent Parameter. Design characteristics inherent in the product or system such as reliability, maintainability, and disposability.
DIP	Design Independent Parameter. Design characteristics that are not controlled by the product or system such as labor rates, material costs, energy costs, etc.
FDR	Final Design Review
GUI	Graphical User Interface
IDR	Initial Design Review
MVP	Minimum Viable Product. The minimum product that can be delivered to a stakeholder to meet stakeholder requirements.
Need Statement	A stakeholder's need is a high-level representation of a capability or objective the stakeholder would like to accomplish. It establishes the relationship between the organization and the customer. It is something that the stakeholders want or which the stakeholders feel is necessary. It is written in plain English and utilizes the keyword "need." Example: *"The business needs to be able to generate reports that detail our revenue stream so that we can report to our board of directors concerning the company's profitability."*

PDR	Preliminary Design Review – Decision Point (System Level)
PO	Product Owner
PRR	Production Readiness Review – Decision Point
Requirements Statement	Requirements statements represent things that are needed to be accomplished to achieve a stakeholder need or goal. They are written in “shall” statements and can be broken down into lower-level increments to give further clarification. A requirement should state who shall (do, perform, provide, weight, or other verb) followed by a description of what should be performed and why it is being performed, if known. Example: *“The software shall generate reports that detail the company’s revenue stream so that the customer can meet board of director mandated reporting requirements.”*
SE	System Engineering. A transdisciplinary and integrative approach to enable the successful realization, use, and retirement of engineered systems, using systems principles and concepts, and scientific, technological, and management methods.
SEFAD	Systems Engineering Focused Agile Scrum
SNR	Stakeholder Needs Review
SND	Stakeholder Needs Document. An official statement set of stakeholders needs that have been communicated by the stakeholders to represent what stakeholders want, or feel is necessary, to accomplish their goals and meet their objectives.
SM	Scrum Master
SME	Subject Matter Expert
SRD	System Requirement Document. An official statement set of system requirements that have been negotiated and agreed to for stakeholders, customers, end-users, and clients.
SRR	System Requirements Review – Decision Point (System Level)

TPM	Technical Project Manager - Individual
TPM-A	Technical Performance Measure – Attribute. A technique that measures the risks inherent in a technical system element to determine how well that element is satisfying requirements. This normally is the method of measurements of performance at specific points in a program and the variation limits for corrective action.
WBS	Work Breakdown Structure
Work Item	Decomposed elements of a project plan that are tasked to project members for completion.

6. RESPONSIBILITIES:

6.1 Stakeholder/Program Office

6.1.1 Develop initial needs and high-level requirements.

6.1.2 Provide guidance throughout the development process to ensure the derived requirements satisfy the high-level requirements and customer needs.

6.1.3 Monitor project to ensure completed on time and within budget.

6.2 Technical Project Manager (TPM) or delegated authority:

6.2.1 TPM leads decision point reviews and signs decision memorandums if authority has been delegated by Program Office.

6.2.2 Schedule and host decision point reviews.

6.3 Systems Engineer (SE)

6.3.1 The Systems Engineer develops, designs, allocates, and manages stakeholder needs and system-level requirements, leads the development of the system architecture, evaluates design tradeoffs, balances technical risk between systems, defines and assesses interfaces, provides oversight of verification and validation activities, as well as many other tasks throughout the course of a program.

6.3.2 The Systems Engineer's responsibilities include:

- Developing and executing the system CDR plans with established quantifiable review criteria, carefully tailored to satisfy program objectives.
- Ensuring the pre-established review criteria have been met to ensure the design has been captured in the allocated baseline and initial product baseline.
- Ensuring assessments and risks associated with all design constraints and considerations are conducted, documented, and provided (e.g., HSI, reliability and maintainability, corrosion, SS, and ESOH considerations).
- Updating RIO plans. Identifying, analyzing, mitigating, and monitoring risks and issues; and identifying, analyzing, managing, and monitoring opportunities. (See the DoD Risk, Issue, and Opportunity Management Guide for Defense Acquisition Programs.) Monitoring and controlling the execution of the CDR closure plans.
- Documenting the plan to SVR in the SEP and elsewhere as appropriate.

6.4 Product Owner:

6.4.1 The Product Owner is accountable for maximizing the value of the product resulting from the work of the Scrum Team [9].

6.4.2 Conduct overall day-to-day management of the project to include the following [9]:

6.4.2.1 Develop, maintain, and explicitly communicate the Product Goal.

6.4.2.2 Create and clearly communicate Product Backlog items**.**

6.4.2.3 Order and prioritize Product Backlog items.

6.4.2.4 Ensure the Product Backlog is transparent, visible, and understood.

6.4.2.5 The Product Owner is one person, not a committee. The Product Owner may represent the needs of many stakeholders in the Product Backlog. Those wanting to change the Product Backlog can do so by trying to convince the Product Owner.

6.4.2.6 Translate Customer/Program Office strategies and needs to tasks for development.

6.4.2.7 Maintain knowledge of the market and customer needs.

6.4.2.8 Serve as a liaison between the Customer/Program Office and the development team.

6.4.2.9 Be accessible to the development team to answer questions related to the product.

6.5 Scrum Master:

6.5.1 The Scrum Master is accountable for establishing Scrum as defined in the Scrum Guide. They do this by helping everyone understand Scrum theory and practice, both within the Scrum Team and the organization [9].

6.5.2 The Scrum Master is accountable for the Scrum Team's effectiveness. They do this by enabling the Scrum Team to improve its practices, within the Scrum framework [9].

6.5.3 The Scrum Master serves the Scrum Team in several ways, including [9]:

6.5.3.1 Coaching the team members in self-management and cross-functionality.

6.5.3.2 Helping the Scrum Team focus on creating high-value Increments that meet the Definition of Done.

6.5.3.3 Causing the removal of impediments to the Scrum Team's progress.

6.5.3.4 Ensuring that all Scrum events take place and are positive, productive, and kept within the timebox.

6.5.4 The Scrum Master serves the Product Owner in several ways, including [9]:

6.5.4.1 Helping find techniques for effective Product Goal definition and Product Backlog management.

6.5.4.2 Helping the Scrum Team understand the need for clear and concise Product Backlog items.

6.5.4.3 Helping establish empirical product planning for a complex environment.

6.5.4.4 Facilitating stakeholder collaboration as requested or needed.

6.5.5 The Scrum Master serves the organization in several ways, including [9]:

6.5.5.1 Leading, training, and coaching the organization in its Scrum adoption.

6.5.5.2 Planning and advising Scrum implementations within the organization.

6.5.5.3 Helping employees and stakeholders understand and enact an empirical approach for complex work.

6.5.5.4 Removing barriers between stakeholders and Scrum Teams.

6.6 Developers:

6.6.1 Developers are the people in the Scrum Team that are committed to creating any aspect of a usable Increment each Sprint [9].

6.6.2 The specific skills needed by the Developers are often broad and will vary with the domain of work [9].

6.6.3 The Developers are always accountable for [9]:

6.6.3.1 Creating a plan for the Sprint.

6.6.3.2 Creating and maintaining the Sprint Backlog.

6.6.3.3 Instilling quality by adhering to a Definition of Done.

6.6.3.4 Adapting their plan each day toward the Sprint Goal.

6.6.3.5 Holding each other accountable as professionals.

7. REVIEWS:

7.1 Stakeholder Needs Review (SNR)

7.1.1 The Stakeholder Needs Review is a multi-disciplined technical review of the stakeholders' needs.

7.1.2 The goal of this review is to ensure that all initial stakeholder needs are documented and that the project is ready to move on to system requirements definition.

7.1.3 Question to answer: Are all stakeholders' needs accounted for in the Stakeholder Needs Registry and is the system ready to develop system level requirements?

7.2 **System Requirements Review (SRR)**

7.2.1 The SRR is a multi-disciplined technical review to ensure that the developer understands the system requirements and is ready to proceed with the initial system design [125].

7.2.2 Question to Answer: Are all stakeholder needs accounted for in the System Requirements Document and is the system ready to develop system level requirements?

7.2.3 For Acquisition Programs: A SRR is mandatory per DoDI 5000.88, Section 3.5.a. This review assesses whether the system requirements as captured in the system performance specification (sometimes referred to as the System Requirements Document (SRD)) [125]:

- Are consistent with the preferred materiel solution (including its support concept).
- Are consistent with technology maturation plans.
- Adequately consider the maturity of interdependent systems.
- Are clearly stated and are measurable and testable**.**

7.3 Initial Design Review (IDR)

7.3.1 The IDR is the first lightweight review held with the stakeholders. At the IDR, the Stakeholder and Product Owner will review the SND and SRD, as well as initial system models and T&E plans. After a review of the initial system design, the decision will be made as to whether the system as proposed will meet stakeholder needs.

7.3.1.1 Question to answer: Will the proposed system meet stakeholder needs?

7.3.2 For Acquisition Programs: This should be held in the form of a formal PDR, mandatory for MDAPs per DoDI 5000.88, Section 3.5.a. The PDR should provide sufficient confidence to proceed with detailed design. The PDR determines whether the preliminary design and basic system architecture are complete, that there is technical confidence the capability need can be satisfied within cost and schedule goals, and that risks have been identified and mitigation plans established. It also provides the acquisition community, end user, and other stakeholders with an opportunity to understand the trade studies conducted during the preliminary design, and thus confirm that design decisions are consistent with the user's performance and schedule needs and applicable requirements documents. The PDR also establishes the allocated baseline [125].

7.3.2.1 See the Systems Engineering Guidebook for all required PDR activities [125].

7.4 **Sprint Review** [9]

7.4.1 The purpose of the Sprint Review is to inspect the outcome of the Sprint and determine future adaptations. The Development Team presents their work to key stakeholders and progress toward the Product Goal is discussed.

7.4.2 During the event, the Scrum Team and stakeholders review what was accomplished in the Sprint and what has changed in their environment. Based on this information, attendees collaborate on what to do next. The Product Backlog may also be adjusted to meet new opportunities, as well as the Stakeholder Needs Register and System Requirements Document. The Sprint Review is a working session, and the Scrum Team should avoid limiting it to a presentation.

7.5 **Publish Authorization Review**

7.5.1 The purpose of the Publish Authorization Review is to inspect the current outcome of the Sprint and the state of the System Increment. The Development Team presents their work to key stakeholders with the goal of a decision to publish the current System Increment.

7.5.2 A Sprint Review is part of the Publish Authorization Review. The Development Team and stakeholders review what was accomplished in the Sprint and what has changed in the system environment. Based on this information, attendees collaborate on what to do next. The Product Backlog may also be adjusted to meet new opportunities, as well as the Stakeholder Needs Register and System Requirements Document.

7.5.3 With stakeholder agreement and approval, a release publication authorization form is signed authorizing the publish of the current System Increment. The approved System Increment is published to the system production environment. This is especially useful if utilizing a CI/CD pipeline and has a high chance of returning much needed stakeholder feedback.

7.6 **Final Design Review (FDR):**

7.6.1 At the FDR, the Stakeholder and Product Owner decide whether the system has reached a critical point of completion in that the pathway to final product delivery is obvious. Depending on the project, this could happen as early as 75% completion to as late as 95% completion.

7.6.1.1 Question to answer: Is the system sufficiently complete and documented to move into Project Finalization?

7.6.1.2 NOTE: *This will result in a Stakeholder Needs and System Requirements freeze. No new work items, other than defects or emergent changes, shall be added to the Product Backlog for work.*

7.6.2 For Acquisition Programs: This should be held in the form of a formal CDR as is mandatory for MDAPs per DoDI 5000.88, Section 3.5.a. The CDR confirms the system design is stable and is expected to meet system performance requirements, confirms the system is on track to achieve affordability and should-cost goals as evidenced by the detailed design documentation and establishes the initial product baseline [125].

7.6.2.1 See the Systems Engineering Guidebook for all required CDR activities [125].

7.7 **Production Readiness Review (PRR)**

7.7.1 At the PRR, the Stakeholder and Product Owner decide whether the system is ready for final product delivery.

7.7.1.1 Question to answer: Is the system sufficiently complete that stakeholder needs are met satisfactorily enough to finalize the project and move to sustainment?

7.7.1.2 NOTE: *This does not mean that all requirements have been developed or that the product backlog is empty. This means that the stakeholder is satisfied with the current state of the system as a working viable product. Remaining requirements and work items will be triaged and packaged to be transferred for sustainment efforts.*

7.7.2 For Acquisition Programs: The PRR, mandatory per DoDI 5000.88, Section 3.5.a. determines whether the system design is ready for production, and whether the developer has accomplished adequate production planning for entering Low-Rate Initial Production (LRIP) and Full-Rate Production (FRP). Production readiness increases over time with incremental assessments at various points in the program life cycle [125].

7.7.3 See the Systems Engineering Guidebook for all required PRR activities [125].

8. PROCESS FLOW: The process flow is highlighted in the following four (4) figures. Note these are typical engineering review Decision Points and Work Items. The intent is not to attempt to address every conceivable combination of tailorable Decision Points and Work Items, but to provide a basis for informed decisions. The bottom line is the desire for a comprehensive, iterative, recursive, and well documented system engineering problem

solving process. Immediately following the SEFAD process diagrams is Table 7: Process Flow Details that contains detail information on each of the items addressed in the process diagrams.

