DoD Software Acquisition: Can DoD Build Weapon Systems Software Better and Faster

Final

Carmela B. Rice

Defense Acquisition University

Senior Service College Fellowship 2018-2019

Huntsville, Alabama

11 April 2019

This research paper is presented to the Defense Acquisition University for partial fulfillment of the academic requirements for the Army's Senior Service College Fellowship (SSCF) under the direction of SSCF Director, William A. Colson, and Research Advisor Steve Monks.

Approval Page

Title: DoD Software Acquisition: Can DoD Build Weapon systems software Better and Faster

Author: Carmela B. Rice

Organization: Defense Acquisition University (DAU) - SSCF

Date of Paper: 11 April 2019

Informed Consent Forms: Not required

Research Advisor [Steve Monks.] Approval Date: 18 April 2019

Second Reviewer [Dana Stewart] Approval Date: 1 May 2019

SSCF Director [William A. Colson] Approval Date: 12 May 2019

OPSEC [Carolyn Farmer] Approval Date: 10 Aug 2019

Table of Contents

Table of Figures

Abstract

Software is a crucial element of almost every weapon system that DoD acquires and the role of software in maintaining national security is growing. DOD has encountered significant challenges in its ability to acquire software for weapon systems for over 40 years, and many of those challenges have remained the same for decades. The purpose of this study is to identify the challenges that DoD faces when acquiring software-intensive weapon systems that cause weapon systems software to be late and not provide desired performance and to evaluate changes to the software acquisition process that can address these challenges. DoD can benefit from adopting Agile methods, and there are other initiatives required to fully mitigate the challenges in DoD software acquisition.

Chapter 1 - Introduction

Background

"S*oftware is inherent in today's complex systems and is often the primary cost, schedule, and technical performance driver in Department of Defense (DoD) programs" (*Heil, 2017, p. 5). DoD has encountered significant challenges in its ability to acquire software for weapon systems for over 40 years. A Government Accountability Office (GAO) report from 1978 states that costs for weapon systems software development during the 1970s increased steadily while software performance remained unreliable. GAO (1978) concluded in the same report that schedule growth and degraded performance could be attributed to the way software was designed and built. Software has become the centerpiece of technological advancements, and it is a vital element of weapon systems needed to maintain US superiority. To address these technical challenges during the 80s and 90s, DoD created and enforced software development standards such as MIL-STD-2167 (DoD, 1985) and MIL-STD-498 (DoD 1994). These standards established software development activities and documentation requirements mandating that weapon systems software developers use best practices on DoD software programs. At that time, these standards prescribed the Waterfall Software Development Process. The waterfall process requires all software requirements to be defined up front, followed by all software design before building the software (Munassar & Govardhan, 2010).

In 1986, DoD tasked the Software Engineering Institute (SEI) to establish a method that would allow DoD to determine the goodness or maturity of contractors' software development capability for DoD programs. The result was the beginning of the Capability Maturity Model

(CMM) (Paulk, 1993). The CMM model eventually evolved to the CMM-Integration (CMMI) model, which “is a framework of best practices that can be applied in any software development program” (Miller & Ward, 2016, p. 3). Both models define levels of maturity against which DoD contractors could be assessed. CMMI assessments became the standard on DoD software development contracts.

During the eighties and nineties, both commercial and defense industries discovered that it was not realistic to define all software requirements up front. Defense Science Board (DSB) (1987) states “Five developments in the past decade have revolutionized the software scene. DoD software practices evolved in the '60's and '70's, and they neither take into account nor utilize these advances” (p. 9). In iterative processes, a software application or system is broken up into multiple software releases. Part of the system is developed in the first release, and each subsequent release adds more functionality. This type of process allows software requirements to be defined and clarified over time by iteration (Munassar & Govardhan, 2010).

During the 1970s and 1980s, DoD was a key driver for industry software development because DoD was a very significant customer of the industrial base. During the 2000s, the boom in mobile computing caused the commercial industry to look for alternative software development strategies that would allow them to get to market faster (DSB, 2018). As such, the commercial industry has evolved software development practices even further. In particular, the commercial industry has adopted Agile software development processes to fit commercial needs to get to market as quickly as possible. Agile processes are similar to traditional iterative processes, except the Agile methods focus on shorter development cycles; require less documentation and more frequent software builds; embrace change at the beginning of each

iteration instead of following an obsolete plan; and emphasize frequent interaction between stakeholders (The Agile Alliance, 2001). The question for DoD is will the current processes used in commercial industry be sufficient to address the challenges that remain in DoD weapon systems software development.

In 2017, the 115th Congress enacted a pilot program to use Agile commercial practices on select DoD programs. The purpose of the pilot is to identify ways to simplify software development methods to improve software acquisition. In 2018, a Defense Science Board (DSB) task force issued a memorandum and report to the Under Secretary of Defense for Research and Engineering that provided seven recommendations to improve software acquisition (DSB, 2018). One of the major recommendations is for DoD to adopt an Agile, or continuous-iterative software development process (DSB, 2018).

Purpose of This Study

Challenges in weapon systems software development have been essentially the same for decades, despite revisions to DoD 5000 and to standards like MIL-STD-2167A. The purpose of this study is to identify the challenges that DoD faces when acquiring software-intensive weapon systems and to evaluate changes to the software acquisition process that can address these challenges. The commercial industry, unlike the defense industry, has evolved over the years to improve the delivery of commercial software (DSB, 1987). The latest evolution includes the adoption of Agile software development methods.

Though there are similarities between commercial software and DoD software, there is a significant difference between software developed for commercial purposes and software

developed for DOD. Software for weapon systems has far-reaching impacts when it comes to safety and loss of life. There is significant research on the various Agile techniques used in commercial industry, and DoD has documented software development challenges in various reports. The missing research is a comparison of those challenges and the benefits of using Agile methods to determine if the adoption of Agile methods can address the challenges that DoD has faced for decades.

Problem Statement

DoD must streamline the software acquisition process to acquire software that meets the needs of the military and do so at the "speed of relevance" in order to meet this objective of the National Defense Strategy (NDS) 2018. DoD must deliver accurate, reliable, and sustainable software products to the field faster in order to keep pace with evolving threats. Thus, DoD must evaluate the next paradigm in DoD software acquisition. Literature shows that the Agile software development lifecycle model has become very prevalent in the commercial industry. DoD, under the direction of Congress and the Under Secretary of Defense for Acquisition & Sustainment, is currently evaluating the use of Agile software methods for weapon systems software development. This paper addresses the following questions:

1) What are the challenges in DoD weapon systems software development that cause software cost and schedule growth and software deficiencies?
2) What can be done to mitigate the issues that have plagued DoD weapon systems software development?

3) How can Agile software development methods address challenges in the development of DoD weapon systems software?

Significance of This Research

The 2018 DSB task report describes the role of software in weapon systems and the national security mission as crucial and growing because today, most of our weapon capabilities are attributed to software (DSB, 2018). Additionally, software in DoD weapon systems has and will continue to grow in size and complexity. This research explores reports and other literature to determine if adopting Agile software development methods can mitigate the challenges that drive the trend of late software deliveries and software deliveries that do not fully meet user needs. Improving DoD ability to deliver software intensive weapon systems is critical to enabling the US to keep pace with adversaries.

Chapter 2 - Research Methodology

The research methodology for this study is a literature review of reports, white papers, journals, DoD documents, briefings, and websites.

Literature Review Focus and Approach

There have been numerous studies and reports documenting the challenges in DoD software development. This study focuses on reports generated by the Defense Science Board (DSB), Government Accountability Office (GAO), MITRE, Software Engineering Institute (SEI) and others in order to document the challenges that DoD has experienced over the last four

decades. There are also many reports, journals, and other documents that discuss the use of Agile software methods. This study culls document content based on information related to the use of Agile methods, how it applies to large complex development and the overall benefits.

Limitations and Recommended Areas of Additional Research

This research does not include technical details of software development or specific management practices. Furthermore, this paper does not explore the different brands of Agile methodologies or the distinctions between them. Additional research and empirical data are needed to understand the long-term effect of using Agile processes on large-scale complex software systems, particularly in terms of sustainability and life-cycle cost.

Chapter 3 - Literature Review

The purpose of this literature review is to research relevant literature associated with the challenges associated with the development of weapon systems software programs and how Agile software development methods can help. The following is a review of documents relevant to this study.

Over Forty Years of Software Development Challenges in DoD

Historical weapon systems software challenges documented in the 1970s.

In 1975, the Office of Secretary of Defense (OSD) tasked the Applied Physics Laboratory of John Hopkins University, the MITRE Corporation, and the DoD software

management steering committee to conduct a study of DoD weapon systems software programs. The report cites requirements changes as the most significant factor in cost growth. Other challenges documented by Kossiakoff (1975) include a lack of visibility into the software development effort, the ability to measure software development progress, interface management, and lack of proper planning. Lack of sound software systems engineering was cited as the root cause for many critical problems, followed by poor architecture, evolving threats, advancing technology, and defects discovered late in the process (Kossiakoff, 1975).

