Software Is Never Done

Refactoring the Acquisition Code for Competitive Advantage

Defense Innovation Board

May 3, 2019

Cover design by the Office of Secretary of Defense, Graphics and Presentations Division

Software Is Never Done: Refactoring the Acquisition Code for Competitive Advantage

Defense Innovation Board, 3 May 2019

J. Michael McQuade and Richard M. Murray (co-chairs)
Gilman Louie, Milo Medin, Jennifer Pahlka, Trae' Stephens

DIB | DEFENSE INNOVATION BOARD

5 April 2019

I am pleased to forward the final report of the Software Acquisition and Practices (SWAP) study conducted by the Defense Innovation Board (DIB). The study, co-chaired by Dr. Richard Murray and Dr. Michael McQuade, was executed pursuant to Section 872 of the 2018 National Defense Authorization Act (NDAA). It makes 10 primary recommendations and 16 additional recommendations to address the most critical statutory, regulatory, and cultural hurdles DoD faces in modernizing its approach to software. In a year replete with reports on artificial intelligence, quantum computing, and blockchain, it may seem mundane to write about software, but software is the foundation of all things digital, and the Department does it exceedingly poorly. It is scarcely possible to imagine how the Department can achieve the modernization objectives of the National Defense Strategy without overhauling its approach to software, which is what motivated us to take on this task.

I have questioned the usefulness of the myriad studies and reports on technology authorized by Congress and DoD over the years. These mandates are often executed in a vacuum and the recommendations rarely seen through to implementation. Consequently, we resolved early on to conduct the SWAP study differently, with emphasis on pragmatism, accessibility, and candor. The iterative and inclusive approach the SWAP study team opted to take gave Congress, industry partners, and Department stakeholders at all levels unprecedented opportunity to participate in and contribute to the vision for the future of DoD software. Most importantly, each one of these communities provided critical feedback to ensure the recommendations contained in this report are timely, actionable, and contextualized by the ultimate mission -- swiftly delivering capability to the warfighter. Fittingly, it seems the recipe for delivering value in reports is the same as in modern software development: user feedback. Ultimately, we produced a report that changed my mind about the usefulness of such reports, and I hope you agree.

I hope the draft implementation plans at Appendix A are particularly valuable; we intended to provide sufficient detail to make them immediately implementable, but not so much as to be inflexible. They should provide the Department and Congress a starting point from which to act on the recommendations and instill critical accountability across the organization and in industry.

I recommend that this report be distributed to the offices within OSD and the Services that deal with software technology and acquisition directly, and to the appropriate industries and organizations that support them, as well as to numerous organizations that have seen their mission transformed by software, however reluctant they may appear to be to acknowledge it. With regard to industry and private sector partners, we deliberately wrote for the defense industrial base as well as for those companies in the national security innovation base that may not identify themselves as defense suppliers, but whose support will be essential for the Department's future technological relevance. Pay mind to the report's first theme: software *truly* is ubiquitous; and a modern acquisition and development approach should be architected at the enterprise-level with broad support and understanding from the "users."

Finally, I fully support the DIB's further involvement, at the discretion of relevant Department leadership, as advisors in the implementation of the recommendations contained in this report. We consider it our privilege to have engaged with such talented and devoted uniformed personnel, civilians, and contractors over the course of the SWAP study and we welcome a continued dialogue about how to make the ideas contained in this report a reality.

Sincerely,

[illegible]

Dr. Eric Schmidt
Chairman
Defense Innovation Board

DIB | DEFENSE INNOVATION BOARD

MEMORANDUM TO THE CHAIRMAN, DEFENSE INNOVATION BOARD

SUBJECT: Final Report of the Defense Innovation Board (DIB) Software Acquisition and Practices (SWAP) Study

Attached is the final report from the DIB's SWAP study, documenting our efforts, analysis and conclusions working with Congress and DoD on how to develop, procure, assure, deploy, and continuously improve software for use in the Department. In developing this report over the past eighteen months, we have had substantial conversations with congressional staffers, DoD leadership at many levels, program offices, contractors, (government and private sector) software developers, and a variety of other representatives from government, industry, academia, and the public. There is broad consensus on the goal of delivering high quality software to DoD users in a manner that is timely, secure, and cost effective. Our study details the external and self-inflicted barriers DoD faces in implementing modern software practices and lays out steps to address current gaps. We hope that our recommendations will serve as a basis for the implementation phase of this work. We are happy and ready to support that effort.

We would like to thank the study members, Gilman Louie, Milo Medin, Jennifer Pahlka and Trae' Stephens for their contributions to the report. Bess Dopkeen served as the initial study director and established an outstanding structure for the study and the support staff. Jeff Boleng supported the study throughout its initial phase and stepped in as study director after Bess. We would also like to acknowledge the outstanding help provided by Courtney Barno, Kevin Garrison, Nick Guertin, Devon Hardy, Sandra O'Dea, Forrest Shull, and Craig Ulsh. A longer list of the many people who have helped on the study is included in the Acknowledgements section of the main report and in Appendix J.

JMichael McQuade

J. Michael McQuade

[illegible]

Richard M. Murray

Software Is Never Done: Refactoring the Acquisition Code for Competitive Advantage

Defense Innovation Board, 3 May 2019

J. Michael McQuade and Richard M. Murray (co-chairs)
Gilman Louie, Milo Medin, Jennifer Pahlka, Trae' Stephens

Extended Abstract

U.S. national security increasingly relies on software to execute missions, integrate and collaborate with allies, and manage the defense enterprise. The ability to develop, procure, assure, deploy, and continuously improve software is thus central to national defense. At the same time, the threats that the United States faces are changing at an ever-increasing pace, and the Department of Defense's (DoD's) ability to adapt and respond is now determined by its ability to develop and deploy software to the field rapidly. The current approach to software development is broken and is a leading source of risk to DoD: it takes too long, is too expensive, and exposes warfighters to unacceptable risk by delaying their access to tools they need to ensure mission success. Instead, software should enable a more effective joint force, strengthen our ability to work with allies, and improve the business processes of the DoD enterprise.

Countless past studies have recognized the deficiencies in software acquisition and practices within DoD, but little seems to be changing. Rather than simply reprint the 1987 Defense Science Board (DSB) study on military software that pretty much said it all, the Defense Innovation Board's (DIB's) congressionally mandated study[1] on Software Acquisition and Practices (SWAP) has taken a different approach. By engaging Congress, DoD, Federally Funded Research and Development Centers (FFRDCs), contractors, and the public in an active and iterative conversation about how DoD can take advantage of the strength of the U.S. commercial software ecosystem, we hope to move past the myriad reports and recommendations that have so far resulted in little progress. Past experience suggests we should not anticipate that this report will miraculously result in solutions to every obstacle we have found, but we hope that the two-year conversation around it will provide the impetus for figuring out how to make the changes for which everyone is clamoring.