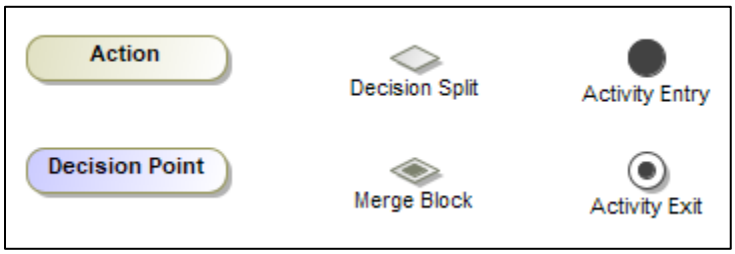

Figure 57: SEFAD Process Key

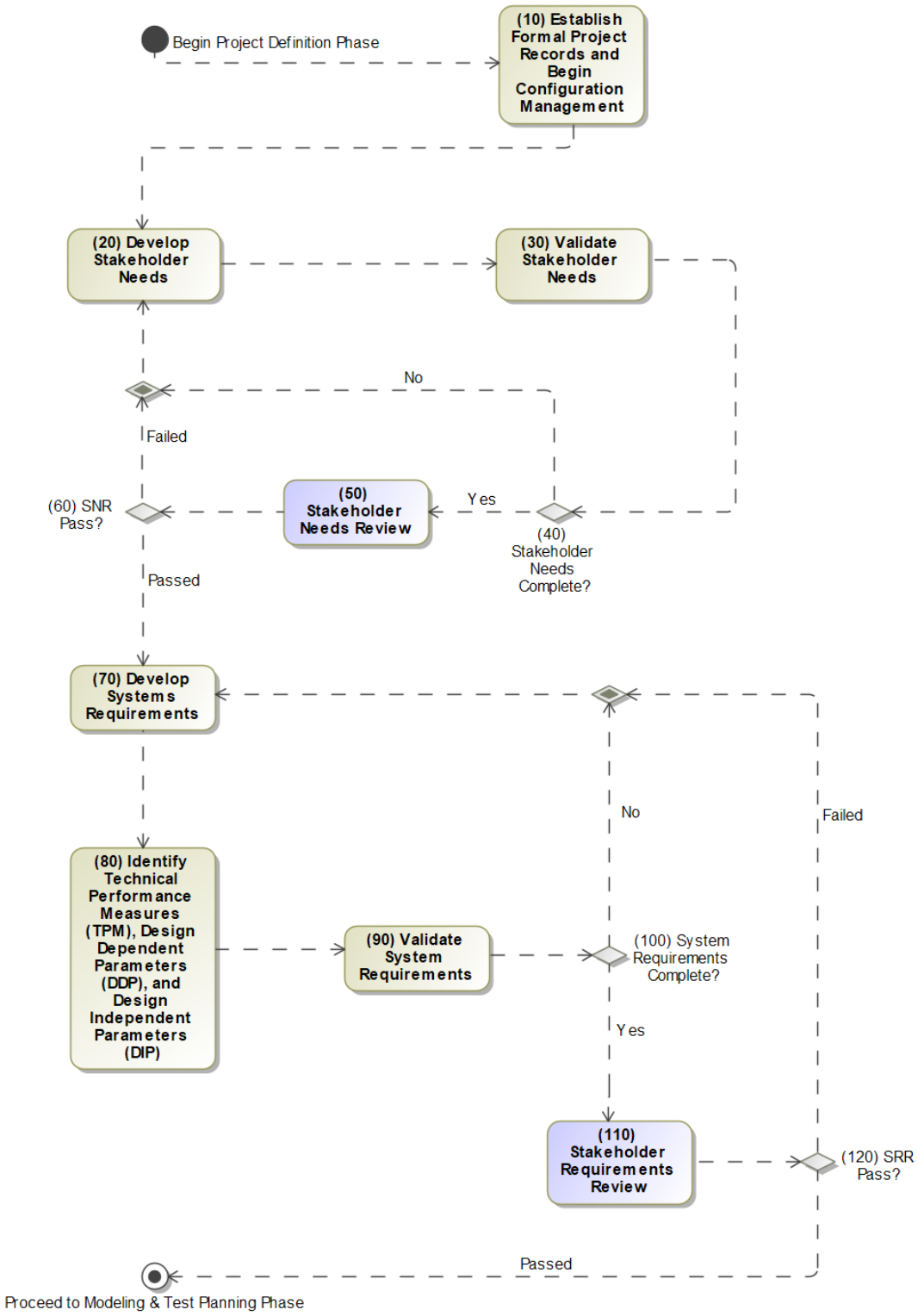

Figure 58: SEFAD Project Definition Phase

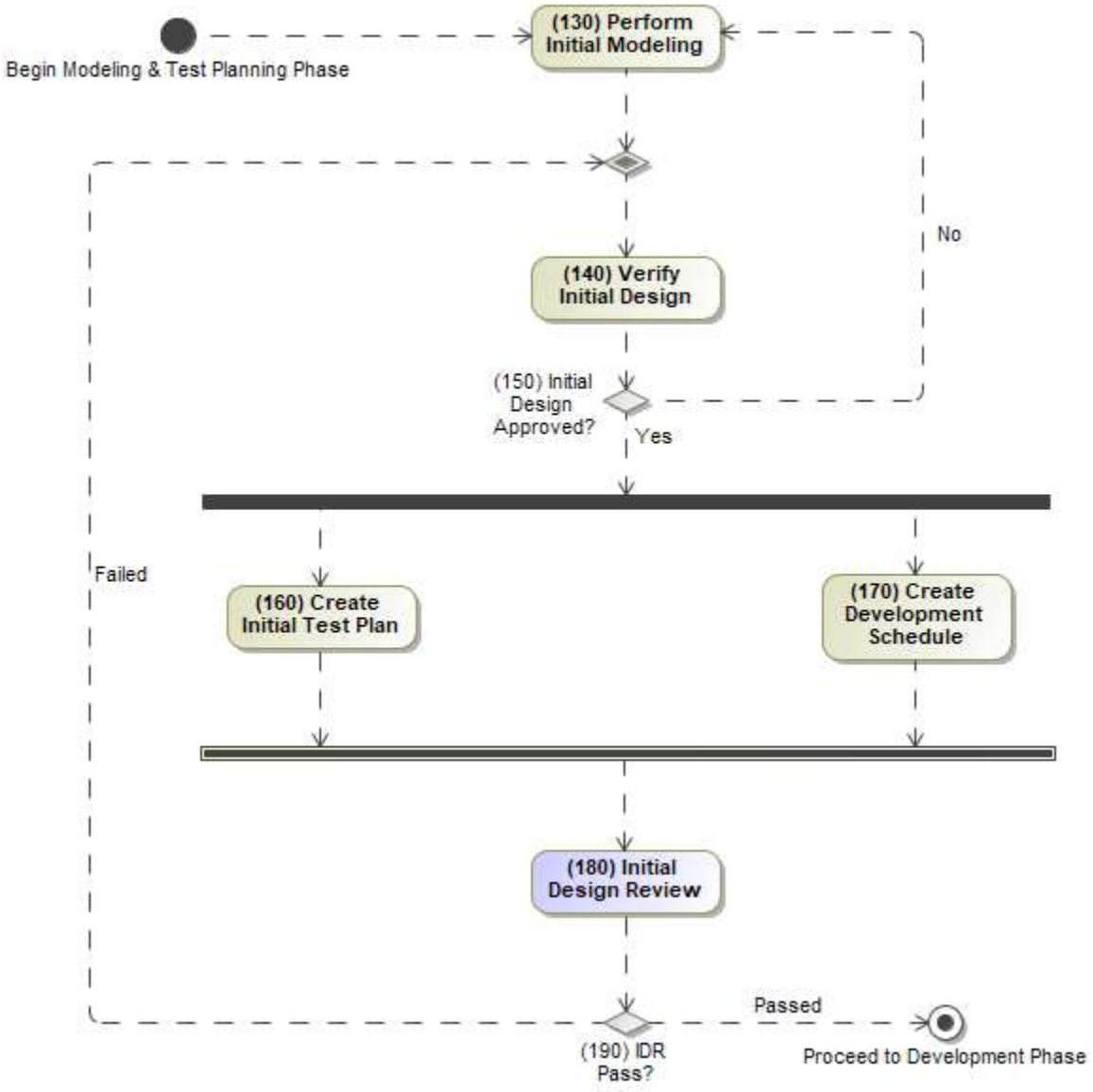

Figure 59: SEFAD Modeling & Test Planning Phase

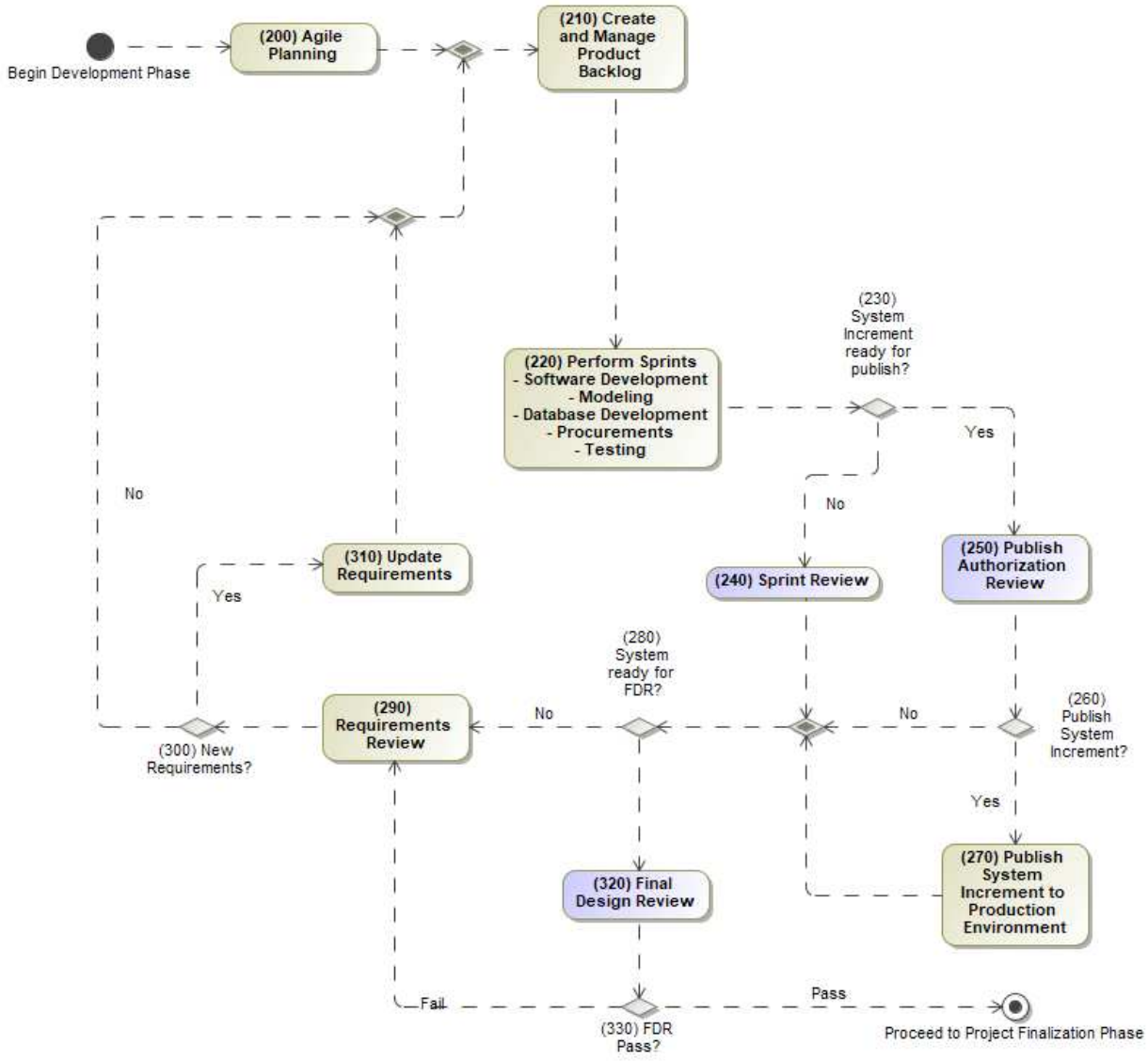

Figure 60: SEFAD Development Phase

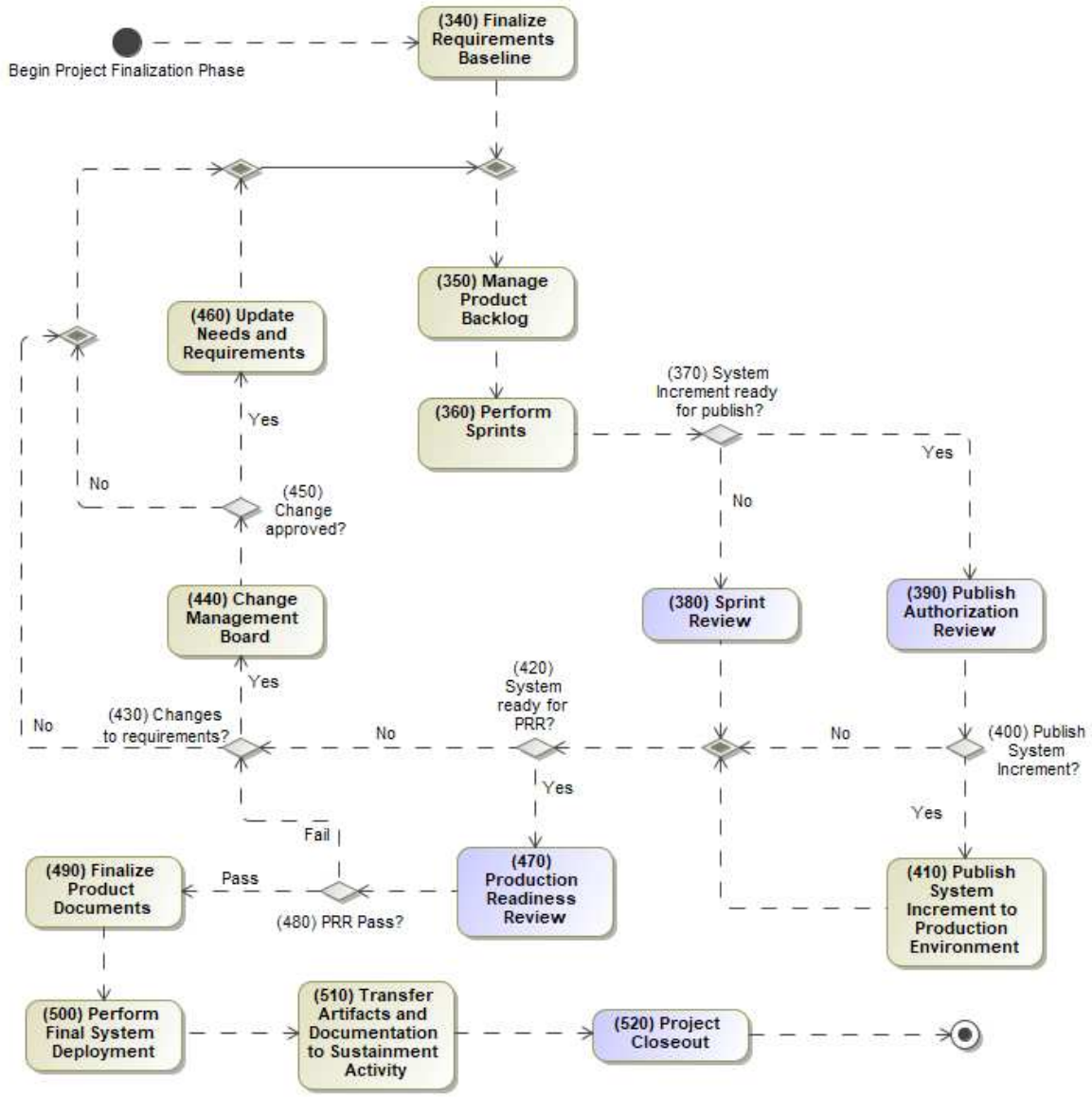

Figure 61: SEFAD Project Finalization Phase

8.1 Process Flow Details: Table 2 provides details on the process flow outlined in Figures 8 - 11.

- Individual Responsible: provides the Individual Responsible for accomplishing the task. In most case this will require bring together a Team of Subject Matter Experts (SMEs).

- Note: Stakeholders may assign a singular individual or a group of individuals to represent their needs in this process. This authority shall be documented in the project plan.

- SMEs Involved: provides a list of SMEs that may be brought together to accomplish the task. This is just a typical list of the Individuals Responsible and the Project Lead may obtain the services of anybody required to accomplish the task.
- Step: provides a pointer to the Process Flow Figure step.
- Action: provides a high-level summary of the action required.
- Artifact: provides a list of the typical outputs associated with the action performed.

Table 7: Process Flow Details

Individual(s) Responsible	SME(s) Involved	Step	Action	ARTIFACT
Product Owner Stakeholder	SE PO TPM	10	Establish project records. Determine configuration management policy for project.	▪ Initial Project Records ▪ Configuration management policy
Systems Engineer Product Owner	SE PO T&E Development Team	20	Develop stakeholder needs. A need is a high-level representation of a capability or objective that the stakeholder would like to accomplish. Needs are written as statements. Example: *Customers need mobile devices that are efficient to install, operate, and maintain, so that they can spend their time doing things which are more important to them.*	▪ Stakeholder Needs Registry

Individual(s) Responsible	SME(s) Involved	Step	Action	ARTIFACT
Systems Engineer Product Owner Stakeholder	SE PO T&E Development Team	30	Validate the customer needs as documented in the Stakeholder Needs Registry.	▪ Validated Stakeholder Needs Registry
Product Owner Stakeholder	SE PO	40	Are all stakeholder needs accounted for? Yes: Move to Stakeholder Needs Review. No: Return to developing stakeholder needs.	▪ Stakeholder needs completeness decision
Product Owner Stakeholder	SE PO TPM	50	Decision Point: SNR Question to answer: Are all stakeholder needs accounted for in the Stakeholder Needs Registry and is the system ready to develop system level requirements?	▪ Approval Decision for Stakeholder Needs Registry
Product Owner Stakeholder	SE PO TPM	60	Yes: Transition to developing system requirements. No: Return to developing stakeholder needs.	▪ SNR pass decision

Individual(s) Responsible	SME(s) Involved	Step	Action	ARTIFACT
Systems Engineer Product Owner	SE PO T&E Development Team	70	Develop system requirements. A requirement is: 1. A condition or capability needed by a user to solve a problem or achieve an objective. 2. A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed documents. 3. A documented representation of a condition or capability as in (1) or (2). Requirements are written as shall statements. Example: *The system shall require users to enter a pin number prior to the access of system functionality.*	▪ System Requirements Document
Systems Engineer Product Owner	SE PO T&E Development Team	80	Identify Technical Performance Measures, Design-Dependent Parameters, and Design Independent Parameters.	▪ Updated System Requirements Document with TPMs, DDPs, and DIPs

Individual(s) Responsible	SME(s) Involved	Step	Action	ARTIFACT
Systems Engineer Product Owner Stakeholder	SE PO T&E Development Team	90	Validate the system requirements as documented in the System Requirements Document.	▪ Validated Systems Requirements Document
Product Owner Stakeholder	SE PO	100	Are all system requirements accounted for? Yes/Pass: Move to System Requirements Review. No/Fail: Return to developing system requirements.	▪ System requirements completeness decision
Product Owner Stakeholder	SE PO TPM	110	Decision Point: SRR Are all system requirements accounted for in the System Requirements Document and is the system ready to move into modeling and test planning?	▪ Approval Decision for the System Requirements Document and Initial Baseline
Product Owner Stakeholder	SE PO TPM	120	Yes/Pass: Transition to the Modeling & Test Planning Phase. No/Fail: Return to developing system requirements.	▪ SRR Pass Decision