In 1978, the GAO reviewed software management in selected weapon system programs. The resulting report states that although software is a critical element of operational performance, DoD does not place requisite emphasis on software efforts as it does for hardware development (GAO, 1978). As a result, mission performance is degraded, life-cycle costs are dramatically increased, and development schedules are extended. GAO (1978) cites the cause of these problems to be a result of technical problems related to the way that software was designed, developed, and tested.

Evolving weapon systems software challenges documented in the 1980s.

In 1987, the Defense Science Board (DSB) Task Force for Military Software issued a report that recommended both technical and managerial changes to DoD software acquisition. DSB (1987) states that software requirements definition is the most difficult and the most essential part of software development. A key finding of DSB (1987) is that attempts to completely specify software requirements before building software are at the core of many software problems in DoD. DSB (1987) states "The systems built today are just too complex for

the mind of man to foresee all of the ramifications purely by the exercise of the analytic imagination" (p. 7). Another key finding in DSB (1987) is that early prototyping and early user involvement is necessary to overcome this difficulty of specifying requirements. Early and frequent user involvement also makes software development processes more effective.

The need for early prototyping and user involvement motivated industry to evolve software development processes. In the commercial sector, iterative development processes are considered modern best practices; however, DoD policy never evolved from the 1970s and remains based on the traditional waterfall process (DSB, 1987). According to DSB (1987), "Software engineering methods and techniques have dramatically advanced over the last decade, yet these techniques are generally not practiced in DoD" (p. 16). Iterative software development processes allow users to operate prototype and incremental versions of software and to provide feedback to the developers, who in turn revise the specification and design of the system (DSB, 1987). This iteration between users and developers overcomes the problem of trying to specify the entire system up front. DSB (1987) states “We believe that users cannot, with any amount of effort and wisdom, accurately describe the operational requirements for a substantial software system without testing by real operators in an operational environment and iteration on the specification” (p. 7). Unfortunately, in DoD, the end user of weapon systems is separated from the developer by layers of organizations. Because the waterfall process requires that all requirements and design are specified up front, it takes longer to produce a release of the software (DSB, 1987). Long development cycles and software delays often impact the path to operational capability because users do not have a chance to operate the system until the end of development (DSB, 1987).

The DSB also noted in 1987 that there is a shortage of personnel with software development expertise to staff DoD programs and expected that the shortage would continue. Studies have recommended actions to create additional training and career paths to establish more uniformed specialists. DoD has taken some action, but the shortage persists. DSB (1987) states that the best programs were staffed with personnel counts that are 10% of the number of contractor personnel, and most programs are not staffed that well.

In 1989, the US Air Force tasked a committee to review existing studies of weapon systems software development in order to determine why software acquisition problems had not been resolved. Air Force systems' reliance on software has steadily increased over the years, and 17 studies have shown software to be an issue in system development. Despite all of the studies and recommendations, problems persist (Air Force Studies Board, 1989). As part of the task, the committee also reviewed existing programs in order to determine the cause of software deficiencies. The study (Air Force Studies Board, 1989) also included investigation of user-developer interactions; prototyping; software development process alternatives; and controls for testability, reliability, and survivability. Overall, the committee focused on three perspectives: processes, people, and technology required to develop software. The committee grouped its findings into five issues: the acquisition process, acquisition practices, software engineering, in-house expertise in software, and software technology (Air Force Studies Board, 1989).

The acquisition process has always assumed a waterfall process for software development, which is not iterative in nature. Waterfall projects are often based on unrealistic expectations, and success is measured by documentation vice actual successful software results (Air Force Studies Board, 1989). Air Force Studies Board (1989) states that the Waterfall model

would be sufficient for systems that are well understood and where system developers have substantial experience in developing such systems. In cases of developing new technology or unprecedented systems, this approach is high risk (Air Force Studies Board, 1989).

Air Force Studies Board (1989) goes on to note that DoD has continued to use firm fixed price contracts to acquire unprecedented systems and that more user involvement is required. It is necessary to have "the right people at the right time" (Air Force Studies Board, 1989, p. 2) with the requisite experience and knowledge in order to appropriately tailor acquisition practices for any given program. Early attention to software development before going into full-scale development is also important. Additionally, Air Force Studies Board (1989) notes the need for disciplined software engineering techniques and tools and for program managers to understand and monitor progress. Commercial tools exist to aid in software development, but the Air Force has been slow to adopt them. Unfortunately, there is also a shortage of in-house personnel resources with adequate software skills; and the need for such personnel is continuing to grow faster than the available resources (Air Force Studies Board, 1989).

Continued weapon systems software challenges documented in the 1990s.

In 1994, the Under Secretary of Defense (Acquisition & Technology) tasked the DSB Task Force on Acquiring Defense Software Commercially to perform a study that would determine when it is appropriate to use commercial practices in the procurement of DoD software, and to identify any required changes to existing acquisition practices in order to use these practices. The Task Force found "indications that commercial development efforts have achieved better predictability and lower costs" (DSB, 1994, p. 1), though there is not a

significant amount of credible data to substantiate this assessment. DSB (1994) notes that software is the overall cost and schedule driver for weapon systems and that software budgets are expected to grow and require greater management control. According to DSB (1994), “Over the last two decades, a number of DoD software development efforts have gotten into trouble, particularly in terms of actual costs and schedule vs. expected or predicted costs and schedule” (p.17). A review of briefings from program managers and other inputs revealed the following reasons that DoD software programs encounter such significant problems:

- Poor Requirements Definition
- Lack of User Involvement in Development Process
- Inability of Users to Foresee Benefits of Automation Without Incremental Capability
    - Inadequate Software Process Management and Control by Contractor
    - Lack of Integrated Product Teams
    - Failure to Establish "Team" With Vendors and Users
- Little Participation of Functional Area Experts
- Ineffective Subcontractor Management
- Lack of Consistent Attention to Software Process
- Too Little Attention to Software Architecture
- Poorly Defined and Inadequately Controlled Interfaces Between Computer Hardware, Communications and Software
- Assumption That Software Upgrades Can "Fix" Hardware Deficiencies (Without Assessment of Cost and Schedule Risks)
- Focus on Innovation Rather than Cost and Risk
- Limited or No Tailoring of Military Specifications Based on Continuing Cost-Benefit Evaluations. (DSB, 1994, p. 17)

Like previous studies, DSB (1994) notes that the acquisition policy assumes a waterfall approach instead of an incremental approach highlighting the bureaucratic impediments to change. Though the commercial base is evolving, DoD would have to make major changes in order to evolve with it (DSB, 1994). At the time of the DSB (1994), there were two sets of

acquisition policies for software: one for embedded software, and another for software that can command and control weapon systems even though issues that arise for both types of software are the same. The study included further analysis of software needs across different types of software and determined that real-time software and command and control software are the types of software that are most sensitive to errors. DoD Automated Information System software, on the other hand, has a low sensitivity to errors (DSB, 1994). Weapon systems software is predominately comprised of real-time and command and control applications.

Assessment of the DoD Software Acquisition Approach revealed some strengths such as highly structured processes, tightly defined requirements, and high-quality products for mission-critical systems that require low failure rates (DSB, 1994). However, the weaknesses of the DoD acquisition approach include high lifecycle costs, long timeframe from concept to fielding, excessive documentation, excessive formal reviews, excessive testing of non-critical systems, poor communication between developer, acquirer, and the user, no cost or schedule requirements, use of traditional waterfall approach, little requirements relaxation for high cost items, inadequate beta testing, and little focus on reusability (DSB, 1994, p. 15).

Emphasis on establishing a “well-formulated” software architecture is necessary to enable software reuse and to promote the flexibility needed to support changes to software after it has been built; and DoD has paid “Too little attention to software architecture” (DSB, 1994, p. 25). The DSB task force determined that software architecture is extremely important to achieve DoD desired results on weapon system programs. DSB (1994) notes that the lack of emphasis on software architecture precludes DoD from benefiting from evolutionary development, early (and

frequent) involvement of users, reuse, and the ability to respond to requirements changes with minimum cost and schedule impacts (DSB, 1994, p.8).

Another key finding documented in DSB (1994), which mirrors earlier studies, is the shortage of personnel with expertise in software acquisition, evaluation of software, software maintenance, software technology, software program management, and general software domain expertise (DSB, 1994, p.25). The DSB task force recommended that DoD address this personnel shortage by establishing education and training initiatives across the department, adding qualified software personnel to DoD, and "rotating government and contractor personnel between PM and developer organizations to build understanding and trust" (DSB, 1994, p.8). The DSB task force also recommended that DoD make better use of the Intergovernmental Personnel Act (IPA), which would allow the DoD to temporarily assign personnel from industry or academia to acquisition staff positions (DSB, 1994).

The DSB task force concluded that using more modern software practices and tools could alleviate many of the challenges that DoD has faced. There is some evidence that commercial software development efforts have achieved better results, though there is some concern with the reliability of the data suggesting commercial efforts may be more cost-effective (DSB, 1994, p.19). Also, as in previous reports, DSB (1994) recognizes the need for early user involvement in the software development effort. "Further, it is difficult for users to be significantly involved until late in a software development process, unless some sort of prototype can be constructed" (DSB, 1994, p. 23).