In this report, we emphasize three fundamental themes:

1. **Speed and cycle time are the most important metrics for managing software**. To maintain advantage, DoD needs to procure, deploy, and update software that works for its users at the speed of mission need, executing more quickly than our adversaries. Statutes, regulations, and cultural norms that get in the way of deploying software to the field quickly weaken our national security and expose our nation to risk.

2. **Software is made by people and for people, so digital talent matters**. DoD's current personnel processes and culture will not allow its military and civilian software capabilities to grow nearly fast or deep enough to meet its mission needs. New mechanisms are needed for attracting, educating, retaining, and promoting digital talent and for supporting the workforce to follow modern practices, including developing software hand in hand with users.

[1] 2018 National Defense Authorization Act (NDAA), Sec. 872. Defense Innovation Board analysis of software acquisition regulations.

3. **Software is different than hardware (and not all software is the same).** Hardware can be developed, procured, and maintained in a linear fashion. Software is an enduring capability that must be supported and continuously improved throughout its life cycle. DoD must streamline its acquisition process and transform its culture to enable effective delivery and oversight of multiple types of software-enabled systems, at scale, and at the speed of relevance.

To take advantage of the power of software, we advocate four main lines of effort:

A. **Congress and DoD should refactor statutes, regulations, and processes for software**, enabling rapid deployment and continuous improvement of software to the field and providing increased insight to reduce the risk of slow, costly, and overgrown programs.

B. **The Office of the Secretary of Defense (OSD) and the Services should create and maintain cross-program/cross-Service digital infrastructure** that enables rapid deployment, scaling, testing, and optimization of software as an enduring capability; manage them using modern development methods; and eliminate the existing hardware-centric regulations and other barriers.

C. **The Services and OSD will need to create new paths for digital talent (especially *internal* talent)** by establishing software development as a high-visibility, high-priority career track and increasing the level of understanding of modern software within the acquisition workforce.

D. **DoD and industry must change the practice of how software is procured and developed** by adopting modern software development approaches, prioritizing speed as the critical metric, ensuring cybersecurity is an integrated element of the entire software life cycle, and purchasing existing commercial software whenever possible.

Report structure. The main report provides an assessment of the current and desired states for software acquisition and practices, as well as a review of previous reports and an assessment of why little has changed in the way DoD acquires software, with emphasis on three fundamental themes. The report's recommendations are broken into four lines of effort, with a set of primary recommendations provided for each (bold), along with additional recommendations that can provide further improvements. Each recommendation is accompanied by a draft implementation plan and potential legislative language.

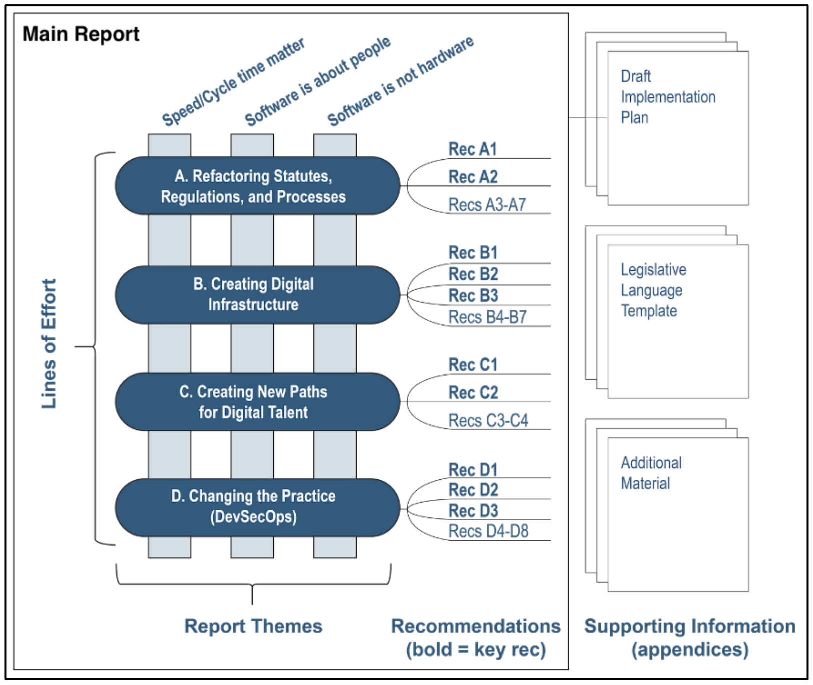

Main Report
Speed/Cycle time matter
Software is about people
Software is not hardware
A. Refactoring Statutes, Regulations, and Processes
B. Creating Digital Infrastructure
C. Creating New Paths for Digital Talent
D. Changing the Practice (DevSecOps)
Lines of Effort
Rec A1
Rec A2
Recs A3-A7
Rec B1
Rec B2
Rec B3
Recs B4-B7
Rec C1
Rec C2
Recs C3-C4
Rec D1
Rec D2
Rec D3
Recs D4-D8
Draft Implementation Plan
Legislative Language Template
Additional Material
Report Themes
Recommendations (bold = key rec)
Supporting Information (appendices)

Table of Contents

Supporting Information

Chapter 0. README (Executive Summary)

In 2011, Marc Andreessen claimed in an op-ed for *The Wall Street Journal* that "Software Is Eating the World."[2] He argued that *every* industry (not just those considered to be "information technology") would be transformed by software—bytes rather than atoms. Eight years later, it is clear he was right.

This transformation is happening in defense, and we are not prepared for it. Software is leveling the playing field with our rivals, eroding the advantages we have spent many decades accruing. Software is the focal point of many important advances in national security technology, including data analytics, artificial intelligence (AI), machine learning (ML), and autonomy. Software is ubiquitous. It is part of everything the Department of Defense (DoD) does, from logistics to management to weapon systems. U.S. national security is critically dependent on the capabilities of DoD's software.

DoD must be able to develop, procure, assure, deploy, and continuously improve software faster than our adversaries. Unfortunately, DoD still treats software much like hardware, and often misunderstands the relationship between speed and security. As a result, a large amount of DoD's software takes too long, costs too much, and is too brittle to be competitive in the long run. If DoD does not take steps to modernize its software acquisition and development practices, we will no longer have the best military in the world, no matter how much we invest or how talented and dedicated our armed forces may be.

