Tailorability-Focused Recommendations for United States Air Force Software Acquisition Policy

by

Dylan Toshinari Muramoto

Submitted to the Department of Aeronautics and Astronautics and the Institute for Data, Systems, and Society on May 1, 2020 in Partial Fulfillment of the Requirements for the Degrees of

Master of Science in Aeronautics and Astronautics
Master of Science in Technology and Policy

Abstract
In order to adapt and respond to threats by near-peer-adversaries that are changing at an increasing pace, the U.S. Department of Defense (DoD) has been focused on reforming software acquisition for rapid development and deployment of software capabilities to the battlefield. Military leaders have been focused on accelerating development and increasing the frequency of deployment, encouraging developers to take risks to reduce schedules. However, military systems have certain levels of performance and quality requirements that must be met to successfully execute missions. Additionally, software systems have many different characteristics that must be considered during development. In this thesis, the DoD software acquisition process and new guidance from the Department and the U.S. Air Force (USAF) regarding software acquisition reforms are detailed first. The existing policy is examined to identify gaps regarding tailoring development processes to different software systems. After providing context on how software is developed and describing four process models to show that different processes are most appropriate for developing systems with certain characteristics, eight specific software system characteristics are identified: user, urgency, lifespan, performance (timing), quality/risk, size, integration, and requirements. Furthermore, recommendations to the USAF and DoD for implementing policy/guidelines that help establish a tailorable software acquisition process based on these eight system characteristics are provided. This thesis hopes to help leaders and managers understand the technical characteristics of software systems and match those with appropriate development process designs and practices, instead of a one-size-fits-all blanket solution, so that the required quality and evolvability of military systems are not compromised in execution of the national security mission.

Thesis Supervisor: Nancy G. Leveson
Title: Professor of Aeronautics and Astronautics
Acknowledgements

I want to thank Dr. Nancy Leveson for teaching and showing me the importance of good systems engineering. I hope to use the principles that have been engrained in me to not only focus on cost, schedule, and performance, but on having an impact. Thank you to my research contacts Major Gregory “Lab” Bieler, Rick Delaney, and Scott Dowty for helping someone with absolutely no operational experience gain insight into software development within the Air Force. Colonel Douglas “Beaker” Wickert, thank you for your impromptu visit to the group and introducing me to Section 804, which sparked my research into this whole topic. My research and degree wouldn’t have been possible without support from Lincoln Lab Group 108 and the guidance of Sam Stambler who showed me that federal research doesn’t have to be hours of sitting in front of a computer; you can build and fly things too.

Along with those that have enabled my research, I want to thank those that have made my experience at MIT unforgettable. To my TPP cohort, the memories that we shared together, along with the endless debates about notions and frameworks, will forever be treasured. Each and every moment, from grinding on PSets in the lounge until midnight to sitting around the campfire at IYR, was special in its own way. I’ll try not to embarrass myself too much or make chair speeches that are too long during our reunions. To my lab and class mates in the AeroAstro department, especially in the System Safety group within the Engineering Systems Laboratory, thank you for showing me that research can indeed change the world. It’s a shame that we can’t walk across the stage together during Commencement, but I’m sure the friendships we’ve built will outlast COVID-19 and remain strong as we make our own impacts in this world.

In 2017, before even submitting my application to MIT, I was told by some powerful people that I would not be allowed to attend even if I was accepted. I was discouraged and tempted to abandon my efforts. However, some people told me that I had nothing to lose if I submitted my application anyways. After being accepted into both AeroAstro and TPP, and with the support of Lincoln Lab, I was finally approved to attend MIT by the same people that told me I wouldn’t be allowed to a few months ago. This experience isn’t encouragement to defy authority; I look at it as an example that if you work hard enough, and rely on support from those that believe in you, anything is possible. As an engineer that believes in careful planning and analysis, my mantra of: “you never know what’s going to happen, so don’t have expectations” is contradictory. Never underestimate the support from those around you. I would not be in this position writing the final paragraph of my thesis without mentors and contacts who believed in me. Finally, I want to thank those closest to me: my mom, my dad, Calvin, and Lani for continuing to believe in me more than I believed in myself. Special recognition to my mom who spent decades of hard work encouraging me to be the best person I can be. The sweat and tears were worth it. I love you all.
Table of Contents

ABSTRACT ...................................................................................................................................................... 3  
ACKNOWLEDGEMENTS .................................................................................................................................. 4 
TABLE OF CONTENTS ..................................................................................................................................... 5  
LIST OF FIGURES ........................................................................................................................................ 6  
LIST OF TABLES ................................................................................................................................ .......... 7  
LIST OF ACRONYMS AND ABBREVIATIONS .............................................................................................. 8 
1. INTRODUCTION ......................................................................................................................................... 9  
  1.1 WHAT IS ACQUISITION? .......................................................................................................................... 9  
  1.2 WHY CHANGE ACQUISITION? ............................................................................................................... 9  
  1.3 RESEARCH SCOPE .................................................................................................................................. 11  
  1.4 RESEARCH QUESTIONS ............................................................................................................................ 12  
  1.5 RESEARCH METHODOLOGY .................................................................................................................... 12  
2. BACKGROUND AND STATE OF THE ART ............................................................................................. 13 
  2.1 DEFENSE ACQUISITION .......................................................................................................................... 13  
      2.1.1 Defense Software Acquisition .......................................................................................................... 16  
      2.1.2 Air Force Software Acquisition ......................................................................................................... 18  
      2.1.3 Tailoring ........................................................................................................................................... 24  
  2.2 SOFTWARE DEVELOPMENT PROCESSES .......................................................................................... 28  
      2.2.1 Waterfall Methodology ....................................................................................................................... 30  
      2.2.2 Incremental Model ................................................................................................................................ 32  
      2.2.3 Spiral Model ........................................................................................................................................ 34  
      2.2.4 Agile Methodology ............................................................................................................................... 36  
3. TAILORING DEVELOPMENT PROCESSES ......................................................................................... 39  
  3.1 USER ...................................................................................................................................................... 41  
  3.2 URGENCY .............................................................................................................................................. 42  
  3.3 LIFESPAN ............................................................................................................................................... 43  
  3.4 PERFORMANCE (TIMING) ......................................................................................................................... 44  
  3.5 QUALITY/RISK ....................................................................................................................................... 45  
  3.6 SIZE ....................................................................................................................................................... 46  
  3.7 INTEGRATION ....................................................................................................................................... 47  
  3.8 REQUIREMENTS .................................................................................................................................... 48  
4. POLICY RECOMMENDATIONS ............................................................................................................. 50  
  4.1 STRENGTHS ....................................................................................................................................... 51  
  4.2 WEAKNESSES ..................................................................................................................................... 52  
  4.3 OPPORTUNITIES .................................................................................................................................. 52  
  4.4 THREATS ............................................................................................................................................... 52  
5. CONCLUSION ............................................................................................................................................. 54  
  7.1 LIMITATION OF RESEARCH .................................................................................................................. 56  
  7.2 FUTURE WORK .................................................................................................................................... 56  
6. REFERENCES ............................................................................................................................................. 57
List of Figures

Figure 1. Generic Acquisition Phases and Decision Phases [2] ................................................... 14
Figure 2. Defense Unique Software Intensive Program [2].......................................................... 16
Figure 3. Incrementally Deployed Software Intensive Program [2]............................................. 17
Figure 4. DoD Adaptive Acquisition Framework [26]................................................................. 19
Figure 5. Software Acquisition Pathway [26]............................................................................. 20
Figure 6. Lifecycle View of Software Acquisition Pathway [30] ................................................ 23
Figure 7. Middle Tier of Acquisition (Rapid Prototyping) Pathway then transitioning to Software Acquisition [33] ............................................................................................................................ 25
Figure 8. Major Capability Acquisition then transitioning to Software Acquisition [33]......... 26
Figure 9. Major Capability Acquisition with Software Acquisition occurring concurrently [33] 26
Figure 10. Six Processes within the Software Development Life Cycle ..................................... 28
Figure 11. Waterfall Software Development Process................................................................. 30
Figure 12. Incremental Software Development Process............................................................ 32
Figure 13. Spiral Model Software Development Process [42] .................................................... 34
Figure 14. Agile Software Development Process ....................................................................... 36
Figure 15. Growth of Source Lines of Code (SLOC) in Avionics Software [13] ...................... 46
List of Tables

Table 1. Characteristics of Software Systems............................................................................... 49
List of Acronyms and Abbreviations

A&S   Acquisition and Sustainment
ACAT  Acquisition Category
AoA   Analysis of Alternatives
API   Application Programming Interface
ASD   Acquisition Strategy Document
AT&L  Acquisition, Technology & Logistics
ATO   Authority to Operate
CDD   Capability Development Document
DAs   Decision Authorities
Dev   Development
DevOps Development and Operations
DIB   Defense Innovation Board
DoD   Department of Defense
DT&E  Developmental Test and Evaluation
FOC   Full Operational Capability
GAO   Government Accountability Office
IOC   Initial Operational Capability
JCIDS Joint Capabilities Integration and Development System
MA    Mission Assurance
MDA   Milestone Decision Authority
MDAP  Major Defense Acquisition Programs
MTA   Middle Tier Acquisition
MVCR  Minimum Viable Capability Release
MVP   Minimum Viable Product
NASA  National Aeronautics and Space Association
NDAA  National Defense Authorization Act
NMS   National Military Strategy
NSS   National Security Strategy
O&S   Operations and Support
Ops   Operations
OSD   Office of the Secretary of Defense
OT&E  Operational Test and Evaluation
P&D   Production and Deployment
PEOs  Program Executive Officers
PMs   Program Managers
RFP   Request For Proposal
SDLC Software Development Life Cycle
SLOC  Source Lines of Code
SRS   Software Requirements Specification
SWAP  Software Acquisition and Practices
TMRR  Technology Maturation and Risk Reduction
UAI   Universal Armament Interface
USAF  United States Air Force
V&V   Verification and Validation
1. Introduction

1.1 What is Acquisition?
The United States Department of Defense (DoD) Acquisition Process is the bureaucratic management and procurement process used to invest in technologies, programs, and support necessary to maintain military superiority and meet the national security agenda. The military has specific needs that must be met, and the objective of the process is to acquire products that fulfill them and improve on current mission capabilities, all while balancing cost, schedule, and performance. The U.S. DoD Acquisition Process involves a complex dynamic between technology and policy. The science, engineering, and management used at every phase of the technology lifecycle is always set within the context of the nation’s laws and regulations. As technologies become increasingly complex and the implications of use increasingly unclear, the policies determining their procurement enter a gray area where decision makers may not fully understand how to effectively develop new systems.

The acquisition process includes the design, engineering, construction, testing, deployment, sustainment, and disposal of military technologies. It is event based and split into multiple phases, milestones, and reviews. In order to operate such a large and complex process, there are certain policies and guidelines put in place. The DoD Directive 5000.01 “The Defense Acquisition System” [1] is an overview that describes the high-level policy for the acquisition system, and DoD Instruction 5000.02 “Operation of the Defense Acquisition System” [2] details how the processes of the system are implemented.

Since the establishment of the process, the technologies being acquired have evolved significantly. Over the past couple decades, software has grown to play an increasingly critical role in new systems. Software-driven systems have become more prevalent as hardware driven systems of the past are becoming obsolete. However, software acquisition processes within the DoD have not updated at the same pace of technology. Meanwhile, the U.S.’s near-peer adversaries – Russia and China – have invested heavily in technology and are developing capabilities at an increasing rate [3]. Former Secretary of the Air Force Heather Wilson called the current DoD process archaic and in need of modernization if we expect to win in a peer-to-peer conflict [4]. The USAF Weapons Systems Software Management Guidebook [5], which provides guidance for organizations to have successful software acquisitions, is over ten years old and references practices from the Military Standard 498 – Software Development and Documentation [6], an even older document published in 1994 that was not expected to be used for more than two years.