There are also program management issues that contribute to software acquisition challenges. According to DSB (1994), "DoD PMs place little emphasis on life-cycle issues (such

as software maintenance and support). Existing DoD-wide software policies, methodologies, and procedures, and their implementation by PMs are inadequate" (p. 23). There is a reluctance to accept partial solutions in order to achieve cost objectives, and Program Managers (PMs) are not incentivized to invest in "corporate" solutions that would facilitate software reuse across the DoD enterprise (DSB, 1994).

In 1994, the Deputy Assistant Secretary of the Air. Force (Communications, Computers, & Support Systems) (DASAF(CCSS)) spoke at the annual Software Technology Conference in Salt Lake City, Utah, saying

> From a Pentagon perspective, it is not the fact that software costs are growing annually and consuming more and more of our defense dollars that worries us. Nor is it the fact that our weapon systems and commanders are becoming more and more reliant on software to perform their mission. Our inability to predict how much a software system will cost, when it will be operational, and whether or not it will satisfy user requirements, is the major concern. (Air Force Software Technology Support Center, 1996, p. i)

DoD leaders want reliable systems that meet user needs to be delivered on time and within budget. In 1996, the Air Force Software Technology Support Center wrote the third version of Guidelines for Successful Acquisition and Management of Software-Intensive Systems to achieve this goal.

Because many software acquisitions are delivered late, never used, or canceled after many wasted years and millions of dollars, Air Force Software Technology Support Center (1996) dedicates a chapter to discussing the reasons that acquisition programs fail and how to recognize these problems. Challenges that plagued DoD software acquisitions 20 years earlier

still plagued DoD in 1996. In 1995, the Senior Defense and International Affairs Advisor to the Comptroller General of the United States testified before the House Committee on Budget and was quoted as saying:

> Over the years, we have reported on the persistent problems that have plagued weapons acquisition. Many new weapons cost more, are less capable than anticipated, and experience schedule delays. These problems are typical of DoD's history of inadequate requirements determination for weapon systems; projecting unrealistic cost, schedule, and performance estimates; developing and producing weapons concurrently; and committing weapon systems to production before adequate testing has been completed. (Air Force Software Technology Support Center, 1996, p. 1-5)

Similar to the DSB report from 1994, the guidelines documented in Air Force Software Technology Support Center (1996) suggest that DoD move away from focusing on specific requirements for single systems, and instead focus on architectures that facilitate product lines and software reuse.

Gibbs (1994) discusses the fact that building software is hard and that there is no "silver bullet" to resolve the problems that DoD has with building software and cautions practitioners about the "buzzword du jour" that surfaces periodically. Buzzwords surface from suggested paradigm shifts that claim to solve software problems. Examples of these buzzwords include structured programming in the 1970s, object-oriented programming in the 1980s, and computer-assisted software engineering in the 1990s. Chapter 1 of Air Force Software Technology Support Center (1996) reiterates the point made by Gibbs (1994), and also highlights the perception that

one of the biggest problems in software programs is poor management; a perception that has been highlighted in other reports such as DSB (1994). Software experts quoted in Air Force Software Technology Support Center (1996) state that software should be developed incrementally and that we must grow the best and brightest software talent in people. This finding is consistent with earlier reports that cite the need for more modern software practices and skilled personnel. Some specific issues noted in Air Force Software Technology Support Center (1996) include the need for formal methodologies (disciplined systems engineering process), need for phased (incremental) development, establishing the basis for cost and schedule estimation before the system and requirements are understood, unstable requirements, need for more user involvement, lack of communication, and the need for system acquisition training with a focus on software.

Current weapon systems software challenges documented since 2000.

In 2000, the Under Secretary of Defense (Acquisition, Technology, & Logistics) tasked the DSB Task Force on Defense Software to perform another study that would determine when it is appropriate to use commercial practices in the procurement of DoD software. At the conclusion of this study, the DSB submitted a report of findings and recommendations. DSB (2000) stated in the report cover letter that “the majority of problems associated with DOD software development programs are a result of undisciplined execution” (p. i). The DSB (2000) report contained major findings, around which it structured a series of recommendations. Specifically, DSB (2000) stated that the problems discovered in the study were fundamental:

> Too often, programs lacked well thought-out, disciplined program management and/or software development processes. Meaningful cost, schedule, and requirements' baselines were lacking, which prevented any possibility of tracking progress against them. In addition, there were numerous examples where the acquisition and/or contractor team lacked adequate software skills to execute the program. (p. 19)

These observations very closely mirror findings that have been documented in other reports for over a decade. The Task Force also noted in DSB (2000) that many recommendations were provided in the prior studies that remain unimplemented. In fact, DSB (2000) found that only 18 recommendations had been put into policy, and only 3 recommendations had been put into practice out of 134 total recommendations (DSB, 2000, p. 3).

In 2008, the Secretary of the Air Force for Acquisition developed a guidebook to support management of Air Force weapon system programs. Prior to listing guidelines, Secretary of the Air Force for Acquisition (2008) documented software challenges experienced by the Air Force on weapon systems software programs, including unrealistic expectations, unstable requirements, ineffective risk management, inadequate staffing of skilled software development teams, effective software processes not established or applied consistently. The root cause of many program failures has been determined as lack of systems engineering, ineffective management and lack of insight into software development effort. Unprecedented systems are difficult to estimate, and estimates are often extremely optimistic (Secretary of the Air Force for Acquisition, 2008). Inadequate or misinterpreted metrics often result in late notification of software problems or delays. "The ability to successfully develop large, complex, unprecedented

software systems in part depends upon having effective processes institutionalized and implemented with discipline" (Secretary of the Air Force for Acquisition, 2008, p. 8).

In 2017, the Naval Surface Warfare Center in Dahlgren compiled a list of common challenges that DoD has sustained over the years contributing to inconsistent execution of software projects and documented some approaches and techniques to facilitate successful project execution. Heil (2017) notes that:

> One of the primary root causes for many software project failures is that the effort was poorly estimated; or that the effort was accurately estimated but the program's senior leader drove the development organizations to 'accept the challenge' of significantly reduced cost and accelerated schedule without reducing the planned capabilities or requirements. (p. 6)

Some of the other challenges said to contribute to project failure include poor requirements and interface management, limited enforcement of best software engineering practices, lack of frequent communication across disciplines, lack of risk management, lack of software engineering expertise in program offices, and failure to apply lessons learned (Heil, 2017). Also, according to Clements (2017), the Army Director for Acquisition Career Management (DACM) Office is assessing the health of workforce skills, yet none of the career fields established by the DoD have a specific focus on software.

In 2018, the DSB conducted a study to determine how iterative software development practices used in commercial industry could be applicable to DoD, DSB (2018) notes that "In the acquisition of new systems, software drives program risk for approximately 60 percent of programs" (p. 14). The current DoD approach to mitigate these risks is to attempt to complete

detailed requirements up front (DSB, 2018), even though previous studies have determined the impossibility of effectively specifying the system completely at the beginning of a project (DSB, 1987). DSB (2018) also points out the importance of software architecture, stating that software problems often come about as a result of undetected problems in the underlying architecture. DSB (2018) is the most recent report, in a long list of reports that date back to 1975, that documents the challenges of DoD software acquisition. The following sections will provide basic software development concepts and discuss where proposed changes that can mitigate software acquisition challenges.

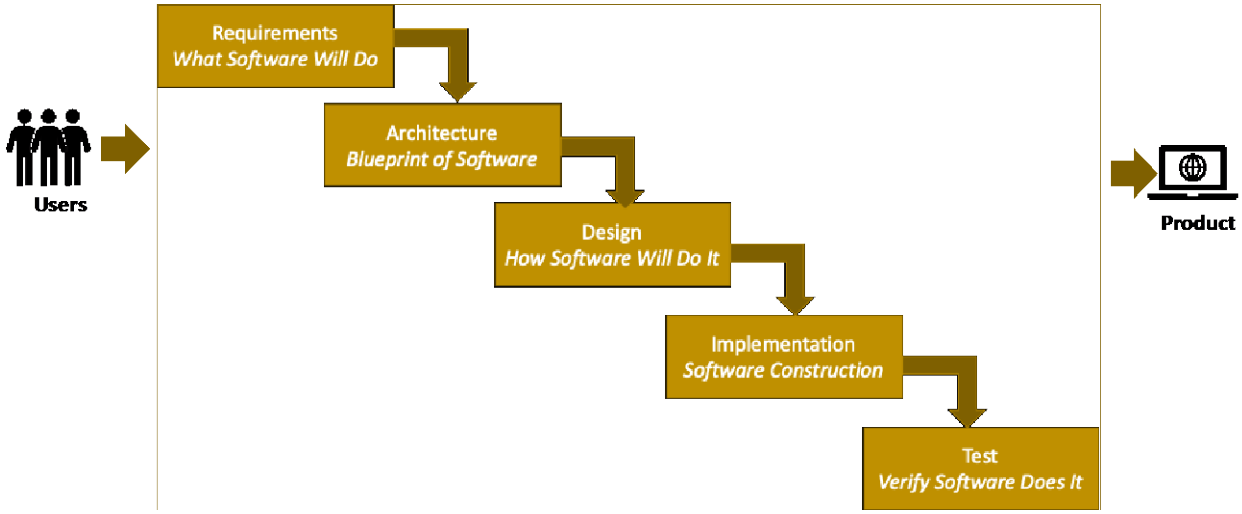
Software Development Paradigms

The purpose of this section is to familiarize the reader with software development terminology and basic software development principles in order to facilitate meaningful conversation on the subject. A software development process is a description of who does what and when in order to develop a new software product or to modify an existing software product, and it includes guidelines and best practices to reduce risk and deliver quality software (Jacobson, Booch, & Rumbaugh, 1999). A software process model, as defined by Munassar & Govardhan (2010), is “an abstract representation to describe the process from a particular perspective" (p. 95). In short, software process models describe how engineers, coders, testers, and to a degree managers, will work together in order to build a software product. Munassar & Govardhan (2010) presents five software process models and discusses the advantages and disadvantages of each. The authors chose some of the most common models in use during 2010