The good news is that there are organizations within DoD that have already acknowledged the risks of falling further behind in software and are leveraging more modern acquisition and development practices with notable success. The Defense Digital Service (DDS), the Defense Innovation Unit (DIU), the Joint Improvised-Threat Defeat Organization (JIDO), and the Air Force's Kessel Run are examples that demonstrate that DoD has the ability to ship world-class software. The challenge remains doing this at scale.

DoD needs to build on these foundations to create an ecosystem and standard operating procedures that enable the practices of great software without requiring employees to "hack the system." To do that, we must address the prioritization, planning, and acquisition processes and policies that create the worst bottlenecks for deploying capability to the field at the speed of relevance. Further, we must address all the practices that not only put the U.S. Armed Forces at risk and reduce the efficiency of DoD's operations, but also drive away the very people who are most needed to develop this critical capability.

Our adversaries are already doing this. China actively leverages its private industry to develop national security software (particularly in AI), recruits top students under the age of 18 to work on "intelligent weapons design,"[3] and poaches U.S. software talent directly from the United States. In Russia, Vladimir Putin has told students, that "artificial intelligence is the future, not only for Russia, but for all humankind.... Whoever becomes the leader in this sphere will become the ruler

[2] Marc Andreessen, "Why Software Is Eating the World," *The Wall Street Journal*, August 20, 2011, 1.

[3] Stephen Chen, "China's Brightest Children Are Being Recruited to Develop AI 'Killer Bots,'" *South China Morning Post*, November 8, 2018.

of the world."[4] We can and must outcompete with software and the people who make it, not only to maintain U.S. military superiority but also to ensure that the power that software represents is used in accordance with American values.

What this report is about. This report summarizes the assessment of the Defense Innovation Board's (DIB's) Software Acquisition and Practices (SWAP) study. Congress charged[5] the DIB to recommend changes to statutes, regulations, processes, and culture to enable the better use of software in DoD. We took an iterative approach, mirroring the way modern software is successfully done, releasing a sequence of concept papers describing our preliminary observations and insights. (The latest versions of these are included in Appendix E.) We used those papers to encourage dialogue with a wide variety of individuals and groups to gain insights into the current barriers to implementing modern software effectively and efficiently. This document captures key insights from these discussions in an easy-to-read format that highlights the elements that we consider critical for DoD's success and serves as a starting point for continued discussions required to implement the changes that we recommend here.

This report is organized as follows:

- **Extended Abstract:** A two-page summary of the key takeaways from the report.
- **README** (this document)**:** A more detailed executive summary of the report. (A README file is used by the open source software community to provide essential information about a software package.) If your boss heard about the report or read the extended abstract, thought it was intriguing, and asked you to read the entire report and provide a short summary, cut and paste this chapter into your reply and you should be good to go.
- **Recommendations Cheat Sheet:** A list of the main lines of effort and primary recommendations, so you can pretty much stop at that point—or better yet, stop after suggesting to your boss they adopt them all.
- **Chapters 1–4:** Short descriptions of key areas and topics. If you attach the extended abstract to any one of these as a preface, it should be comprehensible.
- **Chapter 5:** A more detailed description of the recommendations and our rationale.
- **Supporting Information:** To ensure that the executive summary and the main body of the report satisfy the takeoff test[6] and the staple test,[7] we put most of the additional information generated during the study into a set of appendices. These provide a wealth of examples and

[4] James Vincent, "Putin Says the Nation that Leads in AI 'will be the ruler of the world,'" *The Verge*, September 4, 2017: https://www.theverge.com/2017/9/4/16251226/russia-ai-putin-rule-the-world.
[5] Section 872 of the FY18 NDAA directed the Secretary of Defense to "direct the Defense Innovation Board to undertake a study on streamlining software development and acquisition regulations." The DIB-SWAP members were charged to "review the acquisitions regulations applicable to, and organizational structures within, the Department of Defense…; review ongoing software development and acquisition programs…; produce specific and detailed recommendations…; and produce such additional recommendations for legislation." See Section 872 of the FY18 NDAA at https://www.congress.gov/115/plaws/publ91/PLAW-115publ91.pdf or Appendix J of this report.
[6] Reports should be short enough to read during takeoff, before the movies start and drinks are served.
[7] Any report that is going to be read should be thin enough to be stapled with a regular office stapler.

evidence, but we took care to put our essential arguments up front for less wonky types. Some highlights:

- **Draft implementation** (Appendix A)**:** For each recommendation, a summary of the background, desired state, stakeholders, role of Congress, and actions to be taken.
- **Legislative language** (Appendix B)**:** In response to 2016 NDAA Section 805, template legislative language for a new acquisition pathway and appropriation category for software, aligned with our recommendations.
- **An alternative to P-Forms and R-Forms** (Appendix C)**:** A different mechanism for budget submissions for software programs.
- **FAQs** (frequently asked questions, Appendix D)**:** A list of the most common questions that we get about the study and our attempt to answer them. (Question 1: Hasn't all of this been recommended before? A: Yes…).

Note: If you are reading any portion of the report in paper form, a navigable version is available at http://innovation.defense.gov/software.

Overarching themes. The rise of electronics, computing, and networking has forever transformed the way we live: software is a part of almost everything with which we interact in our daily lives, either directly through embedded computation in the objects around us or indirectly through the use of information technology through all stages of design, development, deployment, and operations. Our military advantage, coordination with allies and partners, operational security, and many other aspects of DoD activities are all contingent upon our software edge, and any lack thereof presents serious consequences. Software drives our military advantage: what makes weapon systems sophisticated is the software, not (just) the hardware.

Commercial trends show what is possible with software, from the use of open source tools to agile development techniques to global-scale cloud computing. Because of these changes, software can be developed, deployed, and updated much more quickly, which means systems need to be in place to support this speed. But modern software development requires a new set of skills and methodologies (e.g., generalist software engineers, specialized product management, DevOps and DevSecOps, agile development). Hence, the policies and systems surrounding software must be transformed to support software, not Cold-War-era weapon manufacturing.

The incoming generation of military and civilian personnel began life digitally plugged-in, with an innate reliance on software-based systems. They will demand new concepts of operations, tactics, and strategies to maintain the edge they need. If DoD can refactor its acquisition processes and transform its culture and personnel policies before it is too late, this software-savvy generation can still set the Department on the right course.

As we studied the methods that the private sector has used to enable software to transform its operations and considered how to best apply those practices to the defense enterprise, three overarching themes emerged as the basis for our recommendations:

1. Speed and cycle time are the most important metrics for software.

2. Software is made by people and for people, so digital talent matters.
3. Software is different than hardware (and not all software is the same).