Product Owner Systems Engineer	SE PO Development Team T&E	130	Perform initial modeling. • Wire diagrams • SysML • DoDAF	▪ Initial system models

Individual(s) Responsible	SME(s) Involved	Step	Action	ARTIFACT
Product Owner Stakeholder	SE PO TPM	140	Verify the Initial Design: Question to Answer: Does the proposed system, as shown in through initial modeling, meets customer needs and system requirements.	▪ Verification of system models and initial design documents.
Product Owner Stakeholder	SE PO TPM	150	Yes: Proceed to the next steps. No: Return to Perform Initial Modeling, taking into account stakeholder feedback.	▪ Initial Model approval decision
Product Owner Systems Engineer Test Engineer	SE T&E	160	Create an initial test plan: • Create Test Plan • Create Test Matrix • Create Initial Test Cases, if applicable	▪ System Test Plan ▪ System Test Matrix ▪ Test Cases
Product Owner Technical Project Manager	PO TPM	170	Create the development schedule. • Create software development plan	▪ Finalized Project Plan ▪ Software Development Schedule

Individual(s) Responsible	SME(s) Involved	Step	Action	ARTIFACT
Product Owner Stakeholder	SE PO TPM	180	Decision Point: IDR Question to Answer: Will the proposed system meet stakeholder needs?	▪ Approved Project Plan ▪ Approved Initial Test Plan and associated artifacts. ▪ Approved Initial Development Schedule ▪ Approved Initial System Models
Product Owner Stakeholder	SE PO TPM	190	Yes/Pass: Transition to the Development Phase. No/Fail: Perform design verification.	▪ IDR Pass Decision
Product Owner Scrum Master Systems Engineer	PO SM SE Development Team Logistics	200	Agile Planning. • Finalize Sprint schedule, including Sprint time boxes. • Assign Scrum Master • Standup Scrum issue tracking system • Create Procurement Plan, if necessary	▪ Agile Scrum Schedule ▪ Agile Team Roster ▪ Software / Hardware Procurement Plan

Individual(s) Responsible	SME(s) Involved	Step	Action	ARTIFACT
Product Owner	PO SM SE Development Team T&E	210	Create and Manage the Product Backlog • Break down requirements into workable issues. • Assign point values to the work items. • Prioritize the Product Backlog • Perform Sprint Planning	▪ Initial Product Backlog ▪ Populated Sprints
Product Owner	PO SM SE Development Team T&E	220	Perform Sprints • Software Development • Modeling • Database Development • Procurements • Testing	▪ Demonstration of current product development through a Sprint Review
Product Owner	PO SE T&E Development Team	230	Does the team feel an increment of the system is ready for release and delivery to the stakeholder? Yes: Hold a Publish Authorization Review No: Hold a Sprint Review	▪ Publishable Increment decision

Individual(s) Responsible	SME(s) Involved	Step	Action	ARTIFACT
Product Owner Stakeholder	PO SE T&E Development Team	240	Sprint Review • Demonstration of Sprint work accomplished. • Determine if the system is ready for FDR	▪ Sprint Closeout Presentation ▪ Record of Accepted Sprint Items ▪ System ready for FDR determination
Product Owner Stakeholder	PO SE T&E Development Team	250	Publish Authorization Review • Demonstration of Sprint work accomplished. • Determination if the System Increment is ready for release. • Stakeholders sign off on Publish Authorization form, if publish is authorized	▪ Sprint Closeout Presentation ▪ Record of Accepted Sprint Items ▪ Publish System Increment decision. ▪ Publish Authorization Form, potentially. ▪ Published System Increment, potentially. ▪ System ready for FDR determination
Product Owner Stakeholder	PO SE	260	Permission to publish the system increment? Yes: Proceed to publish the system increment. No: Proceed to System ready for FDR determination.	▪ System Increment publish decision

Individual(s) Responsible	SME(s) Involved	Step	Action	ARTIFACT
Product Owner	PO SE T&E Development Team	270	Publish the approved System Increment to the production environment. • Inform stakeholders of release publication date and potential downtime. • Ensure training is updated. • Snap a baseline for the published system	▪ Published System Increment ▪ Published System Baseline
Product Owner Stakeholder	SE PO	280	Is the system ready for FDR? Yes: Move to Final Design Review. No: Proceed to Requirements Review.	▪ System ready for FDR decision
Product Owner Stakeholder	SE Development Team T&E	290	Perform a Requirements Review • Are the current requirements sufficient to represent the stakeholder's needs? • Does the stakeholder have new requirements? • Are there any requirements that are OBE?	▪ Requirements to be Added. ▪ Requirements to be Modified. ▪ Requirements to be Removed. ▪ Needs to be Added. ▪ Needs to be Modified. ▪ Needs to be Removed.