1.2 Why Change Acquisition?
There is now a push from senior leadership in the DoD to innovate and increase capabilities at the “speed of relevance” [7] by focusing on improving and accelerating the acquisition process. According to former Secretary of Defense Ash Carter: “today’s era of technological competition is characterized by the additional variables of speed and agility, such that leading the race now depends on who can out-innovate faster than anyone else. It’s no longer just a matter of what we buy. Now more than ever, what also matters is how we buy things, how quickly we buy them, whom we buy them from, and how quickly and creatively we can adapt and use them in different and innovative ways – all this to stay ahead of future threats” [8]. By decreasing barriers to
innovation and removing the red tape of bureaucracy, military leaders want to promote innovation and keep pace with the technological advances of peer adversaries to be prepared for potential conflicts.

Policies, guidance, and standards are being revised as a response to this focus on acceleration. The 2016 National Defense Authorization Act (NDAA) is a Congressional bill that specifies the budget and expenditures of DoD military activities for fiscal year 2016. Within it, the Section 804 Authority [9] introduced the Middle Tier Acquisition (MTA) rapid acquisition interim approach that focuses on delivering technological capabilities within 2-5 years. The overall goal of the Authority is to streamline the testing and development of prototypes or upgrading of existing systems with proven technology. The focus on speed and innovation has shifted a large amount of attention to the software acquisition process specifically.

The Defense Innovation Board (DIB), a Federal Advisory Committee composed of experts in academia and the commercial sector who aim to provide the Secretary of Defense and senior leaders across the Department with independent advice to catalyze innovation within the organization, was congressionally mandated in the 2018 NDAA Section 872 to conduct a study on “streamlining software development and acquisition regulations” [10]. The board finalized the Software Acquisition and Practices (SWAP) study in May 2019 and it includes ten recommendations structured around three themes and broken into four main lines of effort [11]:

Main themes:
1. Speed and cycle time are the most important metrics for managing software
2. Software is made by people and for people, so digital talent matter
3. Software is different than hardware (and not all software is the same)

Lines of effort:
A. (For Congress and Office of the Secretary of Defense (OSD)): Refactor statutes, regulations, and processes for software
B. (For OSD and Services): Create and maintain cross-program/cross-Service digital infrastructure
C. (For Services and OSD): Create new paths for digital talent (especially internal talent)
D. (For DoD and Industry): Change the practice of how software is procured and developed.

Parts of the Department are now starting to spark the change they wish to see. The Office of the Under Secretary of Defense for Acquisition and Sustainment (A&S) has rewritten the DoD Instruction 5000.02 this year by starting with a clean slate and introducing simple compliance guidelines. A&S’s goal was to “develop flow charts that take into consideration the specific nature of the program and identify only those items that help the service get the desired capability into the hands of the warfighter expeditiously and affordably”[12].

The motivation to change and update the process is strong within the DoD, but the primary focus on speed/cycle time metrics, digital talent, and prioritizing software may not be adequate. Focusing on quickly fielding software and making changes may increase the risk of fielding incorrect software systems that impact mission performance. Compared to hardware, software changes and capability modifications can be done quickly. However, making the correct changes
continues to prove to be difficult. In a 2018 Defense Science Board report, software was assessed to be among the most frequent and most critical challenges, driving the risk for approximately 60% of all DoD programs [13]. Even if changes can be made quickly, making incorrect changes may further increase risks as additional cost and time are required to fix mistakes. Ultimately, fielding incorrect software too quickly could lead to detrimental performance consequences that could impact mission success. Focusing on good digital talent is beneficial, but may not be fully leveraged if a one-size-fits-all development process is used. Recognizing that not all software is the same is important, but not helpful if policies and guidance do not appropriately account for differences in developing specific software systems. An equally important consideration not addressed by the SWAP study is that at least two thirds of the software lifecycle consists of maintenance and upgrades. Savings spent on software development can become irrelevant if long-term maintenance and evolution costs spiral out of control. In addition, the ability to maintain and upgrade software is dependent on the design and documentation produced in the original development effort.

The motivation behind analyzing the status quo and providing recommendations for future processes is to help leaders and managers understand the technical characteristics of software systems and match those with appropriate development techniques and practices. This thesis hopes to help accelerate development time but most importantly help ensure that the software developed meets the required levels of performance for military systems.

1.3 Research Scope
In order to scope my research, I focus specifically on the USAF software acquisition process. Due to the momentum gathered by recent USAF acquisition reform efforts and the pressing need to rethink software acquisition, this scope keeps the research topic relevant and has the additional benefit of potentially inciting change.

Furthermore, this research focuses specifically on the development of software for USAF systems. This further constrains the research focus on examining how mission-critical software programs should be developed with mission assurance (MA) in mind. The DoD defines MA as:

“A process to protect or ensure the continued function and resilience of capabilities and assets, including personnel, equipment, facilities, networks, information and information systems, infrastructure, and supply chains, critical to the execution of DoD mission-essential functions in any operating environment or condition.” [14]

Developing correctly functioning software systems is critical for the USAF. Inadvertent missile launches resulting from software flaws exacerbated by extended operation times necessitated by unique combat environments [15], [16], for example, have consequences that could impact mission success. A 2018 report by the Government Accountability Office (GAO) on the cybersecurity of weapon systems [17] found that many of the military’s mission-critical systems could be taken control of; a two-person team took only an hour to gain access to a weapon system and gained full control in a day. Incorrect, unsafe, or insecure software could result in detrimental consequences on the field. The benefits of faster deployment would be negated if software quality and evolvability suffers as a result.
The military operates in unique environments and conditions where certain levels of risk involving damage or loss of life/property must be continuously managed. Ensuring that the software being developed and fielded to conduct missions satisfies required levels of quality and performance is important for MA. Instead of creating a blanket process for all software, this research hopes to create policy recommendations for managers and decisionmakers to choose appropriate development processes that consider the performance and quality needs of the systems being developed while accelerating capabilities to meet the speed and innovation needs of potential future conflicts where possible.

1.4 Research Questions
The USAF has been adopting significant acquisition reforms and has been specifically focusing on software development. The Section 804 Authority has allowed end-users to help shape and iterate technological capabilities because it allows the fielding of minimally viable products on a short timeline. The impact of this acceleration has been clearly seen in software systems, which can be iterated faster and fielded earlier than hardware.

While new software acquisition reform efforts have seen initial success, it is still uncertain what software development processes work and should be used for the DoD. The Silicon Valley startup mantra of “fail fast, fail often” may not be appropriate for DoD applications. Recent efforts within limited domains appear promising and good results have been highlighted, but it remains to be seen how viable it will be for many large and important defense projects. Good software engineering and system engineering principles must be combined with encouraging innovation and acceleration in a way that required quality and evolvability does not suffer. In addition, the use of one-size-fits-all blanket solutions should be avoided. Not all software is the same. Software systems have many different characteristics that should be considered for different development processes. They should be considered from the very beginning of the development process.

Below are the research questions I address in this thesis:

- What are the specific characteristics of software systems that development processes should be tailored to?
- What policy recommendations are appropriate for tailoring DoD/USAF software acquisition processes to the characteristics of military software systems?

1.5 Research Methodology
My thesis first provides detailed coverage of DoD and USAF software acquisition processes/policies. This gives readers a good background on how exactly military software systems are required to be developed. The section also provides an overview of recent policy changes regarding software acquisition along with the existing level of tailorability. The thesis then provides a technical overview of software development processes used in both the commercial and defense sector, explaining the differences in process designs. Following that, there is a detailed examination of different characteristics to consider when selecting and designing software development processes with explanations for why certain levels of tailoring are necessary. The final section provides policy recommendations for how the USAF, and possibly the DoD in general, can implement tailorability options within the software acquisition processes so that the required quality and evolvability of military systems do not suffer.
2. Background and State of the Art

2.1 Defense Acquisition
Defense acquisition involves both technical engineering and national policy. Complex state-of-the-art technological systems are built by people using organizational processes and legal frameworks. In the defense sector there is one customer, the government, that sets prices, acceptable risk levels, and business opportunities for sellers. Companies and contractors then have to meet those needs. This is not only a very complex legal and business environment, but also highly political (even though the weapon acquisition process is supposed to operate apolitically). Many constituencies, including the American people, the legislative and executive branches, the Office of the Secretary of Defense, and the armed forces are involved in acquisitions. Each constituent can have varying degrees of influence on the process. While the actual process is very complex, this section is a simple and idealized explanation of how strategic needs outlined in the National Security Strategy become specific software acquisitions.

First, high-level strategic military needs must be identified and assessed. The President and executive branch work on the National Security Strategy (NSS), a document issued by the White House for Congress outlining the major security issues the nation faces and what outcomes the administration hope to achieve. The National Military Strategy (NMS) is issued by the Chairman of the Joint Chiefs of Staff for the Secretary of Defense based on guidance from the NSS outlining the specific strategic goals of the military. Each military service uses these documents as guidance to identify military-specific capabilities that can contribute towards the national strategy. Gaps in existing capabilities are then identified by another detailed process called the Joint Capabilities Integration and Development System (JCIDS), and new acquisition programs can start.
Acquisition programs must pass a series of phases and milestones. As seen in the previous figure outlining the general acquisition phases, programs have many decision points (represented by triangles) where officials must review program progress and decide whether it can/should proceed. Some decision points are also milestones; in DoD acquisitions there are three: Milestone A – initiates technology maturation & risk reduction phase, Milestone B – initiates engineering & manufacturing development phase, and Milestone C – initiates production & deployment phase. These are used to oversee and manage acquisition programs. Details about the specific activities and decision criteria within the process are included in the DoDI 5000.02 [2].

The beginning of new programs consists of a Materiel Solution Analysis phase during which potential approaches and technology options that could be used to close the identified capability gap are weighed against each other in an Analysis of Alternatives (AoA). If an affordable and feasible solution is deemed to exist, the Milestone Decision Authority (MDA) - the executive program head responsible for determining whether a program meets the milestone criteria - can approve the program to proceed.

The next Technology Maturation and Risk Reduction (TMRR) phase is intended to reduce the technology, engineering, integration, and life-cycle cost risks associated with the specific product/design. This is also when the planning for the subsequent phases of the program occurs. Capability requirements should be matured and validated, and affordability caps finalized during this phase before resources are committed. Safety and security requirements are considered and
set during this phase. Based on the analysis of the proposed material solution and the approved funding, the MDA can approve the program to proceed.

The Engineering and Manufacturing Development phase is broken up into three decision points. The Capability Development Document (CDD) Validation Decision is when the product/program requirements for preliminary design, development, and production activities are approved/reworked. The Development Request For Proposal (RFP) Release Decision is when the proposal to industry sourcing bidders is approved/reworked. By this point, all program risks should be understood and under control; the program plan should be sound, and the program should be affordable and executable. Milestone B is when the decision to award development contracts is made and resources are committed. Developmental Test and Evaluation (DT&E) activities – used to determine the capabilities and limitation of the developing system - along with early Operational Test and Evaluation (OT&E) events occur during this phase.

After development, the Production and Deployment (P&D) phase starts after passing through Milestone C. The stability of the system design is evaluated, requirement fulfillment assessed, and maturity of software capability compared with the development schedule at this milestone. This is to ensure that the program risks are still at a level deemed acceptable by the MDA. During the P&D phase, the commitment to low-rate initial production are made where requirements-compliant products are produced and deployed to operational units. OT&E organizations conduct fielded tests under realistic combat conditions to determine the system’s operational effectiveness, suitability, and survivability. If the system is deemed capable of condition mission operations, the appropriate authority will declare Initial Operational Capability (IOC). The Full-Rate Production/ Full Deployment decision point is when the MDA reviews the results of OT&E, initial manufacturing, and limited deployment before choosing whether or not to approve Full Operational Capability (FOC). Demonstration of controlled manufacturing, acceptable performance and reliability, and establishment of adequate sustainment and support systems are required.