Speed and cycle time are the most important metrics for software. Most DoD software projects are currently managed using "waterfall" development processes, which involve spending years on developing requirements, taking bids and selecting contractors, and then executing programs that must meet the listed requirements before they are "done." This results in software that takes so long to reach the field that it is often not well matched to the current needs of the user or tactics of our adversaries, which have often changed significantly while the software was being written, tested, and accepted. Being able to develop and deploy faster than our adversaries means that we can provide more advanced capabilities, respond to our adversaries' moves, and be more responsive to our end users. Faster reduces risk because it demands focus on the critical functionality rather than over-specification or bloated requirements. It also means we can identify trouble earlier and take faster corrective action, which reduces cost, time, and risk. Faster leads to increased reliability: the more quickly software/code is in the hands of users, the more quickly feedback can focus on efforts to deploy greater capability. Faster gives us a tactical advantage on the battlefield by allowing operation and response inside our adversaries' observe–orient–decide–act (OODA) loops. Faster is more secure. Faster is possible.

Software is made by people and for people, so digital talent matters. Current DoD human resource policies are not conducive to attracting, retaining, and promoting digital talent. Talented software developers and acquisition personnel with software experience are often put in jobs that do not allow them to make use of those talents, particularly in the military where rotating job assignments may not recognize and reward the importance of software development experience. As Steve Jobs observed,[8] one of the major differences between hardware and software is that for hardware the "dynamic range" (ratio between the best in class and average performance) is, at most, 2:1. But, the difference between the best software developer and an average software developer can be 50:1, or even 100:1, and putting great developers on a team with other great developers amplifies this effect. Today, in DoD and the industrial base that supports it, the people with the necessary skills exist, but instead of taking advantage of their skills we put them in environments where it is difficult for them to be effective. DoD does not take advantage of already existing military and civilian personnel expertise by offering pay bonuses, career paths that provide the ability to stay in their specialization, or access to early promotions. Skilled software engineers and the related specialties that are part of the overall software ecosystem need to be treated as a special force; the United States must harness their talent for the great benefits that it can provide.

Software is different than hardware (and not all software is the same). Over the years, Congress and DoD have established a sophisticated set of statutes, regulations, and instructions that govern the development, procurement, and sustainment of defense systems. This process evolved in the context of the Cold War, where major powers designed and built aircraft carriers, nuclear weapons, fighter jets, and submarines that were extremely expensive, lasted a very long time, and required tremendous access to capital and natural resources. Software, on the other hand,

[8] Steve Jobs, "Steve Jobs: The Lost Interview," interview by Robert X. Cringely for the 1995 PBS documentary, *Triumph of the Nerds*, released to limited theaters in 2012, video.

is something that can be mastered by a ragtag bunch of teenagers with very little money—and can be used to quickly destabilize world powers. Currently most parts of DoD develop, procure, and manage software like hardware, assuming that it is developed based on a fixed set of specifications, procured after it has been shown to comply with those specifications, "maintained" by block upgrades, and upgraded by replaying this entire procurement process linearly. But software development is fundamentally different than hardware development, and software should be developed, deployed, and continuously improved using much different cycle times, support infrastructure, and maintenance strategies. Testing and validation of software is also much different than for hardware, both in terms of the ability to automate but also in the potential vulnerabilities found in software that is not kept up to date. Software is never "done" and must be managed as an enduring capability that is treated differently than hardware.

Main lines of effort. DoD's current approach to software is a major driver of cost and schedule overruns for Major Defense Acquisition Programs (MDAPs). Congress and DoD need to come together to fix the acquisition system for software because it is a primary source of its acquisition headaches.

Bringing about the type of change that is required to give DoD the software capabilities it needs is going to take a significant amount of work. While it is possible to use the current acquisition system and DoD processes to develop, procure, assure, deploy, and continuously improve DoD software, the statutes, regulations, processes, and culture are debilitating. The current approach to acquisition was defined in a different era, for different purposes, and only works for software projects through enormous effort and creativity. Congress, the Office of the Secretary of Defense (OSD), the Armed Services, defense contractors, and the myriad government and industry organizations involved in getting software out the door need to make major changes (together).

To better organize our specific recommendations, we identified broad lines of effort that bring together different parts of the defense ecosystem as stakeholders. Here are the four main lines of effort that we recommend they undertake:

A. **(Congress and DoD) Refactor statutes, regulations, and processes for software,** enabling rapid deployment and continuous improvement of software to the field and providing increased insight to reduce the risk of slow, costly, and overgrown programs. The management and oversight of software development and acquisition must focus on different measures and adopt a quicker cadence.

B. **(OSD and the Services) Create and maintain cross-program/cross-Service digital infrastructure** that enables rapid deployment, scaling, testing, and optimization of software as an enduring capability; manage it using modern development methods; and eliminate the existing hardware-centric regulations and other barriers.

C. **(The Services and OSD) Create new paths for digital talent (especially internal talent)** by establishing software development as a high-visibility, high-priority career track—with specialized recruiting, education, promotion, organization, incentives, and salary—and increasing the level of understanding of modern software within the acquisition workforce.

D. **(DoD and industry) Change the practice of how software is procured and developed** by adopting modern software development approaches, prioritizing speed as the critical metric, ensuring cyber protection is an integrated element of the entire software life cycle, and purchasing existing commercial software whenever possible.

None of these can be done by a single organization within the government. They will require a bunch of hard-working, well-meaning people to work together to craft a set of statutes, regulations, processes, and (most importantly) a culture that recognizes the importance of software, the need for speed and agility (theme 1), the critical role that smart people have to play in the process (theme 2), and the impact of inefficiencies of the current approach (theme 3). In many ways this mission is as challenging as any combat mission: while participants' lives may not be directly at risk in defining, implementing, and communicating the needed changes to policy and culture, the lives of those who defend our nation ultimately depend on DoD's ability to redefine its approach to delivering combat-critical software to the field.

Refactor statutes, regulations, and processes, streamlined for software. Congress has created many workarounds to allow DoD to be agile in its development of new weapon systems, and DoD has used many of these to good effect. But the default statutes, regulations, and processes that are used for software too often rely on the traditional hardware mentality (repeat: software is different than hardware), and those practices do not take advantage of what is possible (or, frankly, necessary, given the threat environment) with modern software. We think that a combination of top-down and bottom-up pressure can break us out of the current state of affairs, and creating a new acquisition pathway that is tuned for software (of various types) will make a big difference. To this end, Congress and DoD should prototype and, after proving success, create mechanisms for ideation, appropriation, and deployment of software-driven solutions that take advantage of the unique features of software (versus hardware) development (start small, iterate quickly, terminate early) and provide purpose-fit methods of oversight. As an important aside, note that throughout this study our recommendations adhere to this guiding axiom—start small, iterate quickly—the same axiom that characterizes the best of modern software innovation cycles (see the "DIB Ten Commandments of Software" in Appendix E for more information about the DIB's guiding principles for software acquisition).