Individual(s) Responsible	SME(s) Involved	Step	Action	ARTIFACT
Product Owner	SE PO	300	Are there changes to requirements or stakeholder needs? Yes: Update the System Requirements Documents and/or the Stakeholder Needs Register. No: Proceed to Create and Manage the Product Backlog.	▪ Requirements Change Decision
Product Owner	SE PO	310	Update System Requirements and/or Stakeholder Needs • Update the System Requirements to add, remove, or modify requirements as applicable. • Update the Stakeholder Needs to add, remove, or modify needs as applicable	▪ Updated System Requirements Document ▪ Updated Stakeholder Needs Register
Product Owner Stakeholder	SE PO TPM	320	Decision Point: FDR Is the system sufficiently complete and documented to move into Project Finalization? NOTE: *This will result in a Stakeholder Needs and System Requirements freeze. No new work items, other than defects or emergent changes, shall be added to the Product Backlog for work.*	▪ Approved Project Plan ▪ Approved Test Plan and associated artifacts ▪ Approved Development Schedule ▪ Approved System Models

Individual(s) Responsible	SME(s) Involved	Step	Action	ARTIFACT
Product Owner Stakeholder	SE PO TPM	330	FDR Passed? Yes/Pass: Transition to the Project Finalization Phase. No/Fail: Perform Requirements Review.	▪ FDR Pass decision
Product Owner Systems Engineer	SE PO	340	Finalize the Stakeholder Needs and System Requirements. • Baseline the Stakeholder Needs Register • Baseline the System Requirements Document	▪ Baselined Stakeholder Needs Register ▪ Baselined System Requirements Document
Product Owner	PO SM SE Development Team T&E	350	Manage the Product Backlog • Break down requirements into workable issues. • Assign point values to the work items. • Prioritize the Product Backlog • Perform Sprint Planning	▪ Updated Product Backlog ▪ Populated Sprints
Product Owner	PO SM SE Development Team T&E	360	Perform Sprints • Software Development • Modeling • Database Development • Procurement • Testing	▪ Demonstration of current product development through a Sprint Review

Individual(s) Responsible	SME(s) Involved	Step	Action	ARTIFACT
Product Owner	PO SE T&E Development Team	370	Does the team feel an increment of the system is ready for release and delivery to the stakeholder? • Yes: Hold a Publish Authorization Review • No: Hold a Sprint Review	▪ Publishable Increment decision
Product Owner Stakeholder	PO SE T&E Development Team	380	Sprint Review • Demonstration of Sprint work accomplished. • Determine if the system is ready for FDR	▪ Sprint Closeout Presentation ▪ Record of Accepted Sprint Items ▪ System ready for FDR determination
Product Owner Stakeholder	PO SE T&E Development Team	390	Publish Authorization Review • Demonstration of Sprint work accomplished. • Determination if the System Increment is ready for release. • Stakeholders sign off on Publish Authorization form, if publish is authorized	▪ Sprint Closeout Presentation ▪ Record of Accepted Sprint Items ▪ Publish System Increment decision. ▪ Publish Authorization Form, potentially. ▪ Published System Increment, potentially. ▪ System ready for FDR determination

Individual(s) Responsible	SME(s) Involved	Step	Action	ARTIFACT
Product Owner Stakeholder	PO SE	400	Permission to publish the system increment? Yes: Proceed to publish the system increment. No: Proceed to System ready for PRR determination.	▪ System Increment publish decision
Product Owner	PO SE T&E Development Team	410	Publish the approved System Increment to the production environment. • Inform stakeholders of release publication date and potential downtime. • Ensure training is updated. • Snap a baseline for the published system	▪ Published System Increment ▪ Published System Baseline
Product Owner Stakeholder	SE PO	420	Is the system ready for PRR? • Yes: Move to Production Readiness Review. • No: Proceed to Requirements Review.	▪ System ready for PRR decision
Product Owner Stakeholder	SE PO	430	Is there a change to Stakeholder Needs or System Requirements? • Yes: Move to Change Management Board. • No: Proceed to Manage Product Backlog.	▪ Change to Stakeholder Needs or System Requirements decision

Individual(s) Responsible	SME(s) Involved	Step	Action	ARTIFACT
Product Owner Stakeholder	SE PO	440	Hold a Change Management Board to review proposed changes to the System Model, Stakeholder Needs, or System Requirements.	▪ Approved or Denied Change Requests
Product Owner Stakeholder	SE PO	450	Are there changes to the Stakeholder Needs, System Requirements, or System Model? • Yes: Update Stakeholder Needs, System Requirements, and the System Model, as necessary. • No: Proceed to Manage Product Backlog.	▪ Change approval decision
Product Owner Stakeholder	SE PO	460	Update the Stakeholder Needs, System Requirements, and System Models as appropriate	▪ Updated Stakeholder Needs Registry ▪ Updated System Requirements Document ▪ Updated System Models

Individual(s) Responsible	SME(s) Involved	Step	Action	ARTIFACT
Product Owner Stakeholder	SE PO TPM	470	Decision Point: PRR At the PRR, the Stakeholder and Product Owner decide whether the system is ready for final product delivery. Question to answer: Is the system sufficiently complete that stakeholder needs are met satisfactorily enough to finalize the project and move to sustainment? NOTE: *This does not mean that all requirements have been developed or that the product backlog is empty. This means that the stakeholder is satisfied with the current state of the system as a working viable product. Remaining requirements and work items will be triaged and packaged to be transferred for sustainment efforts.*	▪ Transferrable Product Backlog
Product Owner Stakeholder	SE PO TPM	480	PRR Passed? Yes/Pass: Transition to Finalize Product Documents. No/Fail: Proceed to Changes to Requirements?	▪ PRR Pass decision

Individual(s) Responsible	SME(s) Involved	Step	Action	ARTIFACT
Product Owner	SE PO TPM	490	Finalize project and product documents for transfer to sustainment activity and project archival. • Finalize Stakeholder Needs Registry • Finalize System Requirements Document • Finalize System Models • Project artifacts ready for closeout • Create Gold Discs, if applicable • Prepare and finalize system training documents for delivery. • Transfer any remaining work items in the Product Backlog into Change Requests or Task Items with Stakeholder input	▪ Transferable Product Backlog ▪ Outstanding Requirements as documented in the System Requirements Document ▪ Outstanding Stakeholder Needs as documented in the Stakeholder Needs Registry ▪ Project Artifacts ready for closeout ▪ Gold Discs for delivery ▪ Finalized system training documents
Product Owner	SE PO	500	Perform the final system deployment. • Deploy software to production servers. • Deliver Gold Discs to customers. • Perform final system training	▪ Deployed system and associated artifacts

Individual(s) Responsible	SME(s) Involved	Step	Action	ARTIFACT
Product Owner	PO SE	510	Transition artifacts and documentation to sustainment activity.	▪ Transferred artifacts. ▪ Transferred documents.
Product Owner	PO TPM	520	Perform project closeout activities. • Closeout Project Plan • Finalize financial documents. • Archive project documents	▪ Final Project Plan ▪ Final Financial Documents

Table 8: Performance Assurance and Audit Checklist

Note: This table is intended to be used by internal quality auditors, managers, and process practitioners to monitor and demonstrate adequate performance of this process.

Item	Indication of process compliance	Artifacts of process compliance			
		Record	Location	Custodian	Retention
1	Stakeholder Needs Register	Stakeholder Needs Register, Needs Baselines	See Configuration Management Plan for location/responsibility/ retention policies		
2	System Requirements Document	System Requirements Document, Requirements Baselines			
3	Configuration Management Plan	CM Plan			

Note: This table is intended to be used by internal quality auditors, managers, and process practitioners to monitor and demonstrate adequate performance of this process.

Item	Indication of process compliance	Artifacts of process compliance			
		Record	Location	Custodian	Retention
4	Project Plan	Project Plan Documents			
5	Stakeholder Needs Review (SNR)	Entrance/Exit Criteria, Approval Authority Signoff			
6	System Requirements Review (SRR)	Entrance/Exit Criteria, Approval Authority Signoff			
7	System Models	Documented Models			
8	Test Plan (Unit, Integration, Verification)	Documented Test Plan			
9	Test Matrix	Documented Test Matrix			
10	Test Procedures (Unit, Integration, Verification)	Documented Test Procedures			
1	Initial Design Review (IDR)	Entrance/Exit Criteria, Approval Authority Signoff			
2	Project Schedule (FY, Sprints, Prototypes, Delivery)	Project Schedule			
3	Product Backlog	Work Item Tracking System (JIRA, GitLab, etc.)			

Note: This table is intended to be used by internal quality auditors, managers, and process practitioners to monitor and demonstrate adequate performance of this process.

Item	Indication of process compliance	Artifacts of process compliance			
		Record	Location	Custodian	Retention
4	Sprint Logs	Work Item Tracking System (JIRA, GitLab, etc.)			
5	Sprint Test Results	Test Records			
6	Final Design Review (FDR)	Entrance/Exit Criteria, Approval Authority Signoff			
7	Project Plan[1]	Approved Project Plan			
8	Lessons Learned	Documented Lessons Learned			

SEFAD Implementation

The SEFAD process has been implemented in multiple DoD and government projects. To implement a process that utilizes the application of systems engineering principles, project teams require training to understand the importance and application of systems engineering methods. All projects had various levels of implementation of Agile or Predictive project management, as well as varying levels of requirements management. The application of the SEFAD process is

ongoing but shows exciting potential for increased systems engineering rigor and benefits to cost, schedule, and performance metrics.

Specific DoD project implementation:

- The DLMS project has implemented the following:
    a. Documented and mature Agile Scrum project execution.
    b. Customer driven rigorous requirements management.
    c. Requirements baseline of existing system (“As built”).
    d. Wireframing and light SYSML usage.
    e. Customer driven change management.
    f. SEFAD process implementation.
- The DPAS project has implemented the following:
    a. Documented and growing Agile Scrum project execution.
    b. Customer driven rigorous requirements management.
    c. Positive progress towards requirements baseline of existing system (“As built”).
    d. Customer driven change management.
    e. SEFAD process implementation.
- The DWMS project has implemented the following:
    a. Documented and growing Agile Scrum project execution.
    b. Customer driven rigorous requirements management.
    c. Positive progress towards requirements baseline of existing system (“As built”).
    d. Customer driven change management.
    e. SEFAD process implementation.

f. Cross-department partnership in systems engineering focused on Agile development.

- The DoD Software Hosting System (DSHS) project has implemented the following:
    a. Customer driven rigorous requirements management.
    b. Positive progress towards requirements baseline of existing system ("As built").
    c. SEFAD process implementation.
- The DESS project has implemented the following:
    a. Documented and growing Agile Scrum project execution.
    b. Customer driven rigorous requirements management.
    c. Completion of a requirements baseline of existing system ("As built").
    d. SEFAD process implementation.

Office of the Undersecretary of Defense (OUSD) DMSMS Program Office implementation:

- The Enterprise Part Management System project has implemented the following:
    a. Documented and growing Agile Scrum project execution.
    b. Documented and growing Scaled Agile Framework execution.
    c. Customer driven rigorous requirements management.
    d. Completion of a requirement baseline prior to project execution.
    e. SEFAD process implementation.
    f. Rigorous and thorough use of Model Based Systems Engineering.
        i. Systems modeling using SySML.
        ii. Department of Defense Architectural Framework Views and Models.

Other government branches or agencies implementing this methodology:

- Puget Sound Naval Shipyard (PSNS) Intermediate Maintenance Facility (IMF), Code 300.1
- NUWC Division Keyport, Undersea Weapons Department, Engineering & Production Enablement
- NUWC Division Keyport, Fleet Readiness Department, Electrical Engineering Applied Technology Branch
- NUWC Division Newport, In Service Engineering Activity 1533
- Naval Surface Warfare Center (NSWC) Crane Division, Microelectronics Assurance Branch (GXVR)
- Bureau of Labor Statistics, Office of Technology and Survey Processing
- Defense Human Resources Activity, Defense Manpower Data Center, Technical Services Division

SEFAD - Conclusions

How can Agile Scrum project implement Systems Engineering methodologies without losing the ability to take direct customer feedback and requirements changes to meet a customer driven definition of done? The ability for Agile Scrum practitioners to swiftly and decisively pivot to changes in customer needs, goals, and requirements is key to the success of these types of projects. Predictive style projects are far too rigid for software development and lose this ability to be flexible with desired customer changes. The use of User Stories in Agile Scrum projects allows for rapid refocus of software development projects but comes with the drawback of documenting requirements in a very abstract manner, much more akin to goals than requirements. The use of thorough, standardized, and atomic requirements in the form of systems

engineering shall statements increases the ability for developers and customers to communicate with one another, directly leading to improved system and problem domain understanding. Furthermore, clear requirements definition and management increases the ability for the development team to perform Verification and Validation activities. Additionally, performing modeling activities allows developers to better communicate project vision to customers, receiving rapid feedback, lowering the probability of misunderstanding of the system and problem domain, and decreasing the risk of defect generation and schedule slippage. Therefore, a process that thoughtfully implements systems engineering rigor, in the form of requirements management, modeling, and V&V, while retaining the flexibility of Agile Scrum incremental development and close customer collaboration, is needed.

The SEFAD process uses a "rigid – flexible – rigid" method of developing software. When considering the Systems Engineering V, the upper left-hand portion of the V represents project initiation. In Predictive style projects, this is where requirements are generated and validated with customers, along with initial project management activities. In phase 1 of the SEFAD process, this is still true. Customers and developers work closely to create a needs, goals, and requirements baseline to establish initial system and problem domain understanding. Modeling and V&V activities also occur on the left-hand side of the Systems Engineering V, which is represented in phase 2 of the SEFAD process. Both of these activities are considered "rigid" or standardized, fully documented, and following a prescribed path. When transitioning into development activities, the SEFAD process differs from the Systems Engineering V execution in that everything becomes "flexible," or Agile, in that all project artifacts can be changed, updated, added to, and deleted, including requirements. This allows customers to refocus on a project without having to restart the process. The differentiation of the SEFAD

approach to pure Agile Scrum is an addition of scheduled requirements meetings with customers. Needs, goals, and requirements are traced directly to User Stories, and requirements repository for the project is maintained. Post Sprint Review, customers are immediately asked to communicate any desired additions or deletions in requirements, as well as reprioritization of the requirements repository and Product Backlog. If new requirements are generated, they are traced to existing User Stories or new User Stories are generated and placed in the Product Backlog. In this way, the flexibility of Agile Scrum is maintained while also leveraging the clarity of systems engineering requirements management. When the customer is satisfied with the project and has identified an end state, the team then transitions into phase 4, baselining all system and project artifacts and minimizing changes. This represents a transition back into a rigid methodology, increasing the ability of the team to deliver on schedule. Through the use of a "rigid-flexible-rigid" method to develop software, increases in system understanding and documentation are gained while lowering the risk of negative impacts to cost schedule and performance.