The Operations and Support (O&S) phase is for sustaining the system over its life cycle, to include disposal. Performance should be continuously monitored and necessary resources provided to maintain the capabilities of the system. Affordability and increased efficiency are considerations made alongside ensuring that requirements are still being met by the operational system. Weaknesses or deficiencies should be identified and feed back into the overall acquisition process to inform future fixes and improvements. At the end of the system’s useful life, it should be demilitarized and disposed of safely and securely.

This document-driven process helps with coordination and oversight; the lock-step approach that prevents proceeding without completing previous phases and approvals to pass decision points gives government authorities the power to have close management of programs.
2.1.1 Defense Software Acquisition

With regards to software, the DOD-STD-2167 “Defense Systems Software Development” standard published in 1985 stated that software development contractors “shall implement a software development cycle that includes the following six phases: Preliminary Design, Detailed Design, Coding and Unit Testing, Integration, and Testing” [18]. The standard instructs that phases occur sequentially with the required documentation done before each phase. Contractors shall “define and analyze a complete set of functional, performance, interface, and qualification requirements” [18].

The following DOD-STD-2167A standard was implemented in 1988 with the purpose of reducing the structure of the preceding one. For example, it required contractors to only provide one page of general requirements for coding standards, allowing them to only have to address relevant quality factors in the Software Requirements Specification (SRS) and not the 11 quality factors required previously. However, the linear phase-based structure was maintained; all activities, sans coding, culminate with a formal review or audit by the government. The structure provides a common frame of reference for new government and developer personnel to enter projects at different times. However, this rigid structure faced criticism by contractors because it assumes that design begins after requirements are specified, reviewed, and approved and does not allow coding activities to begin until the design has been documented, reviewed, and approved. Contractors were unable to utilize their own development processes [19]. The following figure outlines the generic process of a software intensive program:

![Figure 2. Defense Unique Software Intensive Program [2]](image)

However, Defense of Department leaders recognized that the development process used to develop tanks and ships is not designed for the faster world digital systems. Large-scale acquisition reforms have been conducted, some with a specific focus on developing new processes for digital acquisitions. The Department of Defense now has a preference for a diversity of methodologies, by replacing DOD-STD-2167A with MIL-STD-498 “Software Development and Documentation” in Nov 1994, which introduced evolutionary acquisition and
incremental development processes as program strategy options. The following figure outlines the generic process of an incrementally deployed software intensive program:

![Generic Process Diagram](image)

**Figure 3. Incrementally Deployed Software Intensive Program [2]**

According to the DAU Milestone Document Identification tool [20], Major Defense Acquisition Programs (MDAP) at Milestone A must fulfill 9 statutory and 23 regulatory milestone/phase information requirements in order to move on. Even for Acquisition Category (ACAT) III or below programs, there are 6 statutory and 19 regulatory information requirements. With the 2016 NDAA Section 804 Authority and the introduction of rapid Middle Tier Acquisition (MTA) approach, there has been a recent policy push to accelerate the development of capabilities and promote innovation. The approach consists of utilizing two acquisition pathways [9]:

1. **Rapid Prototyping** – use innovative technologies to rapidly develop fieldable prototypes to demonstrate new capabilities and meet emerging military needs. Programs under this pathway should field a prototype that can be demonstrated in an operational environment and provide for a residual operational capability within five years of the development of an approved requirement.

2. **Rapid Fielding** – use proven technologies to field production quantities of new or upgraded systems with minimal development required. Programs under this pathway should begin production within six months and complete fielding within five years of the development of an approved requirement.
2.1.2 Air Force Software Acquisition

In response to the national policy, Dr. Will Roper, the Assistant Secretary of the Air Force for Acquisition, Technology, and Logistics, released three memorandums related to rapid acquisition policy in 2018. The first provided seven ‘steps’ for incorporating rapid prototyping into acquisition. Programs were required to: (1) Have an aggressive goal, (2) Bound your risks, (3) Be aggressive but not greedy, (4) Constrain time and budget, not the final performance, (5) It takes a team to go fast, (6) Get a signature from [the Assistant Secretary of the Air Force for Acquisition, Technology & Logistics (AT&L)], (7) GO FAST [21]. The second memorandum outlined responsibilities and mandatory procedures for rapid acquisition activities using rapid prototyping or rapid fielding authorities from Section 804, challenging employees in the Air Force to remove “100 years of total schedule from the Air Force acquisition portfolio” using the new policy changes [22]. The third focused on tailoring DoDI 5000.02 programs by adopting Section 804 methodologies to tailor down acquisition plans by removing “all steps that aren’t needed with brutal minimalism” [23]. It encouraged traditional 5000.02 programs to find innovative ways to tailor their plans for increased speed, even establishing Golden Scissors Awards for those that are the most creative [23]. With these memorandums, the Air Force is giving programs the go-ahead to try and accelerate processes by tailoring them to reduce and bundle steps based on new information that may not have been known at program initiation. Dr. Roper recognizes that this may increase risks, but encouraged the acquisitions community to “run with scissors”, “keep taking risk”, and “keep making your bosses ‘good nervous’” [23].

Solid guidance for implementing rapid acquisition activities in the Air Force is detailed in the 27 June 2019 Air Force Guidance Memorandum for Rapid Acquisition Activities, which is a reissue of the first version released in 2018 [22]. With regards to software, it requires that all new initiatives use Agile software development (detailed in Section 2.2.4 of this thesis) and development operations (DevOps) unless waived by the MDA for cost, schedule, performance, or national security considerations [24]. Additionally, the software development strategy used has to be documented in the Acquisition Strategy Document (ASD).

The Air Force has pushed to adopt Agile and DevOps practices for software development. While Agile addresses communication gaps between customers and developers, DevOps aims to bring software development (Dev) and software operations (Ops) teams together. It focuses on providing continuous integration (consolidating all new source code into a shared version control server) and continuous delivery (deliver software in short cycles, ensuring that the software can be reliably released at any time) [25]. DevOps practices advocate for automation and monitoring of software development phases. DevOps teams work to eliminate silos, encouraging collaboration and communicating between functional teams. DevOps practices can be used to rapidly prototype, test, deploy, and improve software products. The USAF sees Agile and DevOps as promising for encouraging innovation and acceleration.
On 23 January 2020, the DoDI 5000.02 was completely overhauled, introducing new Adaptive Acquisition Framework Pathways, outlined below:

With this new framework composed of six acquisition pathways, instead of having to navigate the previous generic acquisition process to deliver capabilities that involve IT, program managers (PMs) are able to choose pathways that fit the unique characteristics (e.g. complexity, risk, urgency) of their program to deliver capabilities to the field as quickly as possible [27]. The policy hopes to empower program managers to think critically and leverage the data they have on their programs for better decision-making. According to Ellen Lord, the Under Secretary of Defense for Acquisition and Sustainment, this is the most transformational policy change the defense acquisitions community has seen in decades; it embraces delegating program oversight to minimize bureaucratic processes and actively managing risks unique to program characteristics [28]. It gives program managers the legal go-ahead to tailor creative solutions to acquisition problems [29].

Part of the new framework involves the creation of a pathway specifically for the development and delivery of software. The DoD realizes that software and hardware must be treated differently; it has gained insights from the commercial sector and is trying to use them for its own software development processes. The purpose of the pathway is “[t]o facilitate rapid and iterative delivery of software capability (e.g., software-intensive systems and/or software-intensive components or sub-systems) to the user” [26]. It is designed to integrate modern commercial software development practices such as Agile, DevOps, and Lean practices with small cross-functional teams that include operational users, developers, testers, and cybersecurity experts to rapidly and iteratively deliver software that meets the most prioritized user needs. The
following figure outlines the pathway, where the loops represent software iterations as capabilities are developed and the diamonds represent software releases that also double as Minimum Viable Product (MVP) and Minimum Viable Capability Release (MVCR) decision points where working software is reviewed:

**Figure 5. Software Acquisition Pathway [26]**

Compared to software-focused development processes outlined in the old version of the DoDI 5000.02, the new pathway only has two phases. The planning phase is for understanding user and system needs. By actively engaging users, the development team can understand concept of operations, the environment, external systems the developed system must interface with, interoperability requirements, threats, existing capabilities, and other specific needs. Once the necessary context is understood, the capability can be designed using enterprise services. By the end of this phase, a few business decision artifacts should be produced and approved by the decision authority. The documentation must include, but is not limited to [30]:

- **Capability Needs Statement:** Captures high-level needs with enough detail to define the software solution space and threat environment. This must be updated periodically to reflect the actual baseline

- **User Agreement:** Agreement between the operational and acquisition communities to commit to continuous user involvement and assign decision-making authority in the development and delivery of software capability releases, as well as operational tradeoffs among software features, cadence, and management of the requirements backlog. This document ensures that operational user involvement is resourced effectively

- **Acquisition Strategy:** Description of the overall acquisition approach, including key decisions such as:
  - Contract type(s) and contracting strategy to be used
  - Approach to cost-effectively obtaining appropriate intellectual property rights and maintaining an open system architecture
  - Development and test platforms, test resources and infrastructure
  - Identification of dependencies with other programs, either in development or in operation
  - Cadence for operational delivery of the software being acquired
  - Roadmap that describes short-term plans with fidelity while outlining longer term objectives, among others
  - Identification of sustainment factors such as upgrades, security, and performance

The strategy documents the program approach to obtain the required capabilities with acceptable risk, aiming to field a software capability within one year or initiation of the execution phase, and to collect data on fielded software to facilitate continuous
engineering. Follow-on software development efforts will be established in increments of one year or less and evaluated based on software performance in the field. The acquisition strategy should also align to modern software development principles, emphasizing iterative deliveries of quality software early and often, and incorporating frequent feedback from users and other stakeholders

- **Cost Estimates:** Initial cost estimate for the program lifecycle. Estimate should consider the technical content of the program described in the Capability Needs Statement, User Agreement, and Acquisition Strategy; factors associated with continuous capability delivery; and factors associated with an emphasis on software security, software quality, and functionality. The initial cost estimate should be completed prior to entry into the execution phase to support contract initiation, and should be updated at least annually.

Design decisions made during the planning phase (e.g. systems and software architecture, software integration strategy, software development factory and pipelines) are critical and have a major impact on future program cost and schedule during the execution phase. The execution phase is focused on iteratively building up software features with shorter development cycles at a higher frequency. The goal is to scope, develop, and deploy a MVP and MVCR to the user as quickly as possible. MVPs are supposed to demonstrate initial capabilities so users can provide feedback on the working software for updating the needs/requirements and designs of future iterations. The development process of the functionality should integrate quality control, testing, and user engagement to improve its value. Wanted capabilities/features should be decomposed into a prioritized backlog. The development team should work with key stakeholders to achieve a continuous authority to operate or an aggressively streamlined approval process for software delivery [30]. These processes all reflect Agile principles. After the MVP is delivered, the development team should iteratively design, develop, test, and delivery capabilities that meet user priorities. Program teams should continuously assess progress at every iteration to ensure strategic goals remain the same and are being met, and remain responsive to emerging or changing requirements by delivering frequent releases to end users [30].