Create and maintain cross-program/cross-Service digital infrastructure. Current practice in DoD programs is that each individual program builds its own infrastructure for computing, development, testing, and deployment, and there is little ability to build richer development and testing capabilities that are possible by making use of common infrastructure. Instead, we need to create, scale, and optimize an enterprise-level architecture and supporting infrastructure that enables creation and initial fielding of software within six months and continuous delivery of improvements on a three-month cycle. This "digital infrastructure," common in commercial IT, is critical to enable rapid deployment at the speed (and scale) of relevance. In order to implement this recommendation, Congress and DoD leadership must figure out ways to incentivize the Services and defense contractors to build on a common set of tools (instead of inventing their own) *without* just requiring that everyone uses one DoD-wide (or even Service-wide) platform. Similarly, OSD will have to define non-exceptions-based alternatives to (or at least pathways through) Joint Capabilities Integration and Development System (JCIDS), Planning, Programing, Budget and Execution

(PPB&E), and Defense Federal Acquisition Regulation Supplement (DFARS)[9] that are optimized for software. The Director, Operational Test and Evaluation (DOT&E) will need new methods for OT&E that match the software's speed of relevance, and Cost Assessment and Program Evaluation (CAPE) will have to capture better data and leverage AI/ML as a tool for cost assessment and performance evaluation. Finally, the Services will need to identify, champion, and measure platform-based, software-intensive projects that increase software effectiveness, simplify interconnectivity among allies, and reform business practices. Subsequent chapters in our report provide specific recommendations on each of these areas.

Create new paths for digital talent (especially internal talent). The biggest enabler for great software is providing great people with the means to contribute to the national security mission. While the previous recommendations speak to providing the tools and infrastructure DoD technologists need to succeed, it is equally important that the Department's human capital strategies allow them to even do this work consistently in the first place. Driving the cultural transformation to support modern, cloud-based technology requires new types of skills and competencies, changing ratios of program managers to software engineers, moving from waterfall development to DevSecOps[10] development, and dealing with all of the change management that comes with it. This is not an easy task, but arguably one of the most important. While compensation is a major driver in attracting competitive talent, DoD must also make changes in the roles, methodologies, cultures, and other aspects of the transformation that industry is already undergoing and that the government must undergo as well.

Increasing developer talent is not the only workforce challenge. DoD must also change how the government manages its programs and contractors, which goes beyond just moving to DevSecOps development. The government must have experts well steeped in the software development process and architecture design to adequately manage both organic activities and contracted programs. They must have the skills to detect when contractors are going down the wrong path, choosing a bad implementation approach, or otherwise wasting government resources. This is perhaps the best argument for ensuring we have software development experience natively in the government, rather than relying primarily on external vendors; unless there are software-knowledgeable members on the core team, it is impossible to effectively monitor and manage outsourced projects. This is especially true with the movement to DevSecOps.

In implementing this change in the workforce, it is particularly important to provide new career paths for digital talent and enable the infrastructure and environment required to allow them to succeed. The current General Schedule (GS) system favors time in grade over talent. This simply will not work for software. The military promotion system has the same problem. As with sports, great teams make a huge difference and, in software, we need to make sure those teams have the tools they need to succeed and reward them appropriately—through recognition, opportunities for impact, career advancement, and pay. Advanced expertise in procurement, project management, evaluation and testing, and risk mitigation strategies will also be needed to create the types of elite teams that are necessary. A key element of success is finding ways to keep talented

[9] Common DoD acronyms are defined in Appendix I (Acronyms and Glossary).
[10] An iterative software development methodology that combines development, security, and operations as key elements in delivering useful capability to the user of the software. See Section 2.1 for details.

people in their roles (rather than transferring them out because it is the end of their assignment), and promoting people based on their abilities, not based on their years of service.

Change the practice of how software is procured and developed. The items above are where we think Congress and the Department should focus in terms of statutory, regulatory, and process changes. But a major element is also the need to change the *culture* around software within Congress, DoD, and the defense industrial base. We use the term "DevSecOps" as our label for the type of culture that is needed: iterative development that deploys secure applications and software into operations in a continuing (and continuous) fashion.

Numerous projects and groups have demonstrated the ability to implement DevSecOps within the existing acquisition system. But the organizations we previously mentioned—DDS, JIDO, DIU, and Kessel Run—are the exception rather than the rule, and the amount of effort required to initiate and sustain their activities is enormous. Instead, DoD should make legacy programs that use outdated techniques for developing software fight for existence (and in most cases replace them with new activities that embrace a DevSecOps approach).

Getting started now. The types of changes we are talking about will take years to bring to complete fruition. But it would be a mistake to spend two years figuring out what the answer should look like, spend another two years prototyping the solutions to make sure we are right, and then spend two to four more years implementing the changes in statutes, regulations, processes, and culture that are actually required. Let's call that approach the "hardware" approach. Software is different than hardware, and therefore the approach to implementing change for software should be different as well.

Indeed, most (if not all) of the changes we are recommending are not new and not impossible to make. The 1987 DSB Task Force on Military Software,[11] chaired by legendary computer scientist Fred Brooks, wrote an outstanding report that already articulated much of what we are saying here. And the software industry has already implemented and demonstrated the utility of the types of changes we envision. The problem appears to be in getting the military enterprise to adopt a software mindset and implement a DevSecOps approach in a system that was intended to make sure that things would not move too quickly.

DoD could address many of our issues by adopting existing best practices of the private sector for agile development, including making use of software as a service; taking advantage of modern (cloud) infrastructure, tools, computing, and shared libraries; and employing modern software logistics and support delivery systems for software maintenance, development, and updating (patching). We do not need to study these; we need to get going and implement them. Here is a proposed timeline for implementing the primary recommendations of this report, starting *now*:

- (Immediately): Define, within 60 days after delivery of this report to Congress, a detailed implementation plan and assign owners to begin each of the top recommendations.

[11] Defense Science Board Task Force, *Military Software* (Washington, DC: Office of the Under Secretary of Defense for Acquisition, September 1987), https://apps.dtic.mil/dtic/tr/fulltext/u2/a188561.pdf.