CHAPTER 6 – RESEARCH CONTRIBUTIONS

Through completion of the preceding research activities, the following research contributions were realized:

1. This research has developed, validated, and tested a novel Agile Scrum Systems Dynamics Simulation Model. No system dynamics formulations and models of Agile Scrum methods that focus on the direct impact of work item quality on project success are present in the literature.
    a. The model replicates conditions of failure common in DoD software projects showing improvement in software execution integrating systems engineering methodologies with the Agile Scrum Framework.
    b. The model displays how the interrelationship of project requirements and changes drives project success and failure.
    c. The model can be used to perform scenario analysis to assist Agile Scrum project execution as shown through the system dynamics model.
    d. The model validates the benefits of implementing systems engineering rigor within Agile Scrum software executions.
2. This research has designed, created, and tested the costs and benefits of a new systems engineering program management process, the Systems Engineering Focused Agile Development (SEFAD), implementing Systems Engineering Methodologies, such as requirements management and MBSE, within the Agile Scrum Framework.
    a. The SEFAD methodology is defined and presented procedurally in the form of a Department of Defense software development policy, presenting step by step

instructions for the implementation and use of the process in DoD software development efforts, including directions for use with DoD Acquisition projects.

   b. Documentation of the results of surveys using SEFAD compared to previous only Agile Scrum Framework illustrate that the SEFAD methodology is clearly defined, effective, and usable by engineers.

   c. Documentation of the supplementary results communicate recommendations regarding implementation of SEFAD process.

3. Increased understanding of the similarities and differences in requirements management between Predictive projects and Agile Scrum projects.

   a. A paper communicating an in-depth comparison of systems engineering style requirements versus Connextra User Stories, defining the advantages and disadvantages of each.

   b. An analysis of Connextra User Stories as goals versus requirements, defining goals and directly tracing the definition of goals to the abstract nature of User Stories.

   c. An in-depth comparison of traceability present in projects when utilizing systems engineering style requirements versus Connextra User Stories, documenting and visualizing the traceability of each and communicating the advantages and disadvantages to their use.

4. This research has presented and analyzed a novel set of case study data that allowed for qualitative assessment of the costs and benefits of agile project execution of DoD software projects.

5. This research also has realized a set of applied and translational research contributions in that it has increased knowledge in the DoD and the broader software industry of the benefits of implementing systems engineering rigor in Agile software projects.
    a. These practical results were realized through the presentation of these research findings to the following DoD professional groups: U.S. Navy Digital Champions, the Naval Sea Systems Command One Digital Engineering Team, the Enterprise Parts Management DoD Working Group, the Naval Engineering Education Consortium, and the Defense Acquisition University.
    b. Presentation at the National Defense Industrial Association - 27th Annual Systems and Mission Engineering Conference on October 28 - 31, 2024, executive level DoD leadership will have the opportunity to see the benefits of the merger of systems engineering methods with the Agile Scrum framework.
    c. Portions of this research have been published, or scheduled to be published, in multiple professional journals. An article entitled “Engineering Rigor in the Obsolescence Management Information System (OMIS) Project” has been submitted to the Journal of DoD Research and Engineering and is currently in peer review. An article entitled “How Requirements Management Can Improve Agile Scrum Software Projects” is in late drafts and being prepared for submission to Crosstalk, The Journal of Defense Software Engineering. An article entitled “Simulating the Effects of Applying Systems Engineering Rigor in Agile Scrum Software Projects” is in preparatory drafts with a planned submission to Systems Engineering in December 2024.

CHAPTER 7 – CONCLUSIONS

Summary Introduction

This dissertation studied how the application, or lack thereof, of systems engineering methods used in conjunction with the Agile Scrum Framework affects the success or failure of DoD software projects. The dissertation focused specifically on DoD software projects, as these types of projects are typically more risk averse, with many of the projects having life or death stakes if defects are introduced or projects are delayed or not completed to stakeholder satisfaction. A review of the interrelationship between product requirements and changes that drive the success and failure of Agile projects was performed. Case studies were generated to determine the causes of success and failure in DoD software projects, as well as noting the effect of the application of systems engineering methodologies had on said projects. A model of Agile Scrum project execution was created and used to simulate the effects of adding systems engineering methods to DoD software projects. Finally, a new methodology that combines systems engineering methods with the Agile Scrum Framework was created and implemented in multiple ongoing DoD projects.

Chapter Summaries

Chapter 1 focused on the need for the application of system engineering methods to Agile Scrum software executions for DoD software projects. The software development process in industry versus the DoD was explored with an explanation of the need reasoning for the need of higher engineering rigor in DoD software projects. A brief discussion of the utilization of User Stories versus systems engineering shall statements was presented, including the benefits and drawbacks of each. The evolution of the Adaptive Acquisition Framework was discussed as well

as common misunderstandings of the AAF held by many in the DoD. The chapter concludes with an argument for the need to merge systems engineering methods with the Agile Scrum Framework for DoD software projects.

Chapter 2 discusses the interrelationships between project requirements and changes that drive the success when utilizing Agile Scrum User Stories versus systems engineering shall statements. An explanation of the Agile Scrum Framework and software development methodology was presented, discussing the benefits and drawbacks of the framework. Predictive project management was briefly presented, and a discussion of the benefits and drawbacks of the framework was discussed. The similarities and differences in need for requirements management in both Agile Scrum and Predictive projects were detailed, specifically noting the need for flexibility in requirements management with regards to Agile projects. A discussion on requirements traceability was presented with the differences between INCOSE requirements management and Agile User Stories traceability defined. The argument of User Stories being classified as goals rather than requirements was laid out, laying forth the foundation that User Stories cannot accurately represent stakeholder needs and requirements, as they are goals. The chapter concludes with the argument that systems engineering shall statements and MBSE methods should be used to perform INCOSE requirements management in DoD software projects and trace those requirements directly to User Stories, allowing for the flexibility in managing backlogs while increasing engineering and technical rigor and allowing for better and thorough testing through understanding of system requirements coverage.

Chapter 3 presents the relationships, and systematic causes, of the successes and failures of DoD software projects. Case studies were conducted detailing the benefits and drawbacks of the addition of systems engineering methods to Agile Scrum software executions, as well as

reviewing software projects that had no systems engineering methods or Agile Scrum execution. Successes in applying requirements management and thorough test planning in the DESS system were reported. The evolution of the DLMS project execution was presented, including the adoption of Agile Scrum from no project framework and the addition of requirements management and modeling to increase customer communication. A history of the DPAS system was discussed as well as the struggles to deploy an updated version due to a lack of engineering rigor and too much Agility. The negative effects of too much engineering and technical rigor were elaborated on for the DQMS system. The DWMS project was reviewed, noting partial compliance with regards to Agile Scrum and how implementing systems engineering methods could improve project success metrics. Concluding remarks detail the special case of DoD software, where a software system must be sustained for decades and how, due to the nature of growth in DoD software systems, many do not have good documentation and rely instead on tribal knowledge. This resulted in the conclusion of a need for the addition of systems engineering methods in conjunction with the Agile Scrum Framework for DoD software project accomplishment.

Chapter 4 details the development and results of the Agile Scrum Systems Dynamics Simulation Model, a systems dynamics model that simulates a DoD software execution using the Agile Scrum Framework. This framework was developed based upon real world software execution resulting from historical data gathered regarding DoD software projects. Reference mode data was generated and used to validate the model. Additional reference mode data was gathered to verify the model adequately simulated Agile Scrum projects that both had no systems engineering rigor and those that had implemented systems engineering methods. The model

successfully simulates and communicates the benefits of the addition of systems engineering methods in the Agile Scrum Framework.

Chapter 5 presents the Systems Engineering Focused Agile Development methodology which is a framework that combines systems engineering methods with an Agile Scrum software execution. Step by step instructions of the SEFAD are given as well as diagrams showing implementation. The process takes a rigorous-flexible-rigorous approach to software development, requiring requirements, modeling, and V&V baselining, transitioning into Agile Scrum Sprint cycles, and then once again baseline project artifacts for project closeout. Close customer collaboration is required as well as frequent updates to needs, goals, and requirements by the stakeholder team. This framework has been adopted in multiple projects which are cited at the end of the chapter.

In chapter 6, the contributions to the systems engineering and software engineering research communities are detailed. In line with research objectives, contributions include the successful application of systems engineering methodologies to Agile Scrum project executions in six NUWC Division Keyport software projects, as well as the adoption of the SEFAD model by seven other government projects, the development and application of the Agile Scrum Systems Dynamics Simulation Model, five case studies detailing the benefits and drawbacks of implementing systems engineering rigor and Agile Scrum in DoD software projects, and documentation and further understanding of the intersection and divergence between Agile User Stories and systems engineering requirements management. Publication submissions resulting from this research include journal articles, conference presentations, training presentations, DoD working group presentations, and DoD policy.

Future Work

This work presents the foundations of implementing systems engineering rigor thoughtfully in DoD Agile Scrum software projects. Research should continue in this field to determine the benefits of specific systems engineering methods, such as the addition of traced shall statements to User Stories, when tracked over multi-year projects. The initial next step should therefore be the documentation of the benefits and drawbacks of the presented methodology as pertains to the DoD systems that have enacted it as more project data is available. Further use of reference modes with the Agile Scrum Systems Dynamics Simulation Model will result in increased understanding of the impacts of work item quality on software project execution. Rigorous experimentation using the model to validate other driving factors of work item quality which represents the core dynamics of the Agile Scrum Framework can better focus areas for improvements to software development efforts. In future years, gathering this data will allow for a more mature systems dynamics model to be developed, increasing elements that have a dynamic effect on software execution.

The Agile Scrum Systems Dynamics Simulation Model generated for this research was an initial first step and represents a simplified Agile Scrum execution that does not account for other influential factors. The model can be expanded upon to include other driving factors of Agile project success outside of the implementation of systems engineering methods. These factors could include resource instability, changes in scope, and availability of adequately trained and qualified personnel, the addition or removal of personnel from a project, multi-year velocity tracking, and the experience levels of individuals and teams with the process. Continuance of this research will present a further understanding of the dynamics inherent in DoD software development.

Similar to how the Agile Scrum Systems Dynamics Model presents a simulation of an individual Scrum Team, a model can be created to document and simulate the effects of systems engineering rigor on an enterprise development effort, such as Scrum of Scrums or the Scaled Agile Framework (SAFe). Enterprise Agile efforts track and manage multiple Scrum Teams and apply the Agile Framework to business processes and organizational leadership. Driving factors such as schedule slippage in one team can directly affect another team, such as when required cybersecurity certifications of systems are not delivered on time, preventing scheduled versions of software from being published, pushing back the delivery of a hardware system with embedded software to a customer. No research into the merger of systems engineering methods to this type of project execution has been performed and would be of significant use to both the DoD and larger, enterprise level software executions in industry.

Finally, a software simulation solution can be coded, allowing for the automated import of project data and more in-depth simulation and analysis of software project execution. This would allow researchers and project managers to visualize bottlenecks and identify key driving factors behind project successes and failures. By utilizing multiple imports of distinct software project executions over multiple years as reference modes, direct comparisons can be made. Additionally, by creating a software solution to perform these tasks, the limitations of using Vensim or Simulink can be overcome, allowing for better visualization of data through business intelligence style dashboards and machine learning and artificial intelligence activities to better predict positive and negative dynamics factors in software project executions.

BIBLIOGRAPHY

[1] R. Dove and B. Schindel, "Agile Systems Engineering Life Cycle Model for Mixed Discipline Engineering," in *29th Annual INCOSE Interantional Symposium*, Orlando, FL, 2019.

[2] M. A. Lapham, S. Miller, L. Adams, N. Brown, B. Hackemack, C. (. Hammons, L. Levine and A. Schenker, "Agile Methods: Selected DoD Management and Acquisition Concerns," Software Engineering Institute, Carnegie Mellon University, 2011.

[3] S. Das, Z. Ali, S.-N. Bandi, A. Bhagat, N. Chandrakumar, P. Kucheria, M. Pariente, A. Singh and B. Tipping, "Agile Systems Engineering in Building Complex AI Systems," in *Engineering Artificially Intelligent Systems*, Cham, Switzerland, Springer, 2021, pp. 192-208.