The key to success is for government and vendor personnel to continuously improve their processes, practices and roles. According to the policy, this is best done with small empowered teams that scale their efforts across the program, tracking improvements with established metrics [30]. Issues, errors, and defects during operation should be captured and entered in to the program backlog and addressed in later iterations. The software architecture should be monitored and updated to reflect the current software design using automated tools. By continuously monitoring program status, value assessments can be used to compare investments for short-term capability development and the need for enduring long-term solutions. Business decision artifacts (documentation) must include, but are not limited to [30]:

- **Enterprise Services and DevSecOps Pipeline:** Program should develop a plan to leverage enterprise software development services, and use modern tools and technology to develop and deliver software. Using an enterprise-level platform focused on continuous integration and delivery can provide integrated software tools, services, and standards that enable partners and programs to develop, deploy, and operate applications in a
secure, flexible, and interoperable fashion. Enterprise services can span multiple programs and can be scaled and resourced to support demand.

- **MVP, MVCR, and Product Roadmap:** Delivering an MVP demonstrates to users and stakeholders that the program is capable of cost-effectively delivering reasonable increments of needed functionality. A MVCR, the first version of the software that contains sufficient capability to be fielded for operational use, provides baseline capabilities for validating assumptions and determining if the proposed system will deliver the expected or acceptable business/mission value. The product roadmap is a plan for the delivery of multiple software releases over time to reach full capability. It identifies which releases will be evaluated/fielded for operational use, and the point at which the software should transition to sustainment. The roadmap can evolve over time to accommodate changing threats and technology. It projects the planned evolution of the solution capabilities and architecture, and communicates the capabilities/feature sets targeted for delivery at discrete times in an iterative fashion.

- **Test Strategy:** Defines the process by which capabilities, user features, user stories, use cases, etc., will be tested and evaluated to satisfy developmental testing criteria and demonstrate operational effectiveness for operational testing to the maximum extent possible. Specific components of the strategy are listed in the policy document. Automated testing should be used at the unit level, for Application Programming Interface (API) and integration tests, and to the maximum extent possible for user acceptance and to evaluate mission effectiveness.

- **Secure Software & Cyber Security Plan:** Program managers should establish and/or leverage a secure software development pipeline and a security lifecycle plan. Software tests should run automatically where possible, and at a predetermined cadence sufficient to ensure that cybersecurity controls and other considerations are addressed early and throughout the acquisition process. Program managers should establish the conditions to enable a continuous Authority to Operate (ATO) where appropriate. Ensuring software security includes secure development (coding, test, identity/access management, supply chain risk management), secure capabilities (software patching, encryption, runtime monitoring, and logging) and secure lifecycle management (vulnerability management and configuration control). Automated build scripts and test results should be available to the test community so that critical verification functions (e.g., performance, reliability, etc.), validation functions (e.g., effectiveness, suitability, and survivability) can be assessed iteratively and incrementally. The automated cyber testing should be designed to support a continuous ATO if possible or an aggressive accreditation process otherwise; and should be augmented with additional testing where appropriate in accordance with the DoD Cybersecurity Guidelines.

- **Metrics Plan:** Identifies key metrics that allow the program manager and other stakeholders to manage cost, schedule, and performance. It also organizes metrics by common types (or classes) and provides guidance on how to read and interpret each metric. Each program should tailor the set of metrics for the unique considerations of the program. All software acquisition programs must have a set of core metrics.
- **Value Assessment**: The program sponsor and user community should perform a value assessment at least annually and provide justification for resources expended to deliver capabilities. End users will determine if the mission improvements and/or efficiencies gained from the delivered software capabilities are worth the investment. The decision authority will use the value assessment to measure progress on the program and inform resourcing decisions.

The following figure outlines the different roles involved in software acquisition, along with the business artifacts (documentation) required throughout the process:

![Figure 6. Lifecycle View of Software Acquisition Pathway [30]](image_url)

The USAF has already been implementing this new software acquisition pathway, even before the updates to the DoDI 5000.02 were made at the beginning of 2020. Kessel Run [31], an Air Force run software development factory, was established in Boston, MA. It is composed of active duty members and civilians that strive to utilize in-house capability to develop usable software faster than traditional software acquisition methods. Kessel Run practices lean product development, extreme programming, and user centered design to develop software solutions quickly to users in the Air Force. To develop functional and innovative software, they use Agile and DevOps to quickly and continuously deliver software iterations in a classified DoD environment. By showing that the government is able to leverage commercial software practices to develop and deliver software quickly, other software factories have been established. Kobayashi Maru and Section 31 [32] in Los Angeles, CA are building software tools for space command and control and situational awareness using iterative development and guidance from Kessel Run. LevelUP in San Antonio, TX is developing and testing offensive and defensive cyber tools. It is working on developing a secure platform that DoD programs can leverage.
2.1.3 Tailoring

With the updates to the DoDI 5000.02, acquisition process policies are now a lot more open-ended and give program managers more authority to tailor their programs how they see fit.

Under the current policy, MDAs and decision authorities (DAs) have the power to specify the decision points and procedures for programs. They are responsible for tailoring program strategies and oversight, phase content, timing and scope of decision reviews, and decision levels based on the characteristics of the capability being acquired (including complexity, risk, and urgency) [26]. The are the high-level approval authority for the acquisition strategy at all major decision points. One level below that are the Program Executive Officers (PEOs) who are responsible for balancing the risk, cost, schedule, performance, interoperability, sustainability, and affordability of a portfolio of acquisition programs [26] and delivering capabilities to users. Under the supervision of PEOs, PMs do the detailed planning for individual acquisition programs, preparing them for key decisions, and executing approved acquisition and product support strategies [26].

With the approval of MDAs/DAs, PMs can develop acquisition strategies that match the processes, reviews, documents, and metrics outlined in the six acquisition pathways of the Adaptive Acquisition Framework to fit the unique character and risk of their program. When employing multiple pathways, PMs will [26]:

- Define the transition points from one pathway to another
- Anticipate, develop, and coordinate the information requirements required at the new pathway entry point. Documentation requirements will vary depending on the regulatory/statutory requirements for each pathway
- Ensure a smooth transition between pathways

When developing acquisition strategies, PMs also have the responsibility to [26]:
- Design program and business strategies for acquiring appropriate and cost-effective technology solutions and achieving mission success while being mutually advantageous to the DoD and its industry partners. PMs will consider strategies that leverage international acquisition and supportability planning to improve economies of scale, strengthen the defense industrial base, and enhance coalition partner capabilities to prepare for joint operations
- Recognize that cybersecurity is a critical aspect of program planning. It must be addressed early and continuously during the program life cycle to ensure cybersecurity operational and technical risks are identified and reduced so that fielded systems are capable, effective, and resilient
- Consider the procurement of data deliverables and associated license rights needed to support competitive acquisition and life-cycle sustainment strategies
- Prioritize product support and affordability during early program planning to ensure sustained mission effectiveness

- Establish a risk management program to ensure program cost, schedule, and performance objectives are achieved, and to communicate the process for managing program uncertainty. In consultation with the user representative, the PM will determine which environment, safety, and occupational health risks must be eliminated or mitigated, and which risks can be accepted

- When consistent with pathway requirements, develop engineering plans and processes applicable to the pathways to mature technology, conduct necessary systems engineering tradeoffs, and produce and manage appropriate technical baselines through the use of systems engineering technical reviews

Along with all these complex and difficult responsibilities placed on the PM, they must “tailor in” the regulatory information requirements that will be used to describe the management of the program. In the policy, “tailoring-in” means that the PM will identify, and recommend for MDA/DA approval the regulatory information that will be employed to document program plans and how that information will be formatted and provided for review by the decision authority [26]. Therefore, the only tailoring authority the PM actually has is over recommending the legally required documentation of their programs, and not with how the actual capability will be developed using a tailored process.

Furthermore, the figures below from the Defense Acquisition University outline several suggested “tailored” processes that are combined with or transition to the new software acquisition pathway:

![Figure 7. Middle Tier of Acquisition (Rapid Prototyping) Pathway then transitioning to Software Acquisition [33]](image-url)
As seen in the figures, the DoD policy interprets ‘tailorability’ as mixing and matching different pathways, but does not have guidance regarding how each pathway itself should be tailored. The current level of tailorability available in the policy is focused on management and legal oversight rather than the engineering required for developing systems. It fails to consider one of the main themes from the SWAP study; that the characteristics of software can differ from system to system. According to the policy, the new software acquisition pathway is meant for software-intensive systems and/or software-intensive components/subsystems, which would encompass many different military systems with varying levels of risk and performance/quality requirements. A single software acquisition pathway is insufficient to address all software system characteristics.

Congress has identified that different software systems require tailored development processes fit for certain characteristics. The acquisition policy and management section of the FY2020 NDAA includes Section 801, which establishes two new acquisition pathways: an application pathway for software running on commercial commodity hardware operated by the DoD; and an embedded systems pathway for software embedded in weapons systems and other military-unique hardware systems [34]. However, this new DoD-level policy is not effective if the military service-level developers do not have clear guidance on how exactly to tailor software pathways themselves. Additionally, PMs and decision authorities, who already have many different responsibilities, do not have existing guidance on what specific software system characteristics to focus on when choosing and designing processes.
Having multiple acquisition pathways and flexibility for combining/transitioning between them is a step in the right direction, but policy regarding tailoring specific pathways still does not exist. Software is different than hardware, and not all software is the same. Therefore, there is a USAF service-level policy gap regarding guidance for tailoring the software acquisition pathway to the characteristics of different software-driven systems.
2.2 Software Development Processes
Before looking into how the USAF software acquisition process can be tailored, it is important to gain a basic understanding of software development and how different processes can be used to develop different types of systems.

Software development consists of structured approaches that divide the work necessary into defined smaller processes. The software development life cycle (SDLC) generally involves six processes:

1. **Requirements Analysis:** stakeholders and program managers work to determine requirements and plan the basic project approach. The goal of this process is to assess feasibility, which can involve different aspects, including operational, financial, technical, and legal/political feasibility to plan for the best way to meet user requirements successfully with minimum risks. Functional requirements (which describe the behavior of the envisioned system) and non-functional requirements (which elaborate performance characteristics of the envisioned system) are documented, which serve as guidelines for the design process, and checkpoints for the test process.

2. **Design:** The requirements from the previous process are translated into design specifications, which help to define the overall system architecture of the envisioned system. The design is reviewed by stakeholders who evaluate it based on parameters such as risk, modularity, changeability, and resource constraints (budget and time). In this process, the design is configured to minimize projected schedule and costs while maximizing quality as much as possible. The architecture design should be detailed enough, with communication/data flows and all modules/subsystems defined (including external/third party modules) so coders and engineers in the next step have a clear idea of what to build.
3. **Implementation/Coding**: Using the software design from the previous process, code is created by developers using the appropriate programming language(s) and tools determined in earlier processes or by the organization developing the code. Engineers are responsible for dividing the work as necessary and delivering an end product that satisfies the requirements and design determined in the two earlier processes.

4. **Testing**: The code produced in the previous process is tested against the requirements that were gathered and set in the first process. This is where the separate modules/systems are integrated. Code is tested at unit, integration and system levels to check that the overall product satisfies the quality standards defined in the first process. The code should be revised by the developers in the previous process until it passes acceptance testing, which is used to determine whether or not a system satisfies the acceptance criteria and to enable the user, customers, or other authority to determine whether or not to accept the system [35].

5. **Deployment**: After software passes the testing process, it is released to the user. Depending on the organization and industry, the deployment process may require approvals before the working software is deployed. Once users use the software, they should be able to identify bugs and ask for changes that the coders/engineers can address. This process is highly coupled with the testing process and feeds back to the coding process. The feedback loop is part of the software release life cycle; ‘alpha’ and ‘beta’ versions of the software could be released until a production version is ready to be deployed.

6. **Maintenance**: Even when software is deployed and being used, the development process is not done. Software maintenance can be categorized into corrective (e.g. emergency fixes, routine debugging) [36], adaptive (e.g. accommodate changes to data inputs and files, changes to hardware and other system software) [36], perfective (e.g. user enhancement, improved documentation, increased computational efficiency) [36], and preventive (e.g. increasing maintainability/reliability to prevent future problems) activities. They help to ensure that software functionality is maintained and can potentially be improved throughout its operational life. Maintenance should continue until the software is not needed anymore or replaced.