for comparison. Among others, (Munassar & Govardhan, 2010) compares the Waterfall Software Process Model and the Iterative Software Process Model.

The waterfall model is one of the oldest and most widely used software process models on government programs and is considered to be the "classical model of software engineering" (Munassar & Govardhan, 2010, p. 95). The waterfall model also serves as the foundation for other software models. As shown in Figure 1, the waterfall model is comprised of a series of sequential steps that when depicted pictorially, looks like a waterfall. Each step in the process is depicted as a stepping stone that flows down into the next step of the process.

Figure 1: Waterfall Software Process



The process starts by defining system requirements, which includes hardware requirements, design constraints, and design component decisions. System requirements tell the engineers at a high level what the system has to do. From there, software requirements are defined. Software requirements define specific expectations for what the software is supposed to do under specified conditions, how the software interacts with other software applications or design components,

and how well the software needs to perform (for example, how fast the software runs, how much memory it uses, and so forth) (Munassar & Govardhan, 2010). Architecture definition is the next step. Engineers identify and define major software components, the interaction between those software components, and interaction between software components and external interfaces. Following the architectural design step is the detailed design step. In this step, engineers specify how each component will be constructed. Next, implementation (coding) is the step where software is actually constructed. The testing step is where engineers run the software in a test environment in order to identify any errors in the software and to determine if the software meets its requirements (Munassar & Govardhan, 2010). Finally, the maintenance step begins when the software is released to users and continues as long as the software is in use. During the maintenance step, developers correct errors identified by the users and build updates to add more functionality or insert new technology (Munassar & Govardhan, 2010).

One of the most significant criticisms of the waterfall process is the fact that the actual construction of software begins very late in the process. As a result, the end users in the field do not have the opportunity to put hands on an actual product before the product is complete and delivered. Additionally, the documentation required by the Acquisition system is sometimes excessive (Munassar & Govardhan, 2010). During early phases of the project, project success and status is measured by documentation as it is completed; and it is difficult and expensive to make changes to documentation after completing the phase in which the document is developed. Another key issue with this approach is the fact that it is not realistic to “expect accurate requirements so early in the project” (Munassar & Govardhan, 2010, p. 96). In projects that use the waterfall model, however, users and other stakeholders expect all of the requirements to be

defined upfront; and this often leads to projects being disconnected from reality. The waterfall model does have some benefits. The process is simple and easy to understand, and it is based upon and reinforces good habits of software engineering which are widely known (Munassar & Govardhan, 2010).

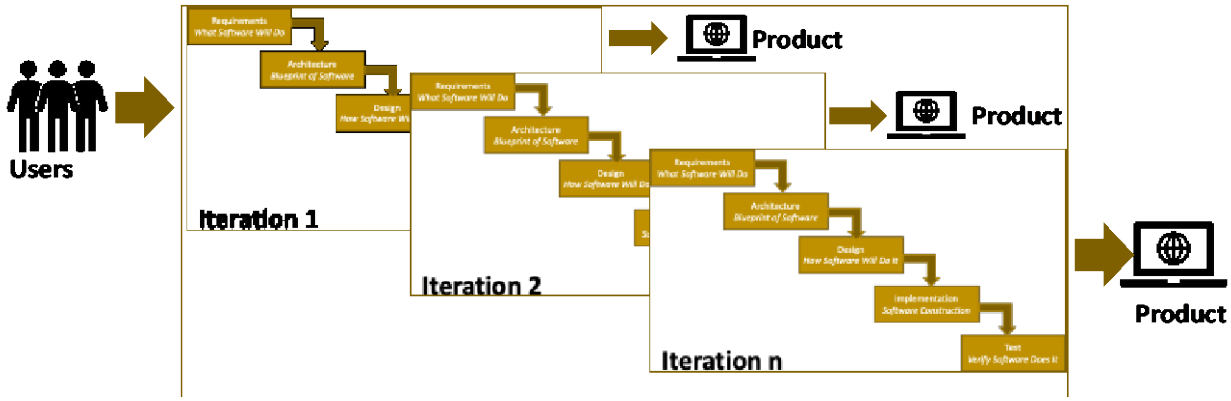
The shortcomings of the waterfall model prompted the establishment of another popular process model called Iterative Development. According to Munassar & Govardhan (2010),

> The problems with the Waterfall Model created a demand for a new method of developing systems which could provide faster results, require less up-front information, and offer greater flexibility. With Iterative Development, developers divide the project into small parts. This division allows the development team to demonstrate results earlier on in the process and obtain valuable feedback from system users. Often, each iteration is a mini-Waterfall process with the feedback from one phase providing vital information for the design of the next phase. In a variation of this model, programs can produce prototypes or release incremental versions of the software. (Munassar & Govardhan, 2010)

The iterative process model, as depicted in Figure 2**,** builds upon the waterfall model because the iterative process is comprised of a series of “iterations”, which are often called mini-waterfall projects. Each mini-waterfall cycle results in a partial product consisting of executable software which can be evaluated for progress. Each step in the waterfall process is executed in each

iteration, in sequence, culminating in a software release at the end of each iteration. Lessons learned from one iteration feed into the next iteration (Munassar & Govardhan, 2010).

Figure 2: Iterative Software Process



The iterative model results in the creation of an initial product sooner, alleviates the need to complete all requirements up front, provides an opportunity for the end user to clarify requirements, and provides a more flexible way to manage the project (Munassar & Govardhan, 2010). The iterative software model allows the development team to plan a little, build a little, and integrate and test a little by partitioning the system into small manageable pieces (Jacobson, Booch, & Rumbaugh, 1999). Testing is conducted at the end of each iteration in order to expose software defects or problems that can be addressed in the next iteration. According to (Jacobson, Booch, & Rumbaugh, 1999), iterative software development alleviates technical risks and results in better software. A significant risk in software development is not building the system that the users actually need; and to that end, it is essential to get the requirements right through iterating and prototyping (Jacobson, Booch, & Rumbaugh, 1999, pp. 332 - 333).

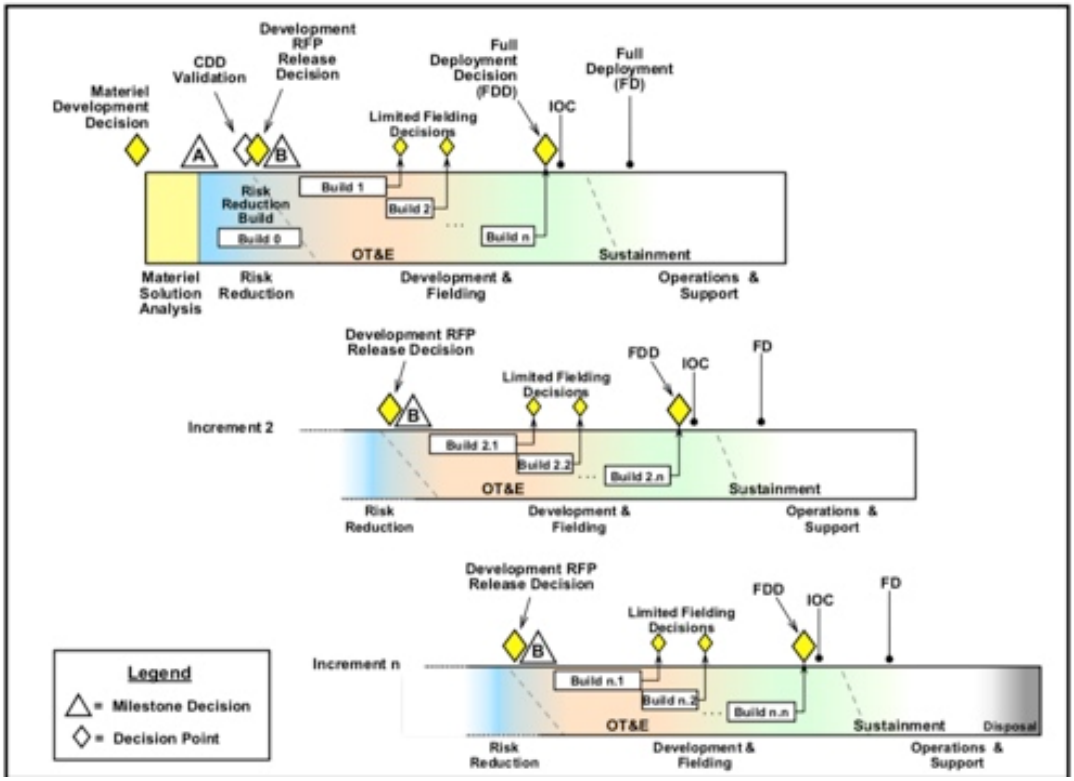
Pressman (2010) describes several prescriptive models of software engineering, which were “originally proposed to bring order to the chaos of software development” (p. 38). Starting with a description of the waterfall process, which was also described above in Munassar & Govardhan (2010), Pressman (2010) notes that the original waterfall model proposed by Winston Royce in 1970 allowed feedback loops between the steps; however, most organizations that use this model do so in a purely linear fashion. The waterfall model is well suited to times when requirements are well understood. Pressman (2010) states “This situation is sometimes encountered when well-defined adaptations or enhancements to an existing system must be made… It also may occur in a limited number of new development efforts, but only when requirements are well defined and reasonably stable” (p. 39).