[4] R. Cooper and E. Kleinschmidt, "Benchmarking the Firm's Critical Success Factors in New Product Development," *Journal of Product Innovation Management,* vol. 12, no. 5, pp. 374-391, 1995.

[5] J. D. Sherman, "Optimal Modes and Levels of Integration, and the Identification of Cross-Functional Coordination Deficiencies in Concurrent Engineering," *IEEE TRANSACTIONS ON ENGINEERING MANAGEMENT,* vol. 51, no. 3, pp. 268-278, 2004.

[6] R. Dove, K. Lunney, M. Orosz and M. Yokell, "Systems Engineering Agility in a Nutshell," *INCOSE Insight,* vol. 26, no. 2, 2023.

[7] A. Begel and N. Nagappan, "Usage and Perceptions of Agile Software Development in an Industrial Context: An Exploratory Study," in *First International Symposium on Empirical Software Engineering and Measurement (ESEM 2007)*, Madrid, Spain, 2007.

[8] M. Fowler and J. Highsmith, "The agile manifesto," *Software Development,* vol. 9, pp. 28-35, 2001.

[9] K. Schwaber and J. Sutherland, The Scrum Guide, Ken Schwaber and Jeff Sutherland, 2020.

[10] A. Azanha, A. R. T. T. Argoud, J. B. d. Camargo Junior and P. D. Antoniolli, "Agile project management with Scrum," *International journal of managing projects in business,* vol. 10, no. 1, pp. 121-142, 2017.

[11] Y. Bijan, J. Yu, J. Stracener and T. Woods, "Systems Requirements Engineering—State of the Methodology," *Systems Engineering,* vol. 16, no. 3, pp. 251-377, 2013.

[12] Chief Information Officer, "DoDAF DoD Architectural Framework Version 2.02," United States Department of Defense, 08 2010. [Online]. Available: https://dodcio.defense.gov/library/dod-architecture-framework/. [Accessed 03 03 2024].

[13] O. Kautz, A. Roth and B. Rumpe, "Achievements, Failures, and the Future of Model-Based Software Engineering," in *The Essence of Software Engineering*, Cham, Switzerland, Springer Open, 2018, pp. 221-236.

[14] B. Boehm, J. A. Lane, S. Koolmanojwong and R. Turner, "Architected Agile Solutions for Software Reliant Systems," in *Agile Software Development*, Berlin, Heidelberg, Springer, 2007, pp. 165-184.

[15] J. S. Dahmann and K. J. Baldwin, "Understanding the Current State of US Defense Systems of Systems and the Implications for Systems Engineering," in *SysCon 2008 - IEEE International Systems Conference*, Montreal, Canada, 2008.

[16] DoD Deputy Chief Information Officer, "DoD Architecture Framework Version 2.02," U.S. Department of Defense, [Online]. Available: https://dodcio.defense.gov/library/dod-architecture-framework/. [Accessed 17 10 2022].

[17] X. Wang, L. Zhao, Y. Wang and J. Sun, "The Role of Requirements Engineering Practices in Agile Development:: An Empirical Study," in *Requirements Engineering. Communications in Computer and Information Science.*, vol. 432, Berlin Heidelberg, Springer-Verlag, 2014, pp. 195-209.

[18] G. Lucassen, F. Dalpiaz, J. M. E.M. van der Werf and S. Brinkkemper, "The Use and Effectiveness of User Stories," in *Requirements Engineering: Foundation for Software Quality*, Gothenburg, Sweden, 2016.

[19] J. E. Kasser, Systems Engineering: A Systemic and Systematic Methodology for Solving Complex Problems, Boca Raton, FL: CRC Press, Taylor & Francis Group, 2020.

[20] F. Paetsch, A. Eberlein and F. Maurer, "Requirements Engineering and Agile Software Development," in *Proceedings of the Twelfth IEEE International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises*, Linz, Austria, 2003.

[21] Y. Wautelet, S. Heng, M. Kolp, I. Mirbel and S. Poelmans, "Building a rationale diagram for evaluating user story sets," in *2016 IEEE Tenth International Conference on Research Challenges in Information Science (RCIS)*, 2016.

[22] P. Zave and M. Jackson, "Four Dark Corners of Requirements Engineering," *ACM Transactions on Software Engineering and Methodology,* vol. 6, no. 1, pp. 1-30, 1997.

[23] A. Moore, B. Nichols, B. Novak and D. Zubrow, "Model-Based Analysis of Agile Development Practices," Carnegie Mellon University, 17 07 2019. [Online]. Available: http://insights.sei.cmu.edu/blog/model-based-analysis-of-agile-development-practices/. [Accessed 3 12 2022].

[24] W. Behutiye, P. Karhapää, D. Costal, M. Oivo and X. Franch, "Non-functional Requirements Documentation in Agile Software Development: Challenges and Solution Proposal," in *Product-Focused Software Process Improvement: 18th International Conference, PROFES 2017*, Innsbruck, Austria, 2017.

[25] P. S. Kochhar, X. Xia and D. Lo, "Practitioners' Views on Good Software Testing Practices," in *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, Montreal, QC, Canada, 2019.

[26] J. Hutchinson, J. Whittle and M. Rouncefield, "Model-driven engineering practices in industry: Social, organizational and managerial factors that lead to success or failure," *Science of Computer Programming,* vol. 89, pp. 144-161, 2013.

[27] R. France and B. Rumpe, "Model-driven Development of Complex Software: A Research Roadmap," in *Future of Software Engineering (FOSE '07)*, Minneapolis, MN, USA, 2007.

[28] B. Selic, "The Pragmatics of Model-Driven Development," *IEEE Software,* vol. 20, no. 5, pp. 19-25, 2003.

[29] N. M. J. Basha, S. A. Moiz and M. Rizwanullah, "Model Based Software Development: Issues & Challenges," *International Journal of Computer Science and Informatics (IJCSI),* vol. 3, no. 2, pp. 226-230, 2020.

[30] Office of the Under Secretary of Defense for Acquisition and Sustainment, *Software Acquisition,* Washington, DC: Office of the Under Secretary of Defense for Acquisition and Sustainment, 2022.

[31] Office of the Under Secretary of Defense for Acquisition and Sustainment, *DOD INSTRUCTION 5000.02: OPERATION OF THE ADAPTIVE ACQUISITION FRAMEWORK,* Washington, DC, 2022.

[32] O. o. t. U. S. o. D. f. A. a. Sustainment, *DOD INSTRUCTION 5000.02: Operation of the Defense Acquisition System,* Washington, DC: Office of the Under Secretary of Defense for Acquisition and Sustainment, 2013.

[33] J. Riposo, M. McKernan and C. Kaihoi Duran, "Prolonged Cycle Times and Schedule Growth in Defense Acquisition: A Literature Review," RAND Corporation, WASHINGTON, DC, 2014.

[34] Defense Acquisition University, "Software Acquisition - Glossary," [Online]. Available: https://aaf.dau.edu/aaf/software/glossary/. [Accessed 28 09 2024].

[35] E. Wrubel and J. Gross, "Contracting for Agile Software Development in Department of Defense: An Introduction," Hanscom AFB, MA, 2015.

[36] Office of the Under Secretary of Defense for Acquisition and Sustainment, *DOD 5000.87: OPERATION OF THE SOFTWARE ACQUISITION PATHWAY,* Washington, DC: Office of the Under Secretary of Defense for Acquisition and Sustainment, 2020.

[37] J. Grenning, Interviewee, *Agile Uprising Podcast: Interview with James Grenning.* [Interview]. 01 JAN 2017.

[38] S. Ashmore and K. Runyan, Introduction to Agile Methods, Upper Saddle River, NJ: Addison-Wesley Professional, 2014.

[39] P. Hohl, J. Klunder, A. van Bennekum, R. Lockard, J. Gifford, J. Munch, M. Stupperich and K. Schneider, "Back to the future: origins and directions of the “Agile Manifesto” – views of the originators," *Journal of Software Engineering Research and Development,* vol. 6, no. 15, 2018.

[40] R. K. Wysocki, Effective Project Management: Traditional, Agile, Extreme, Sixth Edition, Indianapolis, IN: Wiley, 2012.

[41] Project Management Institute, Agile Practice Guide, Project Management Institute, 2017.

[42] Project Management Institute, The Standard for Project Management and A Guide to the Project Management Body of Knowledge, Newtown Square, PA: Project Management Institute, Inc., 2021.

[43] P. Hines, M. Holweg and N. Rich, "Learning to evolve: A review of contemporary lean thinking," *International Journal of Operations & Production Management,* vol. 24, no. 10, pp. 994-1011, 2004.

[44] M. C. Paulk, "A Scrum Adoption Survey," *Software Quality Professional,* vol. 15, no. 2, 2013.

[45] A. Rauf and M. AlGhafees, "Gap Analysis between State of Practice & State of Art Practices in Agile Software Development," in *Agile Conference (AGILE)*, National Harbor, MD, 2015.

[46] ScrumAlliance, "These are the Differences Between Agile and Scrum, and How They Differ From Waterfall," ScrumAlliance, [Online]. Available: https://resources.scrumalliance.org/Article/differences-agile-scrum-differ-waterfall. [Accessed 25 JAN 2024].

[47] TCGen, "Agile VS Scrum: Similarities and Differences," TCGen, [Online]. Available: https://www.tcgen.com/agile/agile-vs-scrum/. [Accessed 24 JAN 2024].

[48] G. Verheyen, "Agile and Scrum, entwined and related," Scrum.org, 16 MAR 2015. [Online]. Available: https://www.scrum.org/resources/blog/agile-and-scrum-entwined-and-related. [Accessed 25 JAN 2024].

[49] I. W. Salemme, "Agile Methodology vs Scrum," pipefy, 30 NOV 2023. [Online]. Available: https://www.pipefy.com/blog/scrum-vs-agile/. [Accessed 25 JAN 2024].

[50] T. H. Davenport, Thinking for a Living: How to Get Better Performances And Results from Knowledge Workers, Boston, MA: Harvard Business Review Press, 2005.

[51] C. Verwijs, "Thinking By Sprinting: What Cognitive Science Tells Us About Why Scrum Works," Scrum.org, 03 FEB 2020. [Online]. Available: https://www.scrum.org/resources/blog/thinking-sprinting-what-cognitive-science-tells-us-about-why-scrum-works. [Accessed 25 JAN 2024].

[52] C. Verwijs and D. Russo, "A Theory of Scrum Team Effectiveness," *ACM Transactions on Software Engineering and Methodology,* vol. 32, no. 3, p. 1–51, 2023.

[53] H. K. Flora and S. V. Chande, "A Systematic Study on Agile Software Development Methodologies and Practices," *International Journal of Computer Science and Information Technologies,* vol. 5, no. 3, pp. 3626-3637, 2014.

[54] Institute of Electrical and Electronic Engineers (IEEE), Systems and software engineering — Life cycle processes — Requirements engineering, New York, NY: Institute of Electrical and Electronic Engineers (IEEE), 2011.

[55] INCOSE Requirements Working Group, Guide to Writing Requirements, San Diego, CA: International Council on Systems Engineering (INCOSE), 2023.

[56] Mountain Goat Software, "Planning Poker," Mountain Goat Software, 14 04 2024. [Online]. Available: https://www.mountaingoatsoftware.com/agile/planning-poker.

[57] V. Mahnic and T. Hovelja, "On using planning poker for estimating user stories," *The Journal of Systems and Software,* vol. 85, no. 9, pp. 2086-2095, 2012.

[58] M. Jorgensen, "A review of studies on expert estimation of software development effort," *The Journal of Systems and Software,* vol. 70, no. 1-2, pp. 37-60, 2004.

[59] B. Ramesh, T. Powers and C. Stubbs, "Implementing Requirements Thaceability: A Case Study," in *Proceedings of 1995 IEEE International Symposium on Requirements Engineering (RE'95)*, York, UK, 1995.

[60] J. Dick, E. Hull and K. Jackson, Requirements Engineering, Cham, Switzerland: Springer International Publishing AG, 2017.

[61] C. Lee, G. Luigi and J. Xiaoping, "An agile approach to capturing requirements and traceability.," in *Proceedings of the 2nd international workshop on traceability in emerging forms of software engineering (TEFSE 2003)*, New York, NY, 2003.

[62] T. Dingsoyr, T. Dyba and N. B. Moe, Agile Software Development: Current Research and Future Directions, Berlin, Germany: Springer Berlin, Heidelberg, 2010.

[63] Scrum.org, "Is it worth using Requirement Tracability Matrix in scrum," Scrum.org, 23 02 2023. [Online]. Available: https://www.scrum.org/forum/scrum-forum/68586/it-worth-using-requirement-tracability-matrix-scrum. [Accessed 14 04 2024].

[64] M. Cohn, "User Stories Applied For Agile Software Development," in *XP Atlanta*, Atlanta, GA, 2004.

[65] M. Rehkopf, "Agile epics: definition, examples, and templates," Atlassian, [Online]. Available: https://www.atlassian.com/agile/project-management/epics. [Accessed 14 04 2024].