- FY19 (create): High-level endorsement of the vision we articulate here, and support for activities that are consistent with the desired end state (i.e., DevSecOps and enterprise-level architecture and infrastructure). Identify and launch programs to move out on the priority recommendations (start small, iterate quickly). If you are reading this and are in a position of leadership in your organization, pass this on to others with your seal of approval and a request for your team to develop two or three plans of action for how it can be applied in your domain. If someone comes to you with a proposal that aligns with the objectives we have outlined here, find a way to be on the front line of changing DoD to a "culture of yes."
- FY20 (deploy): Initial deployment of authorities, budgets, and processes for software acquisition and practices reform. Execute representative programs according to the lines of effort and recommendations in this report, implement now, measure results, and modify approaches. Implement this report in the way we implement modern software.
- FY21 (scale): Streamlined authorities, budgets, and processes enabling software acquisition and practices reform at scale. In this time frame, we need a new methodology to estimate as well as determine the value of software capability delivered (and not based on lines of code).
- FY22 (optimize): Conditions established so that all DoD software development projects transition (by choice) to software- enabled processes, with the talent and ecosystem in place for effective management and insight.

In the remainder of this report, we provide a rationale for the approach that we are advocating. Chapter 1 makes the case for why software is important to DoD, including a taxonomy of the different types of software that need to be considered (not all software is the same). In Chapter 2, we describe how software is developed in the private sector and what is required in terms of workforce, infrastructure, and culture. Chapter 3 is an attempt to summarize what has already been said by other studies and groups, why the situation has not changed, and how we think this study can potentially lead to a different outcome. Chapters 4 and 5 contain our recommendations for how to move forward. In Chapter 4, we present three alternative paths to consider: doing the best we can with the current system; streamlining statutes, regulations, and processes so that they are optimized for software (instead of hardware); and making more radical changes that create entirely new appropriation categories and acquisition pathways. Finally, Chapter 5 describes the path that we recommend be taken, broken out along the lines of effort described above, and with a set of 10 primary recommendations followed by 16 additional recommendations (a detailed draft implementation plan for implementing each is included in Appendix A).

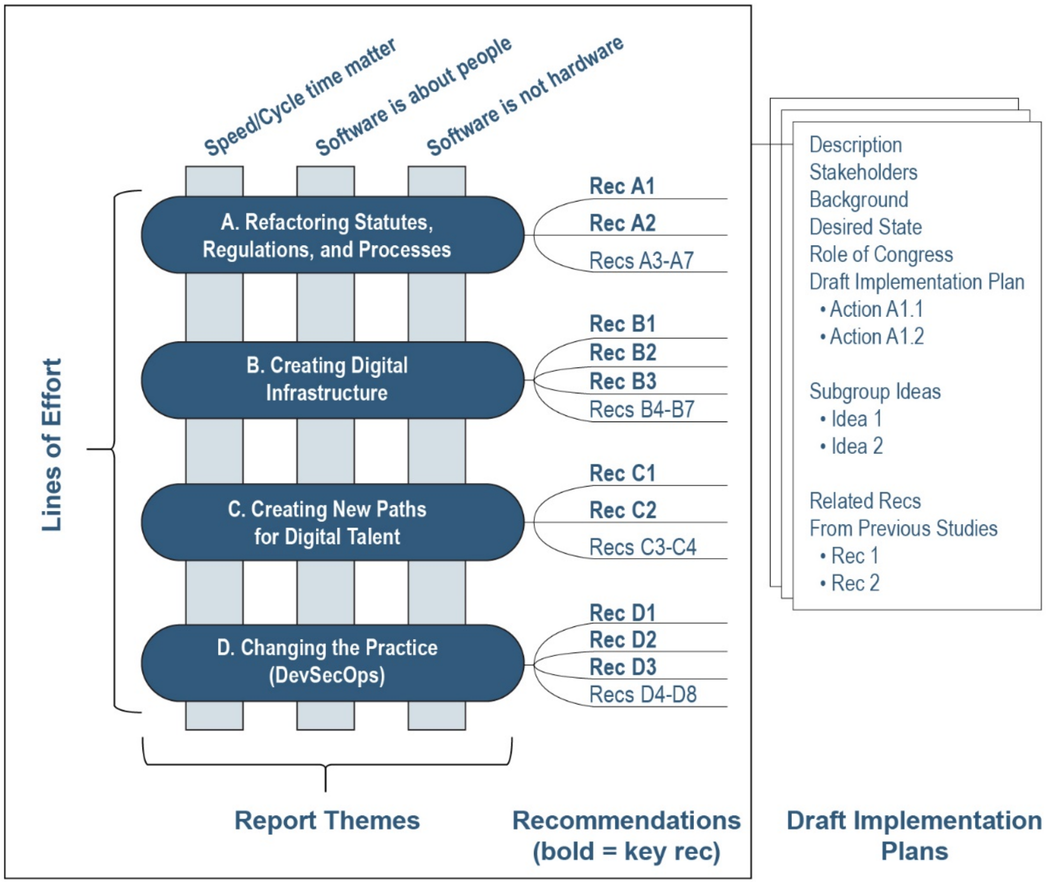
A two-page summary ("cheat sheet") of the lines of effort and recommendations follows.

DIB SWAP Study
Recommendations "Cheat Sheet"

This sheet contains a list of the recommendations from the Defense Innovation Board's (DIB's) Software Acquisition and Practices (SWAP) study. The recommendations below include input from the following sources:

- DIB Guides for Software (Appendix E)
- SWAP working group reports (Appendix F)
- Previous software acquisition reform studies (starting with the 1987 DSB study)

The recommendations are organized according to four major lines of effort and each recommendation contains background information, a proposed owner for implementing the recommendation, as well as a more detailed draft implementation plan, a list of other offices that are affected, and additional details. The following diagram documents this structure:



For each recommendation, a draft implementation plan can be found in Appendix A that gives more detail on the rationale, supporting information, similar recommendations, specific action items, and notes on implementation. Potential legislative language to implement selected recommendations is included in Appendix B.

The Ten Most Important Things to Do (Starting Now!)