The incremental process model was created to address the problems associated with the way that the waterfall model is used. In the introduction to incremental process models, Pressman (2010) notes "There are many situations in which initial software requirements are reasonably well defined, but the overall scope of the development effort precludes a purely linear process" (p. 41). In other words, the requirements may be well understood, but the process is not clear how to implement the solution. Additionally, users may need some limited functionality quickly, which allows developers to deploy early versions, then to refine and expand upon that functionality in subsequent software releases. In incremental models, the initial product, or increment, is the core model that contains most of the basic functionality that the user needs. The user is able to use and evaluate this product and provide feedback. Subsequent increments provide additional known functionality based on the original plan and unknown functionality that

gets discovered as a result of user feedback. So, in effect, the linear process of design-implement-test-maintain is implemented repeatedly in a staggered fashion (Pressman 2010).

DoD recognized the benefit and practice of using iterative and incremental software methods by issuing an interim update to DoD Instruction 5000.02 in November 2013. The purpose of the interim update was to allow Milestone Decision Authorities (MDAs) to tailor program acquisition processes to make them more efficient and to help achieve program objectives (DoD, 2013). The update introduced new defense acquisition program models specifically for DoD software-intensive systems.

Figure 3: Incrementally Fielded Software Intensive Program DoD (2013)



The new models depict how software can be produced in a series of iterations. During each iteration, new capability is developed and results in a software build. A series of software builds result in a deployable capability (DoD, 2013).

The third type of model addressed in Pressman (2010) is the Evolutionary Process Model. This model acknowledges the fact that requirements change as the development project progresses and that sometimes a limited version of the software needs to be released in order to meet deadlines or milestones (Pressman, 2010). In cases where this approach is used, “core” requirements are understood; but the detailed expectations for how the system needs to operate have not been defined. There are two common variations of evolutionary models: prototyping and spiral models. In the case of prototyping, the customer will provide high-level objectives for

the product rather than detailed requirements; or, the developer may need to explore the efficiency of specific algorithms or technology maturity. The developer performs an initial "quick" design of the system and builds it as a prototype. The prototype is used to evaluate the design by demonstrating what a working software application may look like. Prototypes are also used to generate additional requirements, and\or to solicit other feedback from customers. The prototype can be used as the initial product; though sometimes the prototype is thrown away and the system is rebuilt from scratch (Pressman 2010).

Pressman (2010) describes the spiral model as "an evolutionary software process model that couples the iterative nature of prototyping with the controlled and systematic aspects of the waterfall model. The spiral model provides the potential for rapid development of increasingly more complex versions of the software" (p. 45). In the spiral model, the software product is evolved over a series of releases that often starts with a prototype. Thus, using the spiral model allows developers to incrementally build requirements in the same vein as incrementally building the software (Pressman, 2010).

The Agile Manifesto.

In 2001, a group of seventeen software professionals (including Highsmith & Cockburn) came together and called themselves "The Agile Alliance". This group of professionals recognized the need for an alternative to the process-heavy software development models that resulted in significant software documentation but did not focus on the features of the software itself. The Agile Alliance created, agreed to, and signed off on the Manifesto for Agile Software Development, commonly called the Agile Manifesto (The Agile Alliance, 2001). The manifesto has four values and twelve principles. The manifesto acknowledges that there is value in all of

the following elements of a software development process, but that some are valued over others. The Agile Alliance (2001) documents the four values as:

> "Individuals and interactions over processes and tools
>
> Working software over comprehensive documentation
>
> Customer collaboration over contract negotiation
>
> Responding to change over following a plan" (The Agile Alliance, 2001, p.1).

The principles behind the manifesto state:

> *We follow these principles:*

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4. Business people and developers must work together daily throughout the project.
5. Build projects around motivated individuals. Give them the environment and support they need and trust them to get the job done.
6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
7. Working software is the primary measure of progress.
8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
9. Continuous attention to technical excellence and good design enhances agility.
10. Simplicity--the art of maximizing the amount of work not done--is essential.
11. The best architectures, requirements, and designs emerge from self-organizing teams.

12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly. (The Agile Alliance, 2001, p. 1)

Agile software development.

The concept of Agile software development is more of a philosophy than an actual process or method. In 2001, Jim Highsmith and Alistair Cockburn published an article in the IEEE journal discussing the motivation for the Agile movement. Highsmith & Cockburn (2001) says that "Agility, ultimately, is about creating and responding to change" (p. 122). Researchers found that projects should focus on satisfying expectations that customers' have at the time the software is delivered rather than focusing on the expectations they had at the time of project initiation because inevitably requirements change, project scope changes, and technology changes. Because the cost of change increases as a project progresses through the software lifecycle, "the question today is not how to stop change early in a project but how to better handle inevitable changes throughout its life cycle" (Highsmith & Cockburn, 2001, p. 120). Failure to respond to change equates to being unresponsive, which can mean failure. "The market demands and expects innovative, high- quality software that meets its needs— and soon" (Highsmith & Cockburn, 2001, p. 121).

According to DSB (2018), the difference between Agile approaches and traditional iterative software development is "the velocity and granularity of the iterations" (p. 6). In other words, the general construct of the iterative process depicted in Figure 2 is consistent with the general construct of Agile approaches, except the Agile approach uses shorter (faster) iterations and less functionality in each iteration. One of the core elements of Agile software development is "Focusing on small, frequent capability releases" (Modigliani & Chang, 2014, p. 2). In

traditional iterative methods used in DoD, most iterations result in an internal release that the developers test internally, whereas the Agile methods establish iterations that each result in a product intended to be delivered to the customer for evaluation and feedback. User feedback, changes in priorities, and lessons learned are inputs that shape the subsequent iterations. Though Agile methods were originally established to work well for small teams, researchers have studied methods to scale Agile frameworks to larger and more distributed projects.

Agile software development methods.

According to Rigby, Sutherland, & Takeuchi (2016), “There are at least a dozen agile innovation methodologies, which share values and principles but differ in their emphases” (p. 5). Modigliani & Chang (2014) note that while there are many Agile methods, each with their own terminology and techniques, SCRUM is the most widely used. SCRUM is an Agile method that defines roles and tasks, structures interactions through frequent meetings, and incorporates regular customer feedback into the iterative process (Oliver 2018). The SCRUM methodology has a series of iterations called “sprints”, each of which is a pass through the basic steps of the waterfall cycle. The basic steps of the classic cycle are repeated over and over again, frequently delivering working software to the customer (Oliver 2018). According to Rigby, Sutherland, & Takeuchi (2016), the SCRUM guiding principle is to “Empower creative, cross-functional teams” (p. 5).

DoD Push Toward Agile Software Development

For the last decade, DoD has been investigating the potential for using Agile software methods in the federal sector. In 2009, DoD tasked the Software Engineering Institue (SEI) to

assess Agile software development practices in government acquisition programs. One of the primary questions to be answered in the study was to determine whether or not the use of Agile software development methods would be beneficial to DoD. In 2015, the SEI updated the 2009 findings based on actual program experience and recent developments in the Agile community (Miller & Ward, 2016). In 2012, GAO conducted a study to identify the challenges associated with using Agile methods in the federal sector. GAO studied five software projects from different departments in the federal sector that have used an Agile method. The results of the study, documented in GAO (2012), included recommendations for effectively applying Agile methods in the federal sector.