[66] Agile Alliance, "Agile Glossary," Agile Alliance, [Online]. Available: https://www.agilealliance.org/glossary. [Accessed 14 04 2024].

[67] M. Britsch, "The Basics: Epics, Stories, Themes & Features," The Digital Business Analyst, 05 09 2017. [Online]. Available: https://thedigitalbusinessanalyst.co.uk/epics-stories-themes-and-features-4637712cff5c. [Accessed 14 04 2024].

[68] C. Guay, "Scrum tips: Differences between epics, stories, themes and features," 02 09 2019. [Online]. Available: https://const.fr/blog/agile/scrum-differences-epics-stories-themes-features/. [Accessed 14 04 2024].

[69] M. Rehkopf, "User stories with examples and a template," Atlassian, [Online]. Available: https://www.atlassian.com/agile/project-management/user-stories#:~:text=A%20user%20story%20is%20the,the%20end%20user%20or%20customer.. [Accessed 14 04 2024].

[70] Agile Modeling, "User Stories: An Agile Introduction," Agile Modeling, [Online]. Available: https://agilemodeling.com/artifacts/userStory.htm. [Accessed 14 04 2024].

[71] M. Cohn, "Epics, Features and User Stories," Mountain Goat Sofware, 08 11 2022. [Online]. Available: https://www.mountaingoatsoftware.com/blog/stories-epics-and-themes. [Accessed 14 04 2024].

[72] B. Maloney, "What Are User Personas?," ScrumAlliance, [Online]. Available: https://resources.scrumalliance.org/Article/user-personas. [Accessed 14 04 2024].

[73] P. W. Szabo, User Experience Mapping, Packt Publishing, 2017.

[74] D. Broschinsky and L. Baker, "Using Persona with XP at LANDesk Software, an Avocent Company," in *Agile 2008 Conference*, Toronto, Canada, 2008.

[75] A. Cooper, Inmates Are Running the Asylum, The: Why High-Tech Products Drive Us Crazy and How to Restore the Sanity, United States of America: Sams Publishing, 2004.

[76] J. Cleland-Huang, G. Zemont and W. Lukasik, "A heterogeneous solution for improving the return on investment of requirements traceability," in *Proceedings. 12th IEEE International Requirements Engineering Conference*, Kyoto, Japan, 2004.

[77] J. Cleland-Huang, O. Gotel and A. Zisman, Software and Systems Traceability, London, UK: Springer London, 2012.

[78] S. Bose, "How to Report on Traceability and Test Coverage in Jira," TestRail, 30 01 2023. [Online]. Available: https://www.testrail.com/blog/jira-traceability-test-coverage. [Accessed 14 04 2024].

[79] International Council on Systems Engineering, Systems Engineering Handbook, 5th Edition, San Diego, CA: International Council on Systems Engineering (INCOSE), 2023.

[80] S. Friedenthal, A. Moore and R. Steiner, A Practical Guide to SysML: The Systems Modeling Language, Elsevier Inc., 2015.

[81] M. dos Santos Soares and J. Vrancken, "Model-Driven User Requirements Specification using SysML," *JOURNAL OF SOFTWARE,* vol. 3, no. 6, pp. 57-68, 2008.

[82] J. Wheaton and D. R. Herber, "Digital requirements engineering with an INCOSE-derived SysML meta-model," in *Conference on Systems Engineering Research (CSER) 2024*, Tucson, AZ, 2024.

[83] D. R. Herber and K. Eftekhari-Shahroud, "BUILDING A REQUIREMENTS DIGITAL THREAD FROM CONCEPT TO TESTING USING MODEL-BASED STRUCTURED REQUIREMENTS APPLIED TO THRUST REVERSER ACTUATION SYSTEM DEVELOPMENT," in *9th International Conference on Recent Advances in Aerospace Actuation Systems and Components*, Toulouse, France, 2023.

[84] No Magic, Inc., "MagicDraw 19.0 LTR," Dassault Systemes, [Online]. Available: https://docs.nomagic.com/display/MD190/Stereotype. [Accessed 21 04 2024].

[85] D. R. Herber, J. B. Narsinghani and K. Eftekhair-Shahroudi, "Model-Based Structured Requirements in SysML," in *IEEE 2022 International Systems Conference (SysCon)*, Virtual Conference, 2022.

[86] K. Pohl, Requirements Engineering: Fundamentals, Principles, and Techniques, Berlin, Germany: Springer Berlin, Heidelberg, 2010.

[87] T. Günes and F. B. Aydemir, "Automated Goal Model Extraction from User Stories Using NLP," in *2020 IEEE 28th International Requirements Engineering Conference (RE)*, Zurich, Switzerland, 2020.

[88] A. R. Amna and G. Poels, "Ambiguity in user stories: A systematic literature review," *Information and Software Technology,* vol. 145, 2022.

[89] B. Boehm and R. Turner, Balancing Agility and Discipline: A Guide for the Perplexed, Addison-Wesley/Pearson Education, 2004.

[90] V. Chauhan, "Smoke Testing," *International Journal of Scientific and Research Publications,* vol. 4, no. 2, 2014.

[91] Atlassian, "R4J - Requirements Management for Jira," 13 05 2023. [Online]. Available: https://marketplace.atlassian.com/apps/1213064/r4j-requirements-management-for-jira?tab=overview&hosting=cloud.

[92] Defense Acquisition University, "Test Readiness Review (TRR)," Defense Acquisition University, 13 05 2023. [Online]. Available:

https://aaf.dau.edu/aaf/mca/trr/#:~:text=A%20Test%20Readiness%20Review%20(TRR,and%20gathers%20the%20required%20information..

[93] Defense Acquisition University, "Production Readiness Review (PRR)," Defense Acquisition University, [Online]. Available: https://aaf.dau.edu/aaf/mca/prr/. [Accessed 12 05 2023].

[94] M. Wozniak, "Systems Engineering Isn't Scary for Agile Practitioners," in *33rd Annual Incose International Symposium*, Honolulu, HI, USA, 2023.

[95] Defense Standardization Program Office, "SD-22 - Diminishing Manufacturing Sources and Material Shortages: A Guidebook of Best Practices for Implementing a Robust DMSMS Management Program," May 2022. [Online]. Available: https://www.dau.edu/sites/default/files/Migrated/ToolAttachments/Diminishing-Manufacturing-Sources-and-Material-Shortages-%28DMSMS%29-Guidebook-%28SD-22%29.pdf. [Accessed January 2024].

[96] ISTQB, "Smoke Test," International Glossary of Terms Used in Software Testing, [Online]. Available: https://glossary.istqb.org/en_US/term/smoke-test-1. [Accessed 25 03 2024].

[97] CMMI Institute, "Capability Maturity Model Integration," Information Systems Audit and Control Association, [Online]. Available: https://cmmiinstitute.com/. [Accessed 23 January 2024].

[98] U.S. Government Accountability Office, "DoD Business Systems Modernization - Key Marine Corps System Acquisition Needs to be Better Justified, Defined, and Managed," U.S. Government Accountability Office, Washington,DC, 2008.

[99] S. Dimitrijevic, J. Jovanovic and V. Devedzic, "A comparative study of software tools for user story management," *Information and Software Technology,* vol. 57, pp. 352-368, 2015.

[100] R. Jeffries, A. Anderson and C. Hendrickson, Extreme Programming Installed, Upper Saddle River, NJ: Addison-Wesley Professional, 2000.

[101] A. Pranam, "Rapid Prototyping," in *Product Management Essentials*, Berkeley, CA, Apress, 2018, pp. 97-125.

[102] Object Management Group (OMG), "WHAT IS SYSML?," Object Management Group, [Online]. Available: https://www.omgsysml.org/what-is-sysml.htm. [Accessed 2024 03 03].

[103] S. Kumar, "A REVIEW ON CLIENT-SERVER BASED APPLICATIONS AND RESEARCH OPPORTUNITY," *International Journal of Recent Scientific Research,* vol. 10, no. 7(H), pp. 33857-33862, 2019.

[104] T. Sigdestad, "What is the difference between server-side and client-side?," Enonic, 26 01 2023. [Online]. Available: https://www.enonic.com/blog/what-is-the-difference-between-server-side-and-client-side. [Accessed 24 04 2024].

[105] Y. Duan, J. S. Edwards and M. X. Xu, "Web-based expert systems: benefits and challenges," *Information & Management,* vol. 42, no. 6, p. 799–811, 2004.

[106] Amazon Web Services, Inc., "What's the Difference Between a Web Server and an Application Server?," Amazon Web Services, Inc., [Online]. Available: https://aws.amazon.com/compare/the-difference-between-web-server-and-application-server/. [Accessed 24 04 2024].

[107] MDN Web Docs, "Introduction to the server side," Mozilla, Inc., [Online]. Available: https://developer.mozilla.org/en-US/docs/Learn/Server-side/First_steps/Introduction. [Accessed 24 04 2024].

[108] V. Wu, "How to use GitLab for Agile software development," GitLab, 05 03 2018. [Online]. Available: https://about.gitlab.com/blog/2018/03/05/gitlab-for-agile-software-development/. [Accessed 24 04 2024].

[109] Atlassian, "Scrum Board Basics: Getting Started with Agile," Atlassian, [Online]. Available: https://atlassian.com/agile/project-management/scrum-board. [Accessed 24 05 2024].

[110] ScrumAlliance, "What Is a Scrum Board?," ScrumAlliance, [Online]. Available: https://resources.scrumalliance.org/Article/scrum-board. [Accessed 24 05 2024].

[111] J. Wan, Y. Zhu and M. Zeng, "Case Study on Critical Success Factors of Running Scrum," *Journal of Software Engineering and Applications,* vol. 6, no. 2, pp. 59-64 , 2013.

[112] R. Ranjan, "Being too Agile – Why Agile fails and how to avoid it," DELAVAL INTERNATIONAL AB, 27 04 2022. [Online]. Available: https://www.linkedin.com/pulse/being-too-agile-why-fails-how-avoid-ravi-ranjan/. [Accessed 24 05 2024].

[113] "CoffeeScript," Jeremy Ashkenas, [Online]. Available: https://coffeescript.org/#overview. [Accessed 24 05 2024].

[114] M. Stueben, Good Habits for Great Coding, Falls Church, Virginia, USA: Apress, Berkeley, CA, 2018.

[115] Scrum.org, "What is Scrum?," Scrum.org, [Online]. Available: https://www.scrum.org/resources/what-scrum-module. [Accessed 11 08 2023].

[116] T. Sauvola, L. E. Lwakatare, T. Karvonen, P. Kuvaja, H. Holmström Olsson, J. Bosch and M. Oivo, "Towards Customer-Centric Software Development," in *2015 41st Euromicro Conference on Software Engineering and Advanced Applications*, Madeira, Portugal, 2015.

[117] M. Jørgensen, "Do Agile Methods Work for Large Software Projects?," in *19th International Conference, XP 2018*, Porto, Portugal, 2018.

[118] S. Nerur, R. Mahapatra and G. Mangalaraj, "Challenges of Migrating to Agile Methodologies," *COMMUNICATIONS OF THE ACM,* vol. 48, no. 5, pp. 73-78, 2005.

[119] J. López-Martínez, R. Juárez-Ramírez, C. Huertas and S. Jiménez, "Problems in the Adoption of Agile-Scrum Methodologies: A Systematic Literature Review," in *4th*

International Conference in Software Engineering Research and Innovation, Puebla, Mexico, 2016.

[120] K. M. Lui and K. C. Chan, "Pair programming productivity: Novice–novice vs. expert–expert," *International Journal of Human-Computer Studies,* vol. 64, no. 9, pp. 915-925, 2006.

[121] P. Baheti, E. Gehringer and D. Stotts, "Exploring the Efficacy of Distributed Pair Programming," in *XP/Agile Universe 2002*, Berlin, Heidelberg, 2002.

[122] S. Heiberg, U. Puus, P. Salumaa and A. Seeba, "Pair-Programming Effect on Developers Productivity," in *International Conference on Extreme Programming and Agile Processes in Software Engineering*, Berlin, Heidelberg, 2003.

[123] Office of the Under Secretary of Defense for Acquisition and Sustainment, "Operation of the Defense Acquisition System," 07 01 2015. [Online]. Available: https://www.acq.osd.mil/fo/docs/500002p.pdf. [Accessed 17 08 2024].

[124] Assistant Secretary of Defense for Mission Capabilities, "PDR and CDR Assessments," Office of the Under Secretary of Defense for Research and Engineering, [Online]. Available: https://ac.cto.mil/dte/pdr-cdr/. [Accessed 17 08 2024].

[125] Office of the Under Secretary of Defense for Research and Engineering, Systems Engineering Guidebook, Washington, D.C.: Office of the Under Secretary of Defense for Research and Engineering, 2022.

[126] AgileModeling, "Big Modeling Up Front (BMUF) Anti-Pattern," Ambysoft Inc., 2022. [Online]. Available: https://agilemodeling.com/essays/bmuf.htm. [Accessed 17 08 2024].

[127] K. Petersen, C. Wohlin and D. Baca, "The Waterfall Model in Large-Scale Development," in *Product-Focused Software Process Improvement 10th International Conference, PROFES 2009*, Oulu, Finland, 2009.