The fundamental nature of software allows for it to be continuously implemented, tested, deployed, and maintained/improved at a much faster rate than hardware. Software engineering does not involve manufacturing, as once software is developed it can be replicated and deployed quickly. However, that means it is possible to deploy faulty software quickly and only find out about the faults in operational use, when the faults can be extremely costly.

Software can be developed using different methodologies. Process models provide different ordering of technical activities to tradeoff cost, schedule, and performance. Methodologies typically differ in aspects such as cycle time, priority of life cycle processes, and the availability of feedback loops. In this section, four different software development processes are described, along with the software characteristics they are most appropriate for.
2.2.1 Waterfall Methodology

Based on the literature available, the methodology was first defined and examined closely in a 1970 paper titled ‘Managing the Development of Large Software Systems’ [37] by Dr. Winston Royce, an American computer scientist. Although not explicitly referred to as a ‘waterfall’, this software development process is a linear sequential model. Software is developed in sequential phases and does not progress until phases preceding the current point are completed. However, that does not mean that there is no backtracking to a previous phase. A baseline is completed at the end of each phase that acts as the basis for the next phase. If changes are necessary, then the previous baseline may be changed and the process continues forward again. Phases generally should not overlap because the methodology is designed so output(s) from one phase become input(s) for subsequent phases. Additionally, this approach is document-driven, which means that documentation is a critical part of the development process. It is also planning-driven, which means thorough analysis is conducted at the very beginning, even before coding, to generate requirements and design documents that serve as guidelines for the rest of the process. More effort at the front end to understand what needs to be developed is thought to limit the amount of backtracking that is necessary.

The figure below represents the Waterfall process flow using the SDLC processes detailed in the previous section:

![Figure 11. Waterfall Software Development Process](image)

In Dr. Royce paper, the development phases are instead: System Requirements, Software Requirements, Analysis, Program Design, Coding, Testing, and Operations in sequential order [37]. Reviewing most literature on the Waterfall methodology will reveal that although some figures have more/less phases and phases with different names, the general software development process is all the same. There is always analysis at the beginning which involves setting requirements, analyzing feasibility, and determining design. Only after those activities are done does coding start.
A specific example of the Waterfall methodology used in industry is the V-Model, or V-Modell, developed for the German defense sector and has been the Software Development Standard of the German Federal Authorities since the summer of 1992 [38]. The V-Model concept was also developed simultaneously, independently from the Germans, in the U.S. and was documented in the 1991 National Council on Systems Engineering proceedings [39]. The model is a visual representation of the development process with the left side of the “V” shape representing the decomposition of requirements and creation of more detailed system design, and the right side representing the integration and testing steps of the system [40]. When proceeding from left to right (the horizontal access represents development time), there are verification and validation (V&V) activities linking the arms of the “V” creating loops where the working integrated system can be checked against the concept of operations, requirements, and design.

In its general form, the Waterfall methodology is appropriate for developing software systems with these characteristics [41]:

- Clear requirements, stable product definition – full scope of project known in advance, requirements not likely to change often
- Mature technology – not likely to change drastically during development period, low technical risk
- Large, complex systems: Waterfall focuses on close documentation of code and progress tracking
- Large development teams: Waterfall can be developer agnostic – detailed documentation and upfront planning provides enough information to continue development even with changes in personnel
- Critical performance requirements – safety/security-critical, requires rigorous analysis and setting sufficient requirements: Waterfall prioritizes requirements analysis and design processes prior to coding
- High levels of integration with external systems: Waterfall prioritizes thorough design and planning processes
- Delivery urgency not utmost priority: Waterfall has longer cycle times compared to others more focused on deploying software faster
- Long expected lifespan: maintenance and evolvability considerations are made during initial development
- The software is part of a larger, complex system development with hardware and human operators that must interact with the software or what the software is controlling
2.2.2 Incremental Model
This model combines elements of Waterfall in an iterative fashion so that functionality is developed and delivered in small increments. Each linear sequence produces working software versions that can be delivered to customers/users for feedback. As development continues, functionality is added. Defects or issues identified in previous versions can be addressed in future increments. The process model helps developers focus on delivering essential features first while keeping the option for adding functionality if/when needed. Development iterations should continue to cycle until users deem it unnecessary to keep dedicating resources to development, at which point development ceases.

The figure below represents the Incremental process flow using SDLC processes:

![Incremental Software Development Process](image-url)

Figure 12. Incremental Software Development Process

Overall system requirements are addressed first and remain the core basis of all future software iterations. However, not all features are completely known at the very beginning of development, so this process allows for additional features to be developed and delivered in the future. The priority of the process is to develop important functionalities first so that they can be refined with user feedback in future iterations. This way, development risks are spread across multiple smaller increments instead of one large development effort. Lessons learned from each incremental delivery can also be implemented in the future. Additionally, users get important functionality early and have the opportunity to provide feedback on each increment, which reduces future surprises such as mismatch of expectations. One common trap, however, is that critical properties that are not visible but are required for the final system, such as fault tolerance or performance issues, may not be considered and incorporated until the end, when it may be impossible to add them successfully.

The Incremental model requires rigorous initial planning and design. There needs to be an early definition of the functional software system that is as complete as possible in order to effectively split the whole program into defined increments that are not only achievable but fit within the resource constraints of the program. Functionality and performance requirements that cannot be
divided into increments must be planned from the beginning. Additionally, there needs to be a well-defined interface so that current software versions are able to be maintained and upgraded in the future. It is important for developers to carefully choose what functionality is put off to later in order to prevent escalating costs and schedules associated with making changes too late in the development process.

The general Incremental model is appropriate for developing software systems with the following characteristics:

- Well-defined functionality requirements: Incremental model prioritizes developing most essential functionalities first
- High delivery urgency: Incremental model can be used to deliver initial functionality to users quickly
- Leaner systems: Incremental model helps prevent over-functionality syndrome
- Moderate to high technical risk: initial increments in the model can be used to lower the risk of future software iterations but may also increase overall project risk if planning is not done well
- Complete/close to complete system definition – scope of project known in advance; development can be easily broken down into increments
- Smaller development teams: developers need to be highly coordinated to understand previous software iterations and how to add future functionality
2.2.3 Spiral Model

This risk-driven approach to software development focuses on assessment and minimization of overall project risk. It was first described by Barry Boehm in his 1986 paper titled ‘A Spiral Model of Software Development and Enhancement’ [42]. Starting from an overall concept-of-operation document, individual software portions are developed incrementally after undergoing thorough risk analysis that culminate in an operational prototype which informs the development of the final product.

The figure below represents the Spiral process flow, with the vertical axis representing cumulative development costs and horizontal axis representing commitment to the development:

![Figure 13. Spiral Model Software Development Process [42]](image)

Each cycle of the spiral begins with the identification of: (1) objectives (performance, functionality, ability to accommodate change, etc.) for the portion of the product being developed, (2) alternative means of implementing the portion of the product (consider different designs, reuse code, buy off the shelf, etc.), and (3) constraints, or sources of project risk (cost,
schedule, interface, etc.), imposed on the application of the alternatives [42]. Functionality is incrementally added in “spirals” until the software prototype is fully developed according to the specified requirements. Risk analysis is conducted continuously with prototyping. At the end of prototype development, lessons and designs from the operational prototype are supposed to be used to conduct the final product development. The prototype is then meant to be disposed of.

There are three approaches that the Spiral model can be used for: (1) requirements analysis through prototyping, (2) accommodating for design uncertainties as prototype evolves into final product, (3) experimenting with different solutions to assess feasibility before committing to a large investment. There is a heavy emphasis on design, analysis, and prototyping to ensure commitment and cost do not escalate uncontrollably as development continues. As a result, the process needs strong approval and documentation controls. For overall risk management, project development is broken into smaller segments, which makes it easier to implement changes and evaluate the risks with continuing efforts.

In its general form, the Spiral model is appropriate for developing software with the following characteristics:

- Risk management is critical to success – high project risk (related to cost, schedule, system performance), requires rigorous analysis and active management
- Small set of known initial requirements; exact full system requirements, design, and feasibility not certain
- Scope not expected to increase drastically: Spiral model has to be followed closely with thorough documentation and close management since phases are not clearly defined
- Maturing technology – future requirements are unclear and complex: Spiral model allows for changes to be made in future increments
- Large project: Spiral model requires enough budget to develop prototype and expertise to conduct thorough analysis; resource commitments may not be worthwhile for smaller projects
- Not resource constrained – needs schedule and budget to plan, reset objectives, conduct risk analysis, and develop prototype; long term project commitment not likely to change due to changes in economic priorities
- Long lifespan: maintenance and evolvability considerations are made continuously throughout development in the Spiral model
- Frequent increment releases: fixes and additional functionality are implemented regularly in the Spiral model
- Customized uses: risk can be analyzed for specific problems
2.2.4 Agile Methodology

The Agile methodology focuses on the continuous iteration of development and testing to rapidly develop applications. It prioritizes prototype releases and multiple iterations; it emphasizes the use of working software and user feedback over strict planning and in-depth documentation.

In February 2001, seventeen software developers met at a resort in Utah to discuss lightweight software development methods and published the *Manifesto for Agile Software Development*. Based on their experiences with software development, the Manifesto signatories valued: (1) Individuals and interactions over processes and tools, (2) Working software over comprehensive documentation, (3) Customer collaboration over contract negotiation, and (4) Responding to change over following a plan [43].

While these are values that often match well the “lightweight” commercial application environment, the question is whether these values fit the framework that large, long-lived government projects must necessarily fit within or whether they are the most important values for defense systems that may live for 30 or more years, with continuing upgrades by those who may not have been the original developers. They cannot just be thrown away when business needs and sales opportunities change and capturing competitive markets is important.

The figure below represents the Agile process flow using SDLC processes:

![Figure 14. Agile Software Development Process](image-url)
The requirements analysis process is less focused on producing a detailed document with functional and non-functional requirements, but rather serves as a phase where users, stakeholders, and the development team work together to list user needs. Needs and requirements are listed on a backlog, which are reviewed and prioritized based on business or user values. Each iteration, or sprint, is given a limited amount of time for development. Tasks are broken down from the product backlog. The design and coding processes are typically done by small development teams that are typically co-located and use frequent face-to-face communication. Daily team meetings help ensure that managers and the developers are not only on track, but continue to understand the values of the end users. Software testing is done concurrently during development and code is automatically reviewed at the end of the day by automated tools to check for bugs. After the end of the sprint, live demonstrations are typically used before deploying the working software product. Unfinished tasks or unmet requirements are then put back on the backlog to be addressed in the next sprint. The maintenance process is for changing fielded software, as feedback sourced from users is used to change or reprioritize requirements in the next sprint. By continuously iterating and involving users within the process, Agile utilizes feedback loops, closer interactions, and flexibility to speed up software development while allowing the software product to evolve in response to changes in environments and requirements.

There are many Agile-based software development processes used in industry. The short iteration cycle model has strong connections with the Lean Startup concept detailed by entrepreneur Eric Ries [44]. For startups that face large initial resource and credibility obstacles, investing time into building a minimum viable product that addresses the needs of early customers and then building functionality into subsequent iterations is a method to try eliminate waste and focus efforts into practices that add the most value to the software product. This method of feature-driven development is implemented by startups and resource-constrained organizations to deliver essential functionalities quickly.

Extreme Programming [45] also embodies Agile principles and was created by Kent Beck, an American software engineer and one of the original signatories of the Agile Manifesto. In this framework, iterations can typically take one or two weeks to develop, requirements can be changed within iterations, tasks are addressed in a strict priority order, and specific engineering practices such as pair programming, refactoring, and test-driven development are prescribed for development teams.