In January 2017, the 115th Congress enacted the National Defense Authorization Act for Fiscal Year 2018, which included provisions related to software acquisition in DoD. In the National Defense Authorization Act for Fiscal Year 2018, the U.S. Congress provided direction to the Secretary of Defense guiding the acquisition of weapon systems software. Some of the guidance very closely mirrors some of the recommendations cited in reports discussed in this study. Congress (NDAA, 2018) first directed the Secretary of Defense to task the Defense Innovation Board (DIB) to conduct a study "with a view toward streamlining and improving the efficiency and effectiveness of software acquisition in order to maintain defense technology advantage" (p. 216). Congress further directed the Secretary of Defense to establish a pilot program to use Agile or iterative software methods on software-intensive weapon systems. In this pilot, DoD must select one high-risk software-intensive weapon system that has experienced significant cost and\or schedule growth from each service to use Agile and\or iterative software methods. Specifically, each of these pilot programs must modify their acquisition strategy to

align with Agile or iterative methods, include more frequent engagement with the user community, produce meaningful capability in an early increment, and provide training for the workforce on Agile software methods (NDAA, 2018).

Benefits of Agile Software Development

There are many theoretical benefits for the use of Agile software methods in DoD. There is also a sampling of DOD programs that have used Agile methods with some success. DSB (2018) list four studies of programs that compares the use of Agile methods to waterfall methods, some of which saw an increase in efficiency or quality. The Defense Innovation Board (DIB) reviewed some programs that used Agile methods and found a cost reduction when compared to programs using waterfall (DIB, 2019). Though they noted that they saw some success and improvement, these are very small samples with limited explanation as to how much of the software method was purely Agile and how much of the process was represented by a hybrid approach which combines traditional iterative or waterfall methods of software development with the Agile methods in order to achieve shorter iterations and more frequent deliveries.

According to Miller & Ward (2016), when using an Agile approach, a program would not require the contractor to produce a comprehensive set of detailed specifications that define the planned implementation. Rather, the government would participate in the development process via a user representative, who would be an established member of the development team. The development team would collaboratively prioritize capabilities, establish an initial architecture and skeleton of the system up front, and then allow the architecture, detailed specification, and

implementation to evolve over time (Miller & Ward, 2016). The benefit of this approach is the early user involvement and the early start of software implementation.

The Agile process depends on user involvement starting at the project initiation and continuing throughout development. According to Miller & Ward (2016), “The Agile process produces a testable system at the end of each iteration that the users can try out” (p. 33), which is in direct contrast to traditional software development methods. Traditionally, the government relies on detailed documentation produced early in the project lifecycle to assess contractor progress; and users do not have an opportunity to exercise the software until very late in the process after the product is complete. Using an Agile or iterative process, the contractor would develop and deliver prioritized software capabilities during short iterations; and a by-product of this process is early insight into the software effort. Because traditional methods would not result in delivery of a product until late in the software lifecycle, government testing and user evaluation of actual software would be deferred until that time.

Project changes can come about as a result of lessons learned in previous iterations, user feedback, change in priorities, or for other reasons. According to Miller & Ward (2016), “The Agile process accepts changes at the start of each iteration” (p. 33). Quoting Highsmith & Cockburn (2001), “Agility, ultimately, is about creating and responding to change” (p. 122); and agility allows the developers to be responsive to user needs and to shifting priorities. Agility also, as Miller & Ward (2016) note, “keeps the Agile development team focused on what is most important to the users” (p. 33). An example of this agility was documented in GAO (2010) when discussing the use of Agile software development in the Department of Veteran Affairs. “The

department noted that its Agile process allowed the flexibility to adapt to legislative changes and provide functionality according to business priorities" (GAO, 2010, p.3).

In 2016, Harvard Business Review published an article that suggests that Agile methods can improve software success rates, quality, speed to market, and software developers' motivation (Rigby, Sutherland, & Takeuchi, 2016). According to (Rigby, Sutherland, & Takeuchi, 2016), there are several well-studied and documented benefits of using Agile methods, including increased team productivity and satisfaction, decreased waste, decreased documentation, and continual adaption to changing priorities. Agile methods rely on engaged cross-functional teams working together to synergistically use the broad experience of team members from various disciplines and reduced micromanagement from senior managers. These benefits facilitate innovation and creativity when solving new, complex problems in a dynamic environment.

A primary strategy of Agile methods is to reduce the cost of change while maintaining product quality, and the methods rely on people working together while stressing working code (Highsmith & Cockburn, 2001). Extreme Programming (XP) is an example of an Agile method; and the tenets of XP include delivering the first product in weeks, using simple solutions, continually improve the design, and constantly test (Highsmith & Cockburn, 2001). Agile methods are sometimes thought to be undisciplined because "the design is done on an ongoing basis, in smaller chunks, as opposed to all at once and up front" (Highsmith & Cockburn, 2001, p. 120). The Agile philosophy acknowledges the utility of tools, processes, plans, documentation, and contracts; however, when it is necessary to decide whether to maintain the process or focus

on the delivered product, the Agile philosophy is clear that emphasis is on the product instead of process (Highsmith & Cockburn, 2001).

Chapter 4 - Analysis & Findings

Summary of Challenges

There is no single cause for the software acquisition challenges experienced by DoD when acquiring weapon systems software. A review of reports documenting these challenges reveals at least thirteen challenges that DoD has experienced between 1975 and 2018. Analysis of the reports examined shows trends that have spanned decades. Multiple reports also note that most of the recommendations have not been implemented, and none of the reports indicate improvement in any of the challenge areas from one study to the next. The 2019 DIB study directed by Congress is following up on this research and the board is actively working with the Office of the Secretary of Defense (OSD). A summary of the challenges identified most often in the reports examined in this study is depicted below in Table 1 and detailed in subsequent paragraphs.

	Challenge	Description
1.	Software Methods	Disciplined software methods using best practices that have been recognized over the years are not in use or not enforced.
2.	Personnel Staffing and Expertise	There is a department-wide shortage of DoD personnel with software acquisition and software development expertise.
3.	User Involvement	Software acquisition programs do not incorporate user involvement early and often. End users typically do not interact with the software or evaluate it until very late in the program.
4.	Requirements Definition	Requirements are not defined sufficiently to ensure developers build software capabilities that users really want and there is little opportunity to iterate over requirements definition process.
5.	Inaccurate Cost Estimation	Cost overruns are often attributed to inaccurate cost estimates at the beginning of the program.
6.	Unrealistic Expectations	Stakeholders often expect more than can be achieved with existing technology or within existing schedule and budget constraints. Stakeholders are also often unwilling to accept partial solutions and evolve capability over time.
7.	Communication	There is often a lack of structured and frequent communication between stakeholders.
8.	Program Management & Emphasis	Software often does not receive the same emphasis as hardware, and program managers and their staff are often not trained to manage software-intensive programs. As a result, software does not get adequate attention.
9.	Systems Engineering	Sound systems engineering is used to analyze user stated capability requirements and define technical solutions. Sound systems engineering discipline is always used on software-intensive programs.
10.	Requirements Stability	Requirements changes due to lessons learned, technology changes, evolving threat, etc. can cause requirements instability in a process that is not designed to accommodate change.
11.	Architecture	Architecture is often poorly defined. A well-defined architecture is essential to reduce dependencies that cause errors, to remain flexible and responsive to change, and to facilitate reuse.
12.	Visibility & Measuring Progress	Program managers often do not have proper insight into the health and status of software-intensive programs. Metrics are often not useful or do not convey the actual status of the effort.
13.	Long Development Cycles	It often takes years between the time the initial system concept is defined and the time that the capability reaches the user. In the time that passes, the system can become obsolete, the threat can evolve, or the developer may not have properly translated the user's requirements.

Table 1: Summary of DoD Software Acquisition Challenges

Systems Engineering and Software Methods

Problems attributed to a lack of sound systems engineering and software engineering discipline have been documented since the 1970s. The software development community discovered through lessons learned during the 1970s that the success of a software development effort depends on how well the software is designed, built, and tested (GAO, 1978). A second related issue became obvious, during the 1980s. It became clear that the traditional waterfall software method was not effective when building new, innovative systems; and that commercial best practices would be more effective for large projects that push the brink of technology. Even as late as 2017, studies show that the best software engineering practices are not always enforced in DoD, which shows that the problem of maintaining the use of best practice has persisted. It is not enough to document and plan to use sound discipline. Discipline must be enforced rather than sacrificing rigor to meet potentially unrealistic schedules.

Personnel Shortages

Additionally, personnel shortages that have been documented since the 1980s have persisted. DoD does not have a core competency for software acquisition, so it is difficult to manage talent for this competence. For example, according to Clements (2017), the Army Director for Acquisition Career Management (DACM) Office is assessing the health of workforce skills, yet none of the career fields or job established by the DoD have a specific focus on software. Software positions are embedded in the Engineering and Information Technology career fields; however, software does not get sufficient emphasis to attract new talent or to build competency.