[128] S. H. VanderLeest and A. Buter, "Escape the waterfall: Agile for aerospace," in *2009 IEEE/AIAA 28th Digital Avionics Systems Conference*, Orlando, FL, USA, 23-29 October 2009.

[129] H. Saiedan and R. Dale, "Requirements engineering: making the connection between the software developer and customer," *Information and Software Technology,* vol. 42, no. 6, pp. Pages 419-428, 15 April 2000.

[130] M. Pemartin, A. I. Rodriguez-Escudero and J. L. Munuera-Aleman, "Effects of Collaborative Communication on NPDCollaboration Results: Two Routes of Influence*," *Journal of Product Innovation Management,* vol. 35, no. 2, pp. 147-297, Mar 2018.

[131] DoD Chief Information Officer, "DoD Enterprise DevSecOps Strategy Guide," Department of Defense, Washington, DC, March 2021.

[132] P. Hammant, "Trunk Based Development: Introduction," Trunk Based Development, 2017-2020. [Online]. Available: https://trunkbaseddevelopment.com/. [Accessed 25 08 2024].

[133] Amazon Web Services, "What is Unit Testing?," Amazon Web Services, Inc., [Online]. Available: https://aws.amazon.com/what-is/unit-testing/#:~:text=Unit%20testing%20is%20the%20process,test%20for%20each%20code%20unit.. [Accessed 25 08 2024].

[134] D. Radigan, "Story points and estimation," Atlassian, [Online]. Available: https://www.atlassian.com/agile/project-management/estimation#:~:text=Story%20points%20are%20units%20of,work%2C%20and%20risk%20or%20uncertainty.. [Accessed 25 08 2024].

[135] M. Cohn, "Daily Scrums: Synchronization Meetings, Not Status Meetings," Mountain Goat Software, 16 08 2023. [Online]. Available: https://www.mountaingoatsoftware.com/blog/daily-scrum-not-just-for-scrummasters. [Accessed 25 08 2024].

[136] D. Radigan, "What is a stand up meeting & tips to run one," Atlassian, [Online]. Available: https://www.atlassian.com/agile/scrum/standups. [Accessed 25 08 2024].

[137] M. Cohn, "Daily Scrum Meeting," Mountain Goat Software, [Online]. Available: https://www.mountaingoatsoftware.com/agile/scrum/meetings/daily-scrum. [Accessed 25 08 2024].

[138] S. McGee and D. Greer, "Towards an understanding of the causes and effects of software requirements change: two case studies," *Requirements Engineering,* vol. 17, pp. 133-155, 2012.

[139] K. E. Wiegers and J. Beatty, Software Requirements, 3rd Edition, USA: Microsoft Press, 2013.

[140] C. Ghezzi, "Formal Methods and Agile Development: Towards a Happy Marriage," in *The Essence of Software Engineering*, Cham, Switzerland, Springer, 2018, pp. 23-36.

[141] G. J. Miller, "Agile Problems, Challenges, & Failures," in *2013 PMI Global Congress Proceedings*, New Orleans, Louisiana, 2013.

[142] H. F. Hofmann and F. Lehner, "Requirements Engineering as a Success Factor in Software Projects," *IEEE SOFTWARE,* vol. 18, no. 4, pp. 58-66, 2001.

[143] A. Hussain, E. O. Mkpojiogu and F. M. Kamal, "The Role of Requirements in the Success or Failure of Software Projects," in *International Soft Science Conference (ISSC 2016)*, Universiti Utara Malaysia, Malaysia, 11-13 April 2016.

[144] E. B. Rogers III and S. W. Mitchell, "MBSE delivers significant return on investment in evolutionary development of complex SoS," *Systems Engineering,* vol. 24, no. 6, pp. 383-496, 2021.

[145] F. Glail, A. Moulton and S. Madnick, "Agile Project Dynamics: A System Dynamics Investigation of Agile Software Development Methods," 10 2014. [Online]. Available: https://dspace.mit.edu/handle/1721.1/103024. [Accessed 20 10 2024].

[146] L. Cao, B. Ramesh and T. Abdel-Hamid, "Modeling dynamics in agile software development," *ACM Transactions on Management Information Systems (TMIS),* vol. 1, no. 1, pp. 1 - 26, 2010.

[147] I. Fatema and K. Sakib, "Using Qualitative System Dynamics in the Development of an Agile Teamwork," *International Journal on Advances in Software,* vol. 11, no. 1 & 2, pp. 170-185, 2018.

[148] I. Fatima and K. Sakib, "Factors Influencing Productivity of Agile Software Development Teamwork: A Qualitative System Dynamics Approach," in *2017 24th Asia-Pacific Software Engineering Conference*, Nanjing, Jiangsu, China, 2017.

[149] F. Glail, "Agile Project Dynamics: A Strategic Project Management Approach to the Study of Large-Scale Software Development Using System Dynamics," 06 2012. [Online]. Available: https://dspace.mit.edu/handle/1721.1/79513. [Accessed 20 10 2024].

[150] K. Molokken-Ostvold and K. M. Furulund, "The Relationship between Customer Collaboration and Software Project Overruns," in *Agile 2007 (AGILE 2007)*, Washington, DC, USA, 13-17 August 2007.

[151] K. E. van Oorschot, K. Sengupta and L. N. van Wassenhove, "Dynamics of agile software development," in *Proceedings of the 27th International Conference of the System Dynamics Society*, Albuquerque, USA, 2009.

[152] Microsoft, "NORM.INV function," Microsoft, [Online]. Available: Returns the inverse of the normal cumulative distribution for the specified mean and standard deviation. [Accessed 03 10 2024].

[153] G. Doran, "There's a S.M.A.R.T. Way to Write Management's Goals and Objectives," *Management Review,* vol. 70, pp. 35-36, 11 1981.

[154] Atlassian, "Sprint Velocity in Scrum: How to Measure and Improve Performance," Atlassian, [Online]. Available: https://www.atlassian.com/agile/project-management/velocity-scrum. [Accessed 31 08 2024].

[155] JiraAlign, "Velocity by team (PI)," Atlassian, [Online]. Available: https://help.jiraalign.com/hc/en-us/articles/115004539788-Velocity-by-team-PI. [Accessed 31 08 2024].

[156] A. Smalley, "Art of Lean on Work & Waste, Part 7: Defects," Lean Enterprise Institute, 04 12 2020. [Online]. Available: https://www.lean.org/the-lean-post/articles/art-of-lean-on-work-waste-part-7-defects/. [Accessed 01 09 2024].

[157] D. Graziotin, F. Fagerholm, X. Wang and P. Abrahamsson, "What happens when software developers are (un)happy," *The Journal of Systems and Software,* vol. 140, pp. 32-47, June 2018.

[158] E. Wrubel, "Agile Software Teams: How they Engage with Systems Engineering on Department of Defense Acquisition Programs," Carnegie Mellon University Software Engineering Insitute, 24 11 2014. [Online]. Available: https://insights.sei.cmu.edu/blog/agile-software-teams-how-they-engage-with-systems-engineering-on-department-of-defense-acquisition-programs/. [Accessed 02 09 2024].

[159] M. Cottmeyer, "Embracing a Systems Engineering Approach to Agile at Scale," LeadingAgile, 18 05 2023. [Online]. Available: https://www.leadingagile.com/2023/05/embracing-a-systems-engineering-approach-to-agile-at-scale/. [Accessed 02 09 2024].

[160] S. E. Carpenter and A. Dagnino, "Is Agile Too Fragile for Space-Based Systems Engineering?," in *IEEE International Conference on Space Mission Challenges for Information Technology, SMC-IT*, Laurel, MD, USA, 24-26 September 2014.

[161] N. Agarwal and U. Rathod, "Defining success for software projects: An exploratory revelation," *International Journal of Project Management,* vol. 24, no. 4, p. 358–370, 2006.

[162] J. Pereira, N. Cerpa, J. Verner, M. Rivas and J. D. Procaccino, "What do software practitioners really think about project success: A cross-cultural comparison," *Journal of Systems and Software,* vol. 81, no. 6, pp. 897-907, June 2008.

[163] International Council on Systems Engineering, "Systems Engineering," International Council on Systems Engineering (INCOSE), [Online]. Available: https://www.incose.org/about-systems-engineering/system-and-se-definition/systems-engineering-definition. [Accessed 11 8 2023].

[164] V. K. Chauhan, "Smoke Testing," *International Journal of Scientific and Research Publications,* vol. 4, no. 2, 2014.

[165] K. Beck and C. Andres, Extreme Programming Explained: Embrace Change (2nd Edition), Boston, MA: Addison-Wesley Professional, 2005.

[166] J. M. Borky and T. H. Bradley, Effective Model-Based Systems Engineering, Cham, Switzerland: Springer Nature Switzerland AG, 2019.

[167] L. Rosser, P. Marbach, G. Osvalds and D. Lempia, "Systems Engineering for Software Intensive Projects Using Agile Methods," in *INCOSE International Symposium*, Las Vegas, NV, USA, 2014.

[168] M. Huss, D. R. Herber and J. M. Borky, "An Agile Model-Based Software Engineering Approach Illustrated through the Development of a Health Technology System," *Software,* vol. 2, p. 234–257, 2023.

[169] O. C. Z. Gotel and A. C. W. Finkelstein, "An Analysis of the Requirements Traceability Problem," in *Proceedings of IEEE International Conference on Requirements Engineering*, Colorado Springs, CO, USA, 1994.

[170] C. Hood, S. Wiedemann, S. Fichtinger and U. Pautz, Requirements Management: The Interface Between Requirements Development and All Other Systems Engineering Processes, Berlin, Germany: Springer, 2008.

[171] A. van Lamsweerde, "Goal-oriented requirements engineering: a guided tour," in *Proceedings Fifth IEEE International Symposium on Requirements Engineering*, Toronto, ON, Canada, 2001.

LIST OF ABBREVIATIONS

AAF	Adaptive Acquisition Framework
AQDR	Additional Qualification Designator Report
CDR	Critical Design Review
CI/CD	Continuous Integration / Continuous Development
CMP	Configuration Management Plan
CMMI	Capability Maturity Model Integration
CONUS	Inside the 48 Contiguous United States and the District of Columbia
COTS	Commercial Off the Shelf
CV	US Navy Abbreviation for Aircraft Carrier
CRRB	Change Request Review Board
DCS	DoD Control System
DE	Domain Engineering
DESS	DoD Data Exchange Software System
DLMS	Defense Logistics Management System
DMSMS	Diminishing Manufacturing Sources and Material Shortages
DoD	Department of Defense
DoDAF	Department of Defense Architectural Framework
DPAS	DoD Process Automation System
DQMS	DoD Qualification Management System
DSHS	DoD Software Hosting System
DTSS	DoD Tactical Support System
DWMS	DoD Work Management System
ERB	Enhancement Review Board
EVM	Earned Value Management
FCA	Functional Configuration Audit
IDE	Integrated Development Environment
IETMS	Interactive Electronic Technical Manuals

IMA	Intermediate Maintenance Activity
IMF	Intermediate Maintenance Facility
In	Iteration Number
IPT	Integrated Product Team
LADRA	Legacy Application Deployment Readiness Activity
LRIP	Low-Rate Initial Production
FRP	Full Rate Production
MBSE	Model Based Systems Engineering
MBSR	Model-based Structured Requirements
MDA	Milestone Decision Authority
MDE	Model Driven Engineering
MVP	Minimum Viable Product
MVCR	Minimum Viable Capability Release
NMCI	Navy & Marine Corps Intranet
NSWC	Naval Surface Warfare Center
NUWC	Naval Undersea Warfare Center
OBE	Overcome by Events
OCONUS	Outside the Contiguous United States
OMIS	Obsolescence Management Information System
OUSD	Office of the Undersecretary of Defense
PCA	Physical Configuration Audit
PDR	Preliminary Design Review
PPR	Project Performance Review
PRR	Production Readiness Review
PSNS	Puget Sound Naval Shipyard
QCL	Qualification, Certification, Licenses
R4J	Requirements for JIRA
RBR	Release Backlog Review
Rn	Release Number
RMB	Requirements Management Board

RPM	Registered Publication Memorandums
SAFe	Scaled Agile Framework
SETR	Systems Engineering Technical Reviews
SOS	System of Systems
SQL	Structured Query Language
SRB	Stakeholder Review Board
SRD	Software Requirements Document
SRR	Software Requirements Review
SRS	Software Requirements Specification
SRT	Submarine Readiness Tracker
SSBN	US Navy Abbreviation for Sub-surface Ballistic Nuclear Submarine
SSN	US Navy Abbreviation for Sub-surface Nuclear Submarine
T&E	Test & Evaluation
TMIS	Torpedo Management Information System
TRR	Test Readiness Review
TSU	Trainer Scheduling Unit
TYCOM	Type Commander
UDT	User Design Team
UI	User Interface
UML	Unified Modeling Language
USN	United States Navy
USMD	United States Marine Corps
V&V	Verification and Validation