Line of Effort A (Congress and OSD): Refactor statutes, regulations, and processes for software	
A1	Establish one or more new acquisition pathways for software that prioritize continuous integration and delivery of working software in a secure manner, with continuous oversight from automated analytics
A2	Create a new appropriation category for software capability delivery that allows (relevant types of) software to be funded as a single budget item, with no separation between RDT&E, production, and sustainment
Line of Effort B (OSD and Services): Create and maintain cross-program/cross-Service digital infrastructure	
B1	Establish and maintain digital infrastructure within each Service or Agency that enables rapid deployment of secure software to the field, and incentivize its use by contractors
B2	Create, implement, support, and use fully automatable approaches to testing and evaluation (T&E), including security, that allow high-confidence distribution of software to the field on an iterative basis
B3	Create a mechanism for Authorization to Operate (ATO) reciprocity within and between programs, Services, and other DoD agencies to enable sharing of software platforms, components, and infrastructure and rapid integration of capabilities across (hardware) platforms, (weapon) systems, and Services
Line of Effort C (Services and OSD): Create new paths for digital talent (especially *internal* talent)	
C1	Create software development units in each Service consisting of military and civilian personnel who develop and deploy software to the field using DevSecOps practices
C2	Expand the use of (specialized) training programs for CIOs, SAEs, PEOs, and PMs that provide (hands-on) insight into modern software development (e.g., Agile, DevOps, DevSecOps) and the authorities available to enable rapid acquisition of software
Line of Effort D (DoD and industry): Change the practice of how software is procured and developed	
D1	Require access to source code, software frameworks, and development toolchains—with appropriate IP rights—for DoD-specific code, enabling full security testing and rebuilding of binaries from source
D2	Make security a first-order consideration for all software-intensive systems, recognizing that security-at-the-perimeter is not enough
D3	Shift from the use of rigid lists of requirements for software programs to a list of desired features and required interfaces/characteristics to avoid requirements creep, overly ambitious requirements, and program delays

Chapter 5 provides additional context and Appendix A contains draft implementation plans.

Chapter 1. Who Cares: Why Does Software Matter for DoD?

The future battlespace is constructed of not only ships, tanks, missiles, and satellites, but also algorithms, networks, and sensor grids. Like no other time in history, future wars will be fought on civilian and military infrastructures of satellite systems, electric power grids, communications networks, and transportation systems, and within human networks. Both of these battlefields—electronic and human—are susceptible to manipulation by adversary algorithms.

— Cortney Weinbaum and Lt Gen John N.T. "Jack" Shanahan, "Intelligence in a Data-Driven Age," (Joint Force Quarterly 90, 2018), 5

This chapter provides a high-level vision of why software is critical for national security and the types of software we will have to build in the future. We also provide a description of different types of software, where they are used, and why a one-size-fits-all approach will not work.

1.1 Where Are We Coming From, Where Are We Going?

While software development has always been a challenge for the Department of Defense (DoD), today these challenges greatly affect our ability to deploy and maintain mission-critical systems to meet current and future threats. In the past, software simply served as an enabler of hardware systems and weapons platforms. Today, software defines our mission-critical capabilities and our ability to sense, share, integrate, coordinate, and act.

Software is everywhere and is in almost everything that the Department operates and uses. Software drives our weapon systems; command, control, and communications systems; intelligence systems; logistics; and infrastructure, and it drives much of the backroom enterprise processes that make the Department function. If cyber is the new domain in which we are fighting, then our ability to maintain situational awareness and our ability to fight, defend, and counter threats will be based on the capabilities of our software. In this new domain, software is both an enabler as well as a target of the fight.

As our military systems become increasingly networked and automated, as autonomy becomes more prevalent, and as we become more dependent on machine learning (ML) and artificial intelligence (AI), our ability to maintain superiority will be directly linked to our ability to field and maintain software that is better, smarter, and more capable than our adversaries' software. Even our ability to defend against new physical and kinetic threats such as hypersonics, energetics, and biological weapons will be based on software capabilities. We need to identify and respond to these new threats as they happen in near real time. Our ability to identify and respond to these new threats will be based on our ability to develop and push new software-defined capabilities to meet those threats on time scales that greatly outpace our adversaries' ability to do so.

The need to meet future threats requires us to rethink how we develop, procure, assure, deploy, and continuously improve software. DoD's current procurement processes treat software programs like hardware programs, but DoD can no longer take years to develop software for its major systems. Software cannot be an afterthought to hardware, and it cannot be acquired, developed, and managed like hardware. DoD's acquisition and development approaches are increasingly antiquated and do not meet the timely demands of its missions. Fixing the

Department's software approach involves more than just making sure that we get control over cost and budget; it concerns our ability to maintain our fighting readiness and our ability to win the fight and counter any threat regardless of domain and regardless of adversary.

1.2 Weapons and Software and Systems, Oh My! A Taxonomy for DoD

Not all software systems are the same, and therefore it is important to optimize development processes and oversight mechanisms to the different types of software DoD uses. We distinguish here between two different aspects of software: *operational function* (use) and *implementation platform*. To a large extent, a given operational function can be implemented on many different computational platforms depending on whether it is a mission support function (where high-bandwidth connectivity to the cloud is highly likely) or a field-forward software application (where connectivity many be compromised and/or undesirable).

We define three broad operational categories:

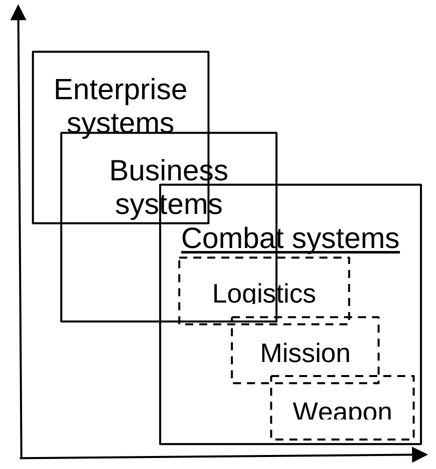


Figure 1.1. Different types of software.

- *Enterprise systems*: very large-scale software systems intended to manage a large collection of users, interface with many other systems, and generally used at the DoD level or equivalent. These systems should always run in the cloud and should use architectures that allow interoperability, expandability, and reliability. In most cases the software should be commercial software purchased (or licensed) without modification to the underlying code, but with DoD-specific configuration. Examples include e-mail systems, accounting systems, travel systems, and human resources (HR) databases.

- *Business systems*: essentially the same as enterprise systems, but operating at a slightly smaller scale (e.g., for one of the Services). Like enterprise systems, they are interoperable, expandable, reliable, and probably based on commercial offerings. Similar functions may be customized differently by individual Services, though they should all interoperate with DoD-wide enterprise systems. Depending on their use, these systems may run in the cloud, in local data centers, or on desktop computers. Examples include software development environments and Service-specific HR, financial, and logistics systems.

- *Combat systems*: software applications that are unique to the national security space and used as part of combat operations. Combat systems may require some level of customization that may be unique to DoD, not the least of which will be specialized cybersecurity considerations to enable them to continue to function during an adversarial attack. (Note that since modern DoD enterprise and business systems depend on software, cyber attacks to disrupt the operations of these systems have the potential to be just as crippling as those aimed at combat systems.)