Inaccurate Cost Estimates

Issues associated with inaccurate cost estimates have been documented since the 1990s. One contributing factor to the inaccuracy of cost estimates is the timing of development of system specification documents, which are the documents that describe what systems must be able to do. System specification documents are completed, and contracts are awarded before the details of software requirements are understood. This practice significantly contributes to the inaccuracy of schedule estimates as well as cost estimates. Program managers must maintain a balance between developing a system that has all of the capability required by the user, completing the system development on time, and achieving these results within budget. This balance is often constrained by inaccurate estimates because users would like to have all of the features built in the software, and they want the software to be delivered on time. If an effort is underestimated, meaning that it will take longer or cost more than anticipated to build the system, there is often a resistance from users and senior leaders to accept a less capable system or to delay the delivery of the system. Constrained schedules and budgets and stakeholder expectations of full system capability are difficult to overcome once the baseline expectations are set.

Unrealistic Expectations

Another challenge that arises from establishing estimates prior to fully understanding requirements is unrealistic stakeholder expectations. Users and other key stakeholders may expect a system to do more than is realistically feasible (due to technology constraints or other technical or programmatic constraints), or stakeholders may believe that systems can be delivered faster than is actually realistic. Software development is extremely difficult and organizations often fail to understand the complexity of developing a new and innovative system.

According to Heil (2017), even when estimates are accurate, senior leaders often drive their organization to "accept the challenge" to deliver originally planned capability for reduced cost or to deliver the capability on an accelerated schedule. Unrealistic expectations are often caused by a stakeholders do not fully understand the scope of the system that needs to be developed.

User Involvement

Insufficient end user involvement and inadequate requirement definition are two related challenges that have been documented many times. In traditional DoD development, users create documents that describe the system that they want and the capabilities that the system needs to provide. Typically, it takes years from the time the users define the initial concept of the system until the time the system is actually built; and users do not have the opportunity to evaluate the system and how it works before the final system is delivered. During the timeframe from concept definition through system development, the users' needs or priorities may evolve; and the Acquisition system does not provide a mechanism to easily insert these changes into the project. Additionally, the users' intent may be unclear or misinterpreted which could cause the developer to build a system that does not meet the users' needs. Because there is limited involvement with the end users, there is typically no reach-back from the developers to the warfighting element that will fight with the weapon system, or feedback from end users to the developers, to ensure that software level specifications are correct. As stated in DSB (1987), software requirements development is the most important step in the software development process because it determines what product will be built. It is also the most difficult part of the process because systems are too complex to just imagine. It is for this reason that early and frequent user involvement is crucial.

Management Emphasis and Visibility

Another prevalent challenge for software development in DoD is the way weapon systems software projects are managed and the lack of emphasis on the software effort. In many cases, management emphasis is on the hardware portions of the system. PMs do not enforce software engineering best practices, nor do they maintain visibility into the software effort. When PMs do not focus on the software effort or have the visibility necessary to understand the status of the effort, problems are discovered late in the process. Late discovery of problems can render the situation irreparable. This problem is related to the lack of software expertise that has also been documented many times. Program managers often do not have software expertise themselves, and they do not have sufficient staff with software expertise. This lack of available expertise makes managing the effort difficult.

Long Development Cycles

Another set of interrelated challenges is the combination of long software development cycles and continually changing requirements. Technology changes very rapidly and specific threats to U.S. warfighters and to America evolve over time. This evolution and change in technology often drive the users to change what the system needs to do. The long development cycles called out in DSB (1987) refer to the year or more timeframe that it typically takes to build one increment of software or the years it often takes to build a complete system. The long development cycles make it difficult to quickly adapt to changes or to quickly course correct problems discovered during development of the system.

Architecture and Interfaces

Starting in the 1980s, the DSB began reporting on the importance of software architecture. A carefully planned architecture is essential if DoD is to achieve software reuse, build software product lines, and develop maintainable software for its weapon systems. A good architecture is fundamental to establishing a software program that can evolve incrementally over time and respond to changes. Many reports have documented concerns such as too little program emphasis on architecture (DSB, 1994), software problems that result from dependencies established in the underlying architecture (DSB 2018), and poorly defined and controlled interfaces (DSB, 1994).

What DoD Could Do to overcome challenges

Several of the sources reviewed in this study discuss the acquisition policies that make it difficult to use modern software methods on DoD acquisition programs. Air Force Software Technology Support Center (1996) discusses recommendations from earlier studies that suggest that DoD use commercial practices but cautions to only adopt the ones that mitigate the issues that impact DoD programs and to avoid any others. DoD 5000 requires formal reviews early in the program that expects the contractor to have completed all software requirements and design. DoD could tailor or make additional modifications to DoD 5000 that anticipate and encourage an iterative software development strategy. Additionally, DoD could modify cost estimation and reporting requirements to better align with an iterative and incremental strategy.

DoD could also implement strategies to attract, train, and retain personnel with software expertise. Issuing recruitment actions that specifically require software expertise instead of general engineering expertise could attract software knowledgeable people to consider working

for DoD. Currently, and throughout history, DoD training classes have very little emphasis on software. DoD could create a software training curriculum and require that acquisition professionals involved in acquiring weapon systems software achieve a certain level of competency. Building a core competency in software for DoD would address the challenge of skilled software personnel shortages.

Measuring Progress

A closely related action that DoD could take is to invest in a strategy to create meaningful project metrics. Lack of visibility and lack of insight into software efforts can be addressed in part by measuring progress in a meaningful way instead of attempting to use the status of documentation to indicate the status of the software effort. DoD needs skilled personnel to determine how to measure progress and create adequate metrics that will provide the necessary software insight. Additionally, program managers could be trained in order to better understand how to interpret metrics and what they mean for the program.

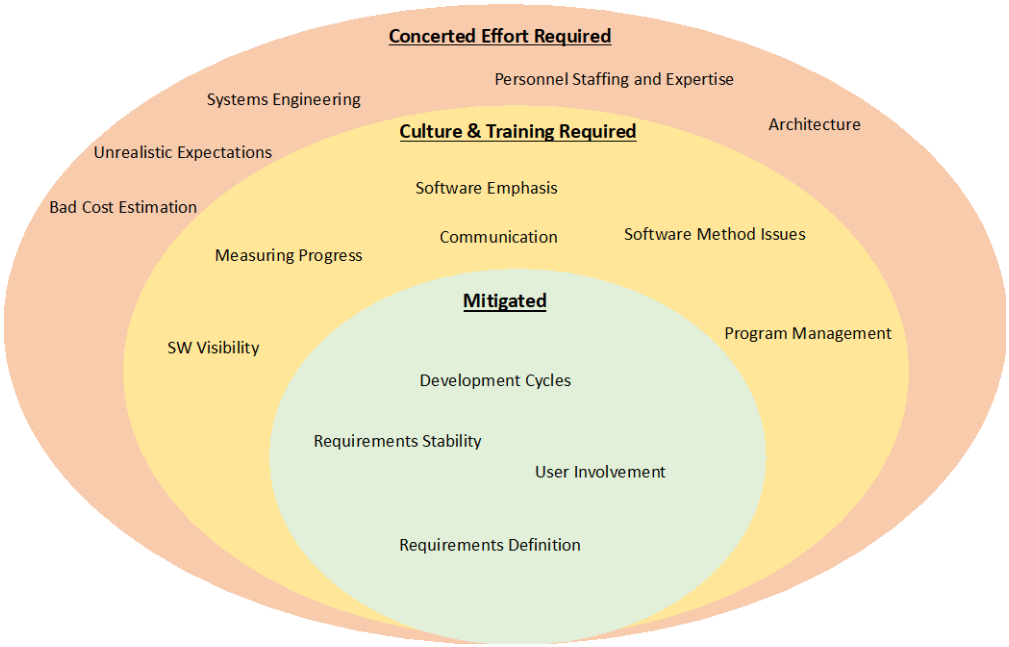
Agile Contributions to Mitigating Challenges in DoD Software Acquisition

The general philosophy behind Agile software development supports some of the recommendations that can be drawn from the reports reviewed in this study. Figure 4 depicts the challenges discussed in this study. The challenges are grouped into three categories: challenges that could potentially be mitigated by adopting and properly executing Agile methods (in green), challenges that could be mitigated if the requisite amount of training and culture changes take place (in amber), and challenges that require additional concerted effort in order to improve acquisition of software-intensive weapon systems (in orange). Some of the challenges associated

with ineffective practice of software methods in DoD could potentially be mitigated by using some of the practices in Agile software development methods properly. Specifically, if DoD embraces the iterative nature of the Agile philosophy, then DoD can address some of the recommendations that can be found in many DSB reports. The DSB began recommending that DoD use iterative software methods that facilitate iterative development of user requirements in the 1980s (DSB, 1987), and in every report since that time. Agile methods are both iterative and incremental, and these characteristics go a long way towards mitigating other challenges. It will remain a challenge to ensure that the proper discipline and execution is maintained. DoD could achieve this type of early delivery and feedback using traditional iterative methods by shortening software build cycles and requiring delivery of incremental capability in much the same way as is accomplished using Agile methods.

Results of the shift to Agile methods will likely be mixed until adequate training and culture shifts take place.