Scrum [46] is another development framework and was developed by Ken Schwaber and Jeff Sutherland who are both original signatories of the Manifesto. In this framework, iterations are organized into sprints that can last from two weeks to a month. Unlike Extreme Programming, Scrum does not allow for requirement changes within sprints, user needs are addressed based on the discretion of the product manager (or “scrum master”), and specific engineering practices are not prescribed. As seen by these examples, even specific software development processes can be tweaked and tailored in different ways to fit the characteristics of the software systems being developed.
The general Agile methodology is appropriate for developing software with the following characteristics [47]–[49]:

- Delivery speed and frequency are utmost priority: Agile has short cycle times, users get working software quickly

- Unclear requirements, unstable product definition – full scope of project not known in advance, requirements likely to change often, users may not know exactly what they want

- Maturing technology: future changes can be implemented quickly using Agile

- System optimization is not a priority: Agile is suitable to develop software that just needs to work minimally and be deployed quickly. Quality is not as important as speed. Errors and flaws can be fixed later and are not critical.

- System performance requirements are not critical: Agile focuses on delivering working software quickly and sourcing user feedback requirements rather than conducting rigorous analysis and design processes in the beginning

- Smaller, less complex software – Agile places priority on delivering essential capabilities quickly and often; refactoring code for large systems could be costly and error-prone

- Smaller development teams: Agile is best used by physically co-located cross-functional teams that rely on daily coordination; it does not emphasize thorough documentation

- Short expected lifespan: Agile does not make thorough maintenance and evolvability considerations at the beginning of development

- Resource constrained: Agile is designed for minimizing short-term development costs and schedule

- Need stakeholder involvement and feedback: Agile helps find expectation mismatch early and is focused on continuously improving software using user feedback

- Specialized uses: address specific problems/applications

- Safety and mission and assurance are not critical properties. Use in the field is an acceptable way to find errors and problems.
3. Tailoring Development Processes

When considering the processes used to develop software, it is critical to understand what purposes the envisioned software system will satisfy and what kind of problems it will be designed to address. For example, an all-digital application used for logistics has very different requirements and designs compared to software controlling aircraft hardware systems. The contexts under which these systems are designed to operate differ, along with the problems they were designed to address. As a result, the development processes used to develop those systems must also differ because different characteristics of the software must be prioritized. For example, an all-digital logistics or scheduling application may be designed for users who need the system to be delivered quickly and don’t really need a high quality highly performing product; just something with basic functionality. The system may be for a specific problem on the battlefield that may not exist very far into the future, so maintenance and evolvability are not primary considerations. For such a software system, a development process focused on rapid cycle times and iterative development using user feedback executed by a small team of co-located developers may be the most appropriate.

On the other hand, software controlling aircraft or munition hardware systems would require an entirely different development process. The process would ideally prioritize executing clear performance requirements correctly and leveraging rigorous design analyses to ensure that code is of a high quality. Instead of deployment time, documentation would be the focus, as such mission critical systems are typically developed by large functionally separate teams and maintainability and future upgrades are important. Additionally, the system itself may be several million lines of code in size and may require the development process to have clearly set expectations for detailed design and thorough documentation. Such systems, especially used for military purposes, would likely have long lifespans that continue to get extended as developing a completely new system is too expensive and time consuming. Therefore, maintenance and upgradeability are important considerations that must be accounted for during development. For such a software system, a development process focused on documentation and clear task breakdowns using rigorous requirements and design analyses executed by large development teams may be the most appropriate.

The disparity between two software systems is dramatized in the previous two paragraphs to highlight the main point: not all software is the same. Just because a system is composed of or involves code does not mean that it can all be developed the same way. Additionally, the problems that the commercial and defense sector face are not the same. The problems, priorities, and resources of the DoD are rarely similar to those of a startup in Silicon Valley. For the military, mission assurance is absolutely critical. As with anything in systems engineering, cost, schedule, and performance of the program must be balanced. Sometimes deploying working software quickly to an operator in the field is a priority that justifies the compromises. Sometimes developing high quality software that fulfills mission-critical requirements is a priority that justifies the compromises necessary to achieve it.
A one-size-fits-all development process that can achieve the best cost, schedule, and performance goals for software systems does not exist. Different processes prioritize different system characteristics, and are appropriate for certain applications. Individual development processes can be further tailored to better fit specific software system characteristics such as code size, development team size, customer type, schedule pressures, expected lifespan, resource availability, etc. Tailoring processes to fit certain system characteristics is the most applicable approach for the DoD’s need to develop specialized systems designed to address unique problems. The main software system characteristics that different development processes should be tailored to address are listed and detailed within the following subsections.
3.1 User
The end-user of a software system is an important characteristic that must be considered during development, especially during the requirements analysis and design processes. Users that will leverage the system for engineering purposes, such as for embedded control or interfacing with data will usually have strict requirements and design standards that need to be met. Critical systems with high levels of risk will likely use mature technologies and have clear requirements that need to be met. As a result, a detailed documentation and easy to manage/track development processes are recommended to suit the needs of this type of user.

Considerations must also be made for the variety of educational and skill levels of the end users. For example, will the users be high-trained fighter pilots that can be expected to understand all the nuances of the limited number of aircraft they are trained to fly or, at the other extreme, will the users be casual users who must be able to use systems on which their training may be limited or which change frequently.

Another type of user is the business manager, those that leverage software systems to track the operations of their organization, use digital tools to make business decisions, and deliver capabilities. Such users may not have very strict requirements or designs for their envisioned systems, but would much rather prefer systems that fit their specific needs and can be adapted to their personal preferences. They are more lenient of bugs than the engineers, but would prefer a mechanism of feedback to report such bugs and expect them to be addressed quickly. Additionally, since the business world is dynamic and is subject to decisions made by others, software systems need to be flexible and quickly incorporate changing requirements necessitated by the environment. Users may not know exactly what they want during system conception, so it is more appropriate to use development processes that focuses on quick delivery, frequent cycle time, and user feedback rather than detailed analysis and design.

The third type of user are those that use software for casual uses, such as for productivity or convenience. In the defense sector, these users would be anyone using word processors, network browsers, websites, etc. This type of user has the lowest expectations for the quality of the software, as they usually are expected to use what they are given and be able to provide minimal feedback on improvements. This usually means that there are similar software systems offered in the market and transitioning to different systems is a matter of preference. When developing such systems, core functionalities must be provided, but there is not a pressing need to deliver a highly polished product nor are there pressures to deliver quickly. Therefore, processes focused on quickly deploying lean systems with minimal resource use are appropriate for systems developed for this type of user.
3.2 Urgency
The urgency to deploy working software is a characteristic essential to determining what development process should be used. The highest urgency, which means that deploying to the field is critical for the system’s existence or to gain market share before competitors enter the market, requires a development process that focuses on delivering core functionality as quickly as possible. In the defense sector, this would be most applicable to a system used in a wartime setting with changing requirements necessitated by developments on the battlefield. Such a system would need to have a very detailed purpose addressing specific problems or changes. Additionally, they would need to be very adaptive. When urgency is that high, it may be most appropriate to use a development process focused on delivering core functionality quickly, but is also tailored with rigorous analyses for mission assurance. Such a tailored process may require a high budget, which would be justified by the fast schedule demands and performance requirements.

Some software systems may consider deployment urgency to be important, but not enough so to dedicate the full investment of resources for the quickest development schedule possible. For example, planned software upgrades for operational systems or periodic patches to address vulnerabilities that can impact mission success would need to be deployed on time. Therefore, development processes that reduce program risk to ensure necessary capabilities are developed on time, tailored with analyses and design processes that take a little more time but are more thorough and consider future upgrades, would be the most appropriate.

For some software systems, the urgency to deploy would be of minor importance. For example, the embedded control software for next generation aircraft expected to be delivered in the next decade or highly developmental technologies are not required to be delivered as quickly as possible. The flexibility in scheduling is most appropriate for using development processes that focus on documentation, close management, rigorous analysis and design activities that incorporate potential future system considerations.
3.3 Lifespan

The expected lifespan of a software system is an important characteristic that typically is not emphasized too much during development but is essential for deciding what processes to use and how to tailor them. At system conception, it may be hard to understand the full scope of the software and easier to focus on the immediate needs and problems. However, requirements and designs made at the very beginning of the process prove to be very important for the maintenance and evolvability of the system in the future, especially as context changes and initial assumptions become invalid. Additionally, changes made later in development usually incur cost and schedule overruns as systems not designed for extended use are retrofitted to operate for longer.

Systems with a long expected lifespan must undergo thorough rigorous requirements setting and design refinements processes to ensure relevance in the future. One example of a system designed to have a long lifespan was the Space Shuttle, which provided routine Earth-to-orbit transportation from 1981 until 2011. Each vehicle in the program was designed with an initial projected lifespan of 100 launches, or 10 years of operational life. The National Aeronautics and Space Association (NASA) focused heavily on setting detailed requirements and carefully creating a design that could be maintained and adapted as the Shuttle continued operating for decades. NASA had a specific process to decompose high-level system requirements into specific component and interface requirements [50]. Also, as with space programs consisting of multiple missions, new software functions needed to be added/adapted to different mission requirements. Over 50% of the primary avionics software modules changed during the first 12 Shuttle flights in response to requested enhancements [51]. Modules for unchanging tasks such as main engine throttling, memory modification, and screen displays were fully tested and stored at the Software Development Laboratory at Johnson Space Center. New mission-specific modules for payloads or rendezvous maneuvers in different orbits were developed and tested using the lab’s tools. For each Shuttle flight, existing modules and new modules were selected and combined into an operational increment to be tested. The development process used for the Shuttle software system was a major factor in its success. Careful planning before writing code, detailed requirements specifications, continuous learning and process improvement, a disciplined top-down structured development approach, extensive record keeping and documentation, extensive and realistic testing and code reviews, and detailed standards were some of the main attributes of the process that helped the Shuttle achieve its mission [52].

Software systems with a medium lifespan, those not expected to be continually developed for decades into the future but also not expected to be disposed of soon after a couple years of use, will most likely require less documentation as the design requirements will not be as demanding and the maintenance period will not be as long as that of a long-lived system. However, a certain level of design and documentation should still exist and be tailored to the specific system.

On the other side of the spectrum, software with short lifespans such as applications developed to address a specific need or test an assumption will most likely require very little documentation and analysis since the system is designed to be developed, used for a short time, and terminated soon after. Therefore, processes focused on accelerated delivery and less on future improvements would be more appropriate.
3.4 Performance (timing)
Software performance related to timing deadlines are important, especially for systems that must operate quickly in the field. For example, hard real-time deadlines such as calculations for ballistic missile trajectories and real-time data communications between aircraft and weapon stores to ensure pinpoint accuracy are mission critical for military systems. To achieve high levels of performance, these software systems must be designed to achieve real-time deadlines regardless of the environment or condition. This means that the development process should not only involve rigorous requirements and design analyses to understand exactly how each software component contributes to overall timing performance, but also comprehensive testing and performance validation in a variety of scenarios. With good designs and a thorough understanding of how the system operates, future changes to timing requirements necessitated by changes in the battlefield would be easier to make.

Soft real-time deadlines are those that are important to mission critical systems, but may not necessarily require deep analyses or tight tolerances to function properly. For example, mission planning software that helps operators decide what mission profiles to fly with different kinds of weapon loadouts need to operate fast enough so that flight plans can be generated before a last-minute mission, but even those performance requirements can be adjusted slightly based on the situation. With soft deadlines, tolerances can be wider so processes that do not require full use of available resources may be appropriate. However, a minimum level of documentation and management oversight would have to exist since there are real-time deadlines and performance requirements could become stricter.