Figure 4: Software Challenges Mitigated by Agile Methods



One of the challenges discussed in several reports is the difficulty of specifying all software requirements at the beginning of the project and the inability of users and the human mind to fully imagine the desired software product without having a software product to test and evaluate. Agile software methods allow software development to start early and the process involves early and frequent user interaction and software evaluation. This process facilitates an interactive process for the user to develop user requirements. This type of iterative and interactive process also addresses the challenge of being able to effectively specify lower level software requirements. For weapon systems software applications that involve human interaction, early prototyping or early iterations provide users the ability to operate software and provide additional information to ensure that the right software is built. As stated in DSB (1987), systems are too complex to imagine or visualize the full scope of the requirements. Any iterative method employed correctly could mitigate this challenge whether it is Agile or not.

Gaps: Challenges That Remain Despite Agile Adoption

Agile methods have significant promise for use in DoD. However, there are basic engineering principles that are not necessarily driven by these methods. Sound systems engineering is critical for large complex systems. A well thought out, modular architecture that minimizes interdependencies between software elements must be defined in order to enable the rapid change that occurs in Agile methods. Additionally, adopting Agile methods will also not address the software personnel deficiency in DoD. In fact, this presents an additional challenge in which personnel will need to learn new ways to think of software development with new terminology and altered expectation.

Poor cost estimation is another area that is not addressed by Agile software methods. Embracing Agile requires that budget analysts accept more cost risk on the front end, and gain better cost estimation fidelity as the project goes on. Additionally, increased program management emphasis is an area that must be addressed in the new paradigm. In order to achieve agility, DoD must embrace the Agile philosophy and change the culture. As stated in (Modigliani & Chang, 2014):

> Rather than simply follow a recipe of Agile methods and steps, programs should try to adopt the Agile philosophy and mindset to instill program agility. Establishing an organizational change discipline and clear vision can help to communicate the organizations strategy of adopting the Agile philosophy. (p. 9-10)

Chapter 5 - Conclusions & Recommendations

DoD has always had significant challenges in developing and acquiring weapon systems software, and these challenges have been the subject of many studies. The challenges have been documented in many reports over the last forty years, and these reports also provided numerous recommendations to mitigate the challenges. DoD has many lessons learned and seems to have relearned those lessons time and time again. This relearning begs the question of whether or not there is a consistent understanding of the depth of the problems that have been studied, such as the need for systems engineering discipline and enforcement of software systems engineering best practices.

Today, DoD is making a concerted effort to migrate to Agile software development practices across the department. This report explores the benefits of adopting Agile software methods as well as challenges that will remain if they are not explicitly addressed. Given the remaining challenges, the following recommendations serve to mitigate these challenges:

1. Adopt Agile practices that make sense within the context of developing weapon systems software. Establish a culture of agility through training, policies, and leadership by example. Tailor Agile methods in a manner to ensure that the methods address the fact that weapon systems software comes with the risk of loss of life and must be handled with more rigor than commercial applications.
2. Do not completely abandon sound systems engineering discipline when using Agile methods. Further study is required to establish best practices for weapons systems that is based on modern best practices in industry. Ensure that basic systems engineering is understood and applied across the DoD. Maintain sound configuration management practices, traceability across software and systems engineering products, and maintain

standards to handle safety critical and mission critical software components. Ensure that frequent testing is comprehensive, including cases that stress the system and push the boundaries of expected software inputs.

3. Focus on well-established architecture within any process adopted by DoD. Ensure that large complex software-intensive system programs emphasize flexible and modular software architectures and that architecture analysis begins very early in the project.
4. Do not focus on speed to the detriment of quality and mission success. Ensure that software build intervals are realistic and commensurate with the amount of functionality developed in the iteration. Factor in the additional time that may be required to do things like test algorithms, perform value-added peer reviews, refactor software, and so forth. Establish realistic expectations and maintain program management insight into the health of programs through meaningful metrics, immersive involvement, and frequent software evaluation.
5. Allow tailorable durations for iterations. Weapon systems software is often mission critical or safety critical and may require longer sprints than the typical 2 – 4 week duration. Each iteration must be long enough (e.g. 4 – 8 weeks) to solve complex problems or factor in rigor.
6. Make a concerted effort to build software competency in DoD. Establish a career track for software personnel that serves to attract and retain personnel with software skills. Mandate a combination of education, training, developmental assignments, and training with industry that together increase the understanding of software systems engineering

and software development within DoD. Program managers, technical staff, and military should all build this competency.

The problem statement in the introduction of this paper included three questions that this report would answer. The first question seeks to identify the challenges that DoD has sustained when acquiring weapon systems software. Chapter 4 summarizes reports from over 40 years of software studies in DoD and recaps most of the challenges discussed in the literature review in Chapter 3. The second question seeks to identify actions that DoD can take in order to mitigate the challenges. The recommendations documented in the paragraph above, as well as other recommendations in Chapter 4 provide potential actions to address the challenges. Each of these recommendations are achievable if implementation plans are developed and executed and if DoD remains committed to the transition across the department. Finally, the third question seeks to identify ways in which Agile methods can help DoD overcome software acquisition challenges. Recommendations in this report indicate that DoD can benefit from adopting the Agile philosophy. However, there are other initiatives required to fully mitigate the challenges in DoD software acquisition.

REFERENCES

Air Force Software Technology Support Center. (1996). *Guidelines for Successful Acquisition and Management of Software-Intensive Systems.* Hill Air Force Base, Utah: Department of the Air Force.

Air Force Studies Board. (1989). *Adapting Software Development Policies to Modern Technology.* Washington D.C.: National Academy Press.

Boehm, B., & Turner, R. (2004). *Balancing agility and discipline: a guide for the perplexed.* MA: Pearson Education.

Clements, T. (2017). ARMY DACM OFFICE ASSESSES WORKFORCE STRENGTHS; WEAKNESSES. *AT&L Magazine.*

Defense Science Board. (1987). *Report of the Defense Science Board Task Force on Military Software.* Washington DC: Office of Secretary of Defense.

Defense Science Board. (1994). *Defense Science Board Task Force on Acquiring Defense Software Commercially.* Washington D.C.: Office of the Under Secretary of Defense for Acquisition & Technology.

DoD. (2013). *Interim DoD Instruction 5000.02.*

GAO. (1978). *Managing Weapon System Software: Progress and Problems.* Washington D.C.: United States General Accounting Office.

GAO. (2010). *Information Technology Veterans Affairs Can Further Improve Its Development Process for Its New Education Benefits System.* Washington DC: Government Accountability Office.

Gibbs, W. (1994). Software's Chronic Crisis. *Scientific American Journal*, 86.

Heil, J. (2017). Keys to Successful DoD Software Project Execution. *Cyber Security & Information Systems Information Analysis Center (CSIAC) Journal*, 5 - 13.

Highsmith, J., & Cockburn, A. (2001). Agile Software Development: The Business of Innovation. *IEEE*, 120-122.

Jacobson, I., Booch, G., & Rumbaugh. (1999). *The Unified Software Development Process.* Reading, MA: Addison-Wesley.

Kossiakoff, A. (1975). *DoD Weapon Systems Software Management Study.* Laurel, MD: Johns Hopkins University.

Miller, S., & Ward, D. (2016). *Update 2016: Considerations for Using Agile in DoD Acquisition.* Pittsburg: Carnegie Mellon University.

Munassar, N. M., & Govardhan, A. (2010). A Comparison Between Five Models Of Software Engineering. *International Journal of Computer Science Issues, Volume 7, Issue 5*, 94 - 101.

Oliver, P. (2018). *Agile Software Development Management Success Series.* Concise Reads.

Pressman, R. (2010). *Software Engineering A Practitioner's Approach.* New York, NY: McGraw-Hill Companies.

Rigby, D. K., Sutherland, J., & Takeuchi, H. (2016). *Embracing Agile.* Harvard Business Review.

Secretary of the Air Force for Acquisition. (2008). *Weapon Systems Software Management Guidebook.* U.S. Air Force.

Smith, H. (2018). *Scrum: The Ultimate Beginners Guide to Learn And Master Scrum Agile Framework.* Hein Smith.

The Agile Alliance. (2001, February). *Principles behind the Agile Manifesto*. Retrieved from Agile Manifesto: https://agilemanifesto.org/principles.html

The Agile Alliance. (2001, February). The Agile Manifesto. Retrieved from Agile Manifesto: https://agilemanifesto.org/history.html

TRADOC Analysis Center. (2015). *A Descriptive Guide to Conducting Trade Space Analysis.* Fort Leavenworth, Kansas.

VII, P. (2016). *Agile Product Management.* Pashun Publishing Press.

Appendix A Acronyms

DSB	Defense Science Board
GAO	Government Accountability Office
IEEE	Institute of Electrical and Electronics Engineers
IPA	Intergovernmental Personnel Act
MDA	Milestone Decision Authority
MIL-STD	Military Standard
NDS	National Defense Strategy
OSD	Office of Secretary of Defense
PM	Program Manager
SEI	Software Engineering Institute
XP	Extreme Programming

Disclaimer

The views and opinions expressed or implied in this research paper are those of the author; no agency or department of the United States Government has officially sanctioned any of these view and opinions.