Finally, there are systems with no real-time aspects, except that lags in timing could cause user annoyances. Such systems could be productivity tools or management software where long processing times have no real impact on the mission and would simply be an obstacle that users have to navigate by waiting or using manual means to do the same work.
3.5 Quality/Risk
Different systems have varying levels of quality that need to be met and levels of risk that need to be managed. Safety-critical and security-critical systems have the highest quality requirements and need risks to be regulated. For example, flight control software for aircraft and spacecraft have serious hazard implications if the system fails to operate as required or behaves in an unexpected manner that could lead to unintended consequences. Such systems must be designed with critical requirements such as fault-tolerance in mind to actively prevent unsafe conditions that could lead to hazards. Additionally, security-critical systems that fail to prevent access or minimize damage from attackers could compromise missions in the military. These systems are typically required to be certified to certain level of quality standards and risk must be constantly reduced throughout development. Developing these systems is not easy, and documentation-based development focused on regulating risk through rigid and disciplined processes is needed.

However, not all software systems require the highest levels of quality and rigorous risk management. For some, the priority is on large market share and speed to market. While quality has to be acceptable for many users to continuously prefer those systems over alternatives, speed to market in the commercial world is also a factor that makes it suitable for exchanging detailed quality control for quicker deployment of working software. The key to developing these systems is to achieve a balance between assuring a certain level of quality while taking risks through deploying potentially incomplete systems more quickly into the field.

There are also systems with large latitudes allowed for quality and risk. For example, software for developing technologies or experimental purposes do not have high quality standards to meet and are very low risk since they are not expected to be fielded onto operational systems. Using development processes without rigid structures and detailed analyses would be appropriate.
3.6 Size

Many modern systems are software-driven, and software is becoming increasingly prevalent and complex. The figure below illustrates that software size for many military avionics systems, measured by source lines of code (SLOC), has grown dramatically over the decades:

![Figure 15. Growth of Source Lines of Code (SLOC) in Avionics Software [13]](image)

For context, one million LOC is equivalent to about 18,000 pages in printed text. There are systems with well over that many lines when considering not just the core modules but the support software and network software required to operate them. Therefore, as systems increase in size, it is more appropriate use development processes that are designed with more comprehensive documentation and management so that developers are able to understand how the software was designed and coded.

For categorization purposes, this thesis considers small systems to be under 100,000 SLOC, or the equivalent of about 1,800 pages in printed text. Medium systems are classified as 100,000 SLOC to 1,000,000 SLOC, the equivalent of about 1,800 pages to 18,000 pages of text. Large systems are classified as over 1,000,000 SLOC, or over 18,000 pages. Given that more modern systems such as the F-35 have over 6 million SLOC in airborne software alone according to Figure 15. Growth of Source Lines of Code (SLOC) in Avionics Software [13], the categorization criteria for different size systems will only increase into the future.

It is important to note that SLOC is not the only metric that characterizes the size and complexity of software systems, as different attributes such as programming languages and architecture designs are also factors. However, when modern systems consist of tens of millions of lines, it becomes clear that development processes with documentation management and design tracking are not only appropriate but necessary for larger systems.
3.7 Integration
The degree of integration the software will have with other systems is an important consideration during development. Software embedded within larger hardware systems such as aircraft and tanks need to consider different interfaces for control and communication. Therefore, requirements and designs will need to include these considerations to understand how the final system as a whole will operate. Additionally, verification and validation phases further in the development process are need to ensure that the individual software components are integrated.

Proper integration requires careful thought and planning. As an example, the Universal Armament Interface (UAI) program was contracted out by the USAF in 2005 to address the challenges of integrating future precision-guided weapons on aircraft in a more timely and efficient manner without having to revise aircraft software for each new weapon. The UAI concept included defining an aircraft-to-store interface with a common digital message set for the data bus specified in a military standard compliant interface, determining a standard interface for the mission planning components, defining common coefficient based launch acceptability region equations, and determining a configuration file definition to configure the aircraft/store system interface [53], [54].

The requirements definition, documentation, and approval of a precision guided munition interface on combat aircraft was conducted by a diverse group of stakeholders. The stakeholders were required to define and document the specific implementation of the interface based on the unique characteristics and capabilities of each aircraft. Each UAI compliant component was independently validated using certification tools involving simulations. UAI brought a level of interoperability far higher than previous standardization efforts. The standard eliminated the lengthy task of developing weapon-specific interface software that used to burden integration efforts, allowing the warfighter to use new smart-weapon capabilities sooner and at lower cost. One of the main contributors to UAIs success was that so much time and effort was spent on defining requirements and analyzing different designs with a group of practitioners/experts before official standards were put into place. Clearly, it is appropriate to use development processes that involve thorough consideration of requirements and designs for software systems with high levels of integration within larger systems. Additionally, as the level of integration grows, so usually will the level of system complexity, which would require tailoring of processes to include more comprehensive documentation and change management.

Stand-alone software, however, may not require as much of a focus on mapping out system dependencies and creating standardized communication formats. Instead, processes used to develop such software can have more of a focus on prioritizing other characteristics such as urgency, lifespan, and quality which would then require specific tailoring. Lower degrees of integration narrow the focus on the specific software system with less need to consider the overall system and how it should operate within the operational environment.
3.8 Requirements
Last but not least, the requirements for specific software systems is probably one of the most important characteristics in tailoring development processes. Requirements influence other characteristics in varying degrees and have many nuances.

First, system requirements can be categorized as customized or general. For example, systems designed to address specific problems may need to be customized for different characteristics such as specialized users, unique environments, and certain performance or quality levels. However not all systems, even those designed for the military, need to be customized. It is possible for the military to adopt general requirements from the commercial sector. Baseline requirements for application security, data management, and networking can be adopted from commercial systems, especially if performance and risk management are not the main focus of development.

Second, requirements can be categorized as knowable, unknowable, or pretty well known but requires effort. For example, when problems are clear, users know exactly what they want, system lifespan is short, or performance and quality baselines are clearly outlined, the requirements are typically well known. However, not all systems being developed are clearly known. Depending on users and operational environment, along with the expected lifespan and future considerations for evolvability and integration, requirements may be unknowable based on the information available during system conception.

However, not all system requirements are categorized as one or the other. Systems can also have pretty well-known requirements but require effort to further refine. For example, the F-22 Raptor is a stealth fifth-generation tactical fighter aircraft; its development began in 1986 and initial production began in 2001. A total of 750 aircraft were planned to be bought, but in 2009 the program was terminated at 187 operational units [55]. The USAF established a modernization program in 2003, consisting of 10 separate programs, to provide multiple capabilities through hardware and software upgrades. However, the F-22 Program Office found that previous software upgrades resulted in finding and correcting deficiencies that required even more upgrades [56]. While the basic requirements for the fighter were well known, as the system’s lifespan increased and operational environment changed, users wanted to add more functionality while correcting previously undetected deficiencies. Refining new requirements based on changing contexts involves further effort and design considerations that need to be integrated within the development process.

Finally, related to the nuance of knowing requirements, definition is also an important consideration during development. Systems with well-defined requirements closed to changes, such as aircraft flight control software, can be developed with processes tailored to have longer cycle times and rigid documentation. On the opposite side of the spectrum, systems with flexible requirements that are open to changes based on user feedback, would be more appropriately developed with processes tailored to incorporate faster cycle frequencies where functionality can be evolved and expectations between users and developers can be continuously matched.
The nuances of each software system characteristic that can be used to tailor different development processes are outlined in the following table:

**Table 1. Characteristics of Software Systems**

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>User</th>
<th>Engineering</th>
<th>Business Manager</th>
<th>Casual</th>
</tr>
</thead>
<tbody>
<tr>
<td>User</td>
<td>Engineering</td>
<td>Business Manager</td>
<td>Casual</td>
<td></td>
</tr>
<tr>
<td>Urgency</td>
<td>Critical for existence</td>
<td>Important</td>
<td>Minor importance</td>
<td></td>
</tr>
<tr>
<td>Lifespan</td>
<td>Long</td>
<td>Medium</td>
<td>Short</td>
<td></td>
</tr>
<tr>
<td>Performance (timing)</td>
<td>Hard real-time deadlines</td>
<td>Soft real-time deadlines</td>
<td>No real-time aspects except annoyances</td>
<td></td>
</tr>
<tr>
<td>Quality/Risk</td>
<td>Safety-critical, security-critical, regulated</td>
<td>Prioritize large market share and speed to market</td>
<td>Large latitude allowed</td>
<td></td>
</tr>
<tr>
<td>Size</td>
<td>Small (&lt;100K SLOC)</td>
<td>Medium (100K-1M)</td>
<td>Large (&gt;1M)</td>
<td></td>
</tr>
<tr>
<td>Integration</td>
<td>Embedded within larger hardware system</td>
<td>Stand-alone software</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Requirements</td>
<td>Customized</td>
<td>General</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Knowable</td>
<td>Unknowable</td>
<td>Pretty well-known but requires effort</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Well-defined, closed system</td>
<td>Flexible, open system</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
4. Policy Recommendations

Current DoD and USAF acquisition policy lacks guidance regarding tailoring the new Software Acquisition Pathway introduced in the updated DoD 500.02 as part of the Adaptive Acquisition Framework. While previous acquisition policy such as the MIL-STD-498 “Software Development and Documentation” included multiple development processes as program strategy options, the new Software Acquisition Pathway as described in the DoD 5000.02 is not sufficient for developing software systems with different characteristics. The military operates in unique environments where mission assurance is a key consideration; ensuring that the required quality and evolvability of military systems do not suffer is of utmost importance.

Based on research of existing policy and software development methodologies, tailoring development processes to fit certain software system characteristics is the most applicable approach for the DoD’s need to develop specialized systems. While the DoDI 5000.02 and Defense Acquisition Guidebook do include that system acquisition strategies need to be tailored with regards to complexity, risk, and urgency, the current available guidance is more focused on management and legal oversight rather than tailoring engineering process models required for developing systems. The burden of responsibility for tailoring is placed on the acquisition chain of command, with the MDAs/DAs and Program Executive Office at the very top and the individual Program Managers at the lower level. Without clear tailoring guidance for how software system characteristics should be accounted for and exactly how development methodologies can be tailored, programs will remain inconsistent with each other as PMs with varying levels of software development experience under different types of organizational influences continue to subjectively tailor their programs. Additionally, there is a risk that managers will choose to develop software systems under the single Software Acquisition Pathway to avoid having to go through the tailoring process entirely. Specific DoD or service level policy/guidance regarding how to tailor development processes according to the characteristics of software systems is not only helpful to acquisition practitioners who already have so many responsibilities; they will help ensure that a certain level of rigor expected for military system development is maintained.

The Office of the Secretary of Defense (OSD), the Office of the Under Secretary of Defense for Acquisition & Sustainment, and the Office of the Secretary of the Air Force for Acquisition, Technology & Logistics, along with the associated acquisition offices for other military services should set clear tailorability guidelines based on the unique characteristics of software systems for programs involving software to implement. The USAF needs to emphasize within its acquisition policy that software is not only different from hardware; the characteristics of software can differ from system to system. The specific software system characteristics of: user, urgency, lifespan, performance (timing), quality/risk, size, integration, and requirements need to be fully addressed in order to tailor the most appropriate development processes. All PMs should be required to analyze the software systems they are developing according to these eight characteristics. The new policy and guidelines should include enough information to help PMs make the appropriate process design decisions based on the nuances of each system.
characteristic listed in Table 1. Tailored development processes must include the most appropriate design traits that specifically address the characteristics of the software system.

Justifications for how the program’s development process is appropriately tailored to address each characteristic should be included within the Acquisition Strategy document required by the Software Acquisition Pathway. The justifications should be passed along the acquisition chain of command for approval. Additionally, feedback regarding specific tailoring choices should be included for consideration in the same document and reviewed by the appropriate decision authorities.

Once new tailorability-focused policy and guidelines are established with stakeholder input from the OSD and the acquisition offices under it, they should be made widely available to acquisition practitioners in the form of a standalone guide or inclusion into Chapter 1 (Program Management) or Chapter 3 (Systems Engineering) of the Defense Acquisition Guidebook. If DoD-wide policy is not established, the USAF must take charge in setting new tailorability-focused regulation and ensure that new developments are publicized widely. There should not be a need to scour outdated standards to find the most applicable and active policy for each military service. Personal experience also shows that finding related policies and guidance, such as service-specific implementation guidelines of NDAA provisions using publicly available resources, is difficult. Additionally, policy is not always updated or current. Outdated policies are usually not updated to reference most current versions, and current policies could not all be found at a consistent location. Memorandums are scattered, and multiple sources have to be accessed to compile documents that provide a complete picture of recent regulatory changes. Policy changes should be consolidated in regularly released announcements that document all progress and specific measures being put into place, along with links to relevant policies. Additionally, outdated policies should be removed or archived.

4.1 Strengths
This policy recommendation outlines eight specific software system characteristics and the associated nuances that should be addressed when designing the appropriate development process. Specific guidance will help guide PMs to ensure consistency across programs and due diligence in program evaluation. Overall, it will make the process of tailoring less subjective, and shift focus from management and legal oversight to more detailed technical guidance that draws on good systems engineering principles. The recommended policy will preserve the flexibility needed to tailor appropriate development processes while implementing specific steps for ensuring that the performance and quality needs of the system being developed are considered from the very beginning.

Furthermore, set guidance using systems engineering principles will allow the USAF to evaluate development processes more objectively. Program offices and PMs will be less susceptible to sales pitches and advertisements from the commercial sector. Potential implementations of new process designs will first have to prove that they sufficiently address the eight software system characteristics before being deemed appropriate for the software development effort.
4.2 Weaknesses
As seen in the previous chapter, the characteristics of software systems are nuanced and the most appropriate development processes are not clear cut. The characteristics of software differ from system to system and are on a non-binary continuum. Tailoring policies/guidelines will not be beneficial if the practitioners responsible for tailoring fail to do so correctly. The military has trouble retaining experience as members are not in the same job positions for very long compared to their civilian counterparts. Additionally, members may be assigned to fill job positions with responsibilities they were not trained specifically for. For example, an aeronautical engineer with limited software engineering experience could be assigned as a program manager for a purely digital system.

Decision makers for process tailoring must understand the considerations needed to develop military software. They need a baseline understanding of software development processes and how to match system characteristics with the appropriate processes. Inexperience may also lead to inappropriate considerations for the uniqueness of military systems. Manager may inaccurately think that commercial software systems and development practices can be implemented wholesale for the military.

4.3 Opportunities
The DoD as a whole is currently focused on improving software acquisition processes. As a result, the community is open to suggestions on how to develop software more effectively. This policy recommendation is a great opportunity to provide solid and specific tailoring guidance on improving software development in the defense sector. With the USAF leading the implementation of tailorability guidance, the recommended policy can serve as a baseline for other military services.

For more technical details on software development processes, the USAF should draw on the wealth of resources it has access to such as its wide portfolio of completed development projects and the experiences of software developers and program managers that have executed development processes on a daily basis for years. More stakeholder involvement and detailed look at historical development projects will expand the organization’s internal expertise. A focus on learning from the past will incentivize better documentation of development efforts, extract lessons from both successful and failed programs, and provide an opportunity to improve future developments.

4.4 Threats
To prevent incorrect execution of the recommended policy, the guidelines must achieve a balance of specificity. The decision authorities should have enough detail to prevent being purely subjective in their decision making, but not be so limited by policy that they would want to avoid the tailoring process altogether. The level of authority given to PMs should be just enough for them to tailor processes most appropriately to the systems they are responsible for developing but can still stay within the bounds of the guidance provided.
There is also a threat that with the increased focus on tailoring processes, practitioners could become attracted to new innovations within the commercial sector and be lured by advertisements of new development processes that are not meant for specialized military systems. The policy and guidelines implemented should have clear and easy to understand evaluation criteria for potential development processes based on the eight system characteristics to prevent such a situation. A detailed process with a certain level of rigor must be established.

Finally, promoting the appropriate cultural changes will continue to be a challenge for new policies and guidelines. Some organizations will be unwilling to change their routine practices, and will project the image that they are trying to make changes while having no intention to implement them properly. For example, an organization focused on delivering software capabilities quickly may produce the required documentation and justifications for how the development process they used was appropriate, but in reality gloss over the important considerations necessary for critical military systems. Additionally, there is the possibility that organizations that want to make the correct cultural changes are not sure exactly how to or do not know if they are doing so properly. Therefore, it is critical to establish tailorability policy and guidelines that achieve the appropriate balance of specificity and implement checks to ensure the new policy is being adhered to correctly.
5. Conclusion

The DoD is currently focused on accelerating defense acquisition processes as a result of pressures from near-peer adversaries to innovate and be able to quickly leverage digital capabilities. To current leaders, the biggest risks from the acquisition system is delivering capabilities too late or not delivering at all to the field. However, with the current focus on speed and agility in developing software, it is important to recognize that military systems have certain levels of performance and quality necessary for mission assurance. Prioritizing speed/cycle time metrics for digital systems may not be adequate if incorrect software systems that impact mission performance are fielded. Even if changes can be made quickly, incorrect changes may further increase risks as additional cost and time are required to fix mistakes. Ultimately, fielding incorrect software too quickly could lead to detrimental impacts on mission success.

Defense acquisition in general is a mix of both technology and policy; complex systems are developed by people using organizational processes and legal frameworks established by Congress and military services to achieve national security goals outlined by the President. Recognizing that traditional defense acquisition was too rigid and did not have processes tailored to the characteristics of different systems, the Office of the Secretary of Defense for Acquisition and Sustainment rewrote the DoDI 5000.02 and introduced the DoD Adaptive Acquisition Framework, in which the Software Acquisition Pathway was introduced. Although the currently available policy states that the new pathways in the framework are tailorable, existing tailorability guidance is focused on management and legal oversight. Examples of customized acquisition processes are focused on showing how different acquisition pathways can be combined, and there is no guidance on how the specific engineering processes with the software pathway itself can be tailored. Additionally, tailoring responsibilities are placed on individuals within the acquisition chain of command, especially Program Managers who do not have specific guidance and must also balance other program management responsibilities. Therefore, there is a need to establish acquisition policies and guidelines regarding tailoring development processes to fit the characteristics of software systems.

To determine the specific characteristics of software systems that processes should be tailored to, I first defined the six processes within the Software Development Life Cycle: Requirements Analysis, Design, Implementation/Coding, Testing, Deployment, and Maintenance to give some context on how software development efforts are generally addressed. I then described four software development methodologies: Waterfall, Incremental, Spiral, and Agile to show that software can be developed with different process models which provide different ordering of technical activities to tradeoff the cost, schedule, and performance considerations of systems engineering and program management. Furthermore, describing the four process methodologies showed that certain processes are more appropriate for developing software systems with specific characteristics. As a result, I argue that tailoring processes to fit certain system characteristics is the most applicable approach for the DoD’s need to develop specialized systems.
The main software system characteristics that different development processes should be tailored to address are: user, urgency, lifespan, performance (timing), quality/risk, size, integration, and requirements. There are several nuances within each system characteristic. The end-user of system can use the software for engineering, business management, or casual purposes. The urgency to deploy working software can be critical for existence, important, or of minor importance. The expected lifespan of a software system can range from long, medium, to short. The required software performance related to timing can have hard or soft real-time deadlines, or no real-time aspects at all. The levels of quality that need to be met and levels of risk that need to be managed can range from safety/security-critical where risk must be managed, to where quality is necessary to capture a large market share but speed of delivery is a priority, down to where there is a large latitude allowed. The size of software systems, measured in source lines of code, can be categorized into small (under 100 thousand), medium (ranging from 100 thousand to 1 million), and large (over 1 million). The degree of integration the software will have with other systems can range from software embedded within larger hardware systems to stand-alone software. Finally, system requirements can vary on multiple levels. Requirements can be generalized or customized for specialized uses. Additionally, requirements can be categorized as knowable, unknowable, or pretty well known but requiring effort to determine. Lastly, system requirements can be well-defined and closed to changes, or on the opposite side of the spectrum, be flexible and open to changes. Envisioned systems can fit into any number of combinations of these characteristics. Therefore, it is important to design the most appropriate development processes by tailoring based on system characteristics.

It is recommended that the acquisition components of the DoD, USAF, and other military services establish clear detailed policy and guidelines regarding tailoring development processes to fit the eight software system characteristics identified. Acquisition practitioners need to understand the nuances involved within each characteristic and sufficiently determine how their envisioned systems fit according to the characteristics outlined. Only then can development processes and plans be tailored appropriately. Instead of introducing new processes and using trial and error to see if they are effective, it is more beneficial to use known software engineering and system engineering principles with encouraging innovation in ways that required quality and evolvability do not suffer. Implementing tailorability of engineering processes into the software acquisition pathway so that development efforts can better fit software systems being developed is not only beneficial for the acquisition process, it is necessary.

Clear detailed policy and guidelines for tailoring development processes will help ensure consistency throughout programs and maintain a certain level of due diligence at the very beginning of development when processes are designed. Subjectivity in design will be decreased, and practitioners will be less susceptible to adopting new commercial practices wholesale without adequate evaluation to justify such adoption is appropriate. Additionally, the recommended policy will provide an opportunity to improve the USAF’s internal expertise on software development, promote more comprehensive documentation of development programs, and use lessons learned from the past to inform future development efforts.

To reduce the negative externalities that may arise with implementing the recommendations, such as incorrect or unwilling adoption of new tailoring guidelines, it is critical for the policy to
achieve a balance of specificity to show enough detail on how tailoring is supposed to be done, but not be too restrictive that the guidance is seen as ‘more bureaucracy’ that is time consuming and not worth the effort. Additionally, clear policy and guidance will allow stakeholders to provide feedback at various points along the program approval process and establish checkpoints for ensuring that the proper actions are correctly executed.

7.1 Limitation of Research
Specific policies are still currently being developed with regards to different acquisition pathways and how exactly they are to be executed at the service-level. As mentioned in the beginning, the FY2020 NDAA establishes two new software acquisition specific pathways in Section 801. It is possible that policies and guidance released after the submission of this document could change many of the background details in the policy-oriented sections of my thesis. However, this does not change the need to consider the eight software system characteristics outlined in the technical components of this thesis.

As emphasized repeatedly, the characteristics of software can vary significantly from system to system. Therefore, it is complicated and not practical to establish guidelines that state exactly what development process is most appropriate for a specific software system. Guidance to the level of granularity where practitioners follow a flow chart that tells them exactly what development process they need to use will be inaccurate. For some systems, certain characteristics may conflict with each other and some may not even apply, which would require program managers to make tradeoffs that they deem the most appropriate. Additionally, too little detail within the guidance will result in inconsistencies from system to system and more subjective than objective evaluation. How to achieve the appropriate balance of specificity in the recommended policy is not known and not within the scope of this thesis.

7.2 Future Work
Given more time, it would be interesting to examine in detail the recent DoD-wide policies currently regarding different software development methodologies in Section 801 of the FY2020 NDAA and contact those responsible for creating that policy to obtain first-hand insight into how tailorable is considered and addressed within DoD-level guidance.

Additionally, working with acquisition practitioners at the military service and program office levels would be useful for gaining insight into how DoD policies are interpreted and executed at lower levels of management. This would help in understanding the appropriate level of specificity necessary within policy and guidelines for effective adoption within different organizations.
6. References


[9] 114th Congress, FY2016 NDAA Section 804 “Middle Tier of Acquisition for Rapid Prototyping and Rapid Fielding.”