We further break down combat systems into subcategories:

- ○ *Logistics systems*: any system used to keep track of materials, supplies, and transport as part of operational use (versus Service-scale logistics systems, with which they should interoperate). While used actively during operations, logistics systems are likely to run on commercial hardware and operating systems, allowing them to build on commercial off-the-shelf (COTS) technologies. Platform-based architectures enable integration of new capabilities and functions over time (probably on a months-long or annual time scale). Operation in the cloud or based on servers is likely.

- ○ *Mission systems*: any system used to plan and monitor ongoing operations. Similar to logistics systems, this software will typically use commercial hardware and operating systems and may be run in the cloud, on local services, or via a combination of the two (including fallback modes). Even if run locally (such as in an air operations center), they will heavily leverage cloud technologies, at least in terms of critical functions. These systems should be able to incorporate new functionality at a rate that is set by the speed at which the operational environment changes (days to months).

- ○ *Weapon systems*: any system capable of delivering lethal force, as well as any direct support systems used as part of the operation of the weapon. Note that our definition differs from the standard DoD definition[1] of a weapon system, which also includes any related equipment, materials, services, personnel, and means of delivery and deployment (if applicable) required for self-sufficiency. The DoD definition would most likely include the mission and logistics functions, which we find useful to break out separately. Software on weapon systems is traditionally closely tied to hardware, but as we move toward greater reliability of software-defined systems and distributed intelligence, weapon systems software is becoming increasingly hardware independent (similar to operating systems for mobile devices, which run across many different hardware platforms).

We also define several different types of computing platforms on which the operational functions above might be implemented:

- *Cloud computing*: computing that is typically provided in a manner such that the specific location of the compute hardware is not relevant (and may change over time). These systems typically run on commercial hardware and use commercial operating systems, and the applications running on them run even as the underlying hardware changes. The important point here is that the hardware and operating systems are generally transparent to the application and its users (see figure 1.2).

[1] The Department of Defense, *DoD Dictionary of Military and Associated Terms* (Washington, DC: Department of Defense, as of February 2019), 252.

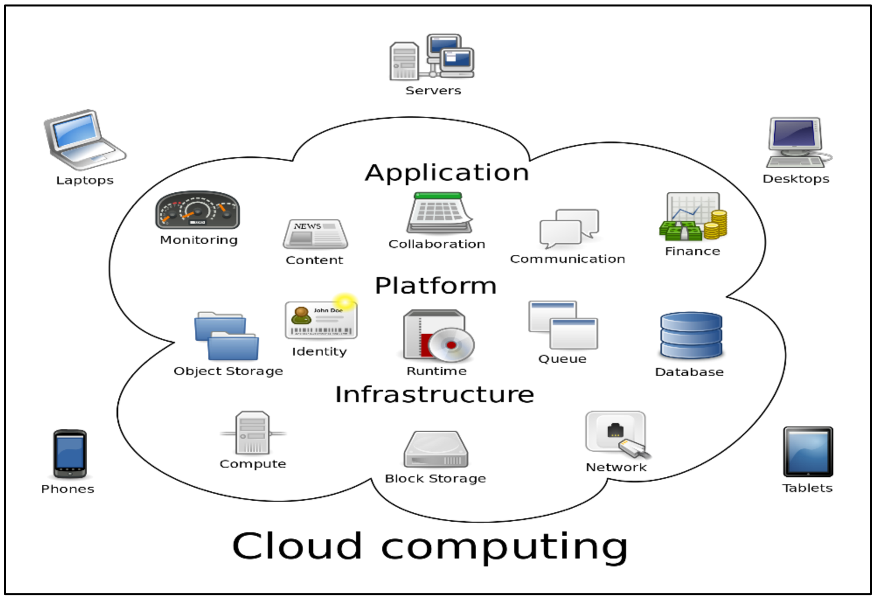


Figure 1.2. Cloud computing environment.
[Image by Sam Johnston is licensed under CC BY-SA 3.0]

- *Client/server computing*: computing provided by a combination of hardware resources available in a computing center (servers) as well as local computing (client). These systems usually run on commercial hardware and use commercial operating systems.
- *Desktop/laptop/computing*: computing that is carried out on a single system, often by interacting with data sources across a network. These systems usually run on commercial hardware and use commercial operating systems.
- *Mobile computing:* computing that is carried out on a mobile device, usually connected to the network via wireless communications. These systems usually run on commercial operating systems using commodity chipsets.
- *Embedded computing*: computing that is tied to a physical, often-customized hardware platform and that has special features that require careful integration between software and hardware (see figure 1.3).

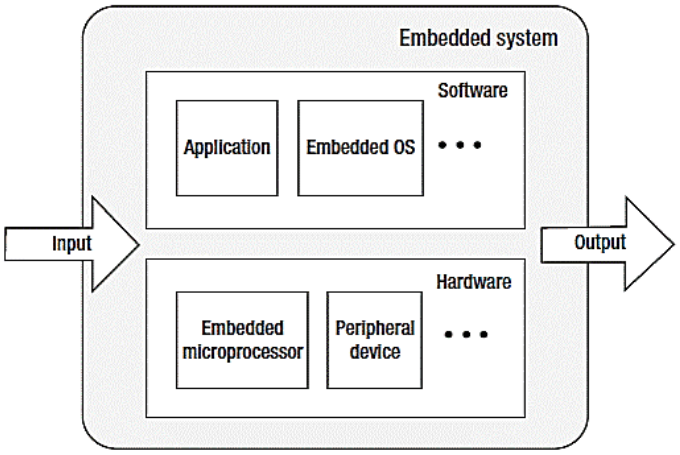


Figure 1.3. Embedded system architecture.
[Image from Ebrary.net]

A single software system may have multiple components or functions that span several of these definitions, and components of an integrated system likely have elements that do the same. The key point is that each type of software system has different requirements in terms of how quickly it can/should be updated, the level of information assurance required, and the organizations that will participate in development, testing, customization, and use of the software. Different statutes, regulations, and processes may be required for different types of software (and these would differ greatly from those used for hardware).

Having defined systems that deliver effects and the kinds of computing platforms on which software is hosted, we now distinguish between four primary types of software. We use these terms throughout the rest of the report to differentiate the acquisition and deployment approaches needed for different types of software:

- **Type A (Commercial Off-the-Shelf [COTS] applications):** The first class of software consists of applications that are available from commercial suppliers. Business processes, financial management, HR, software development, collaboration tools, accounting software, and other "enterprise" applications in DoD are generally not more complicated nor significantly larger in scale than those in the private sector. Unmodified commercial software should be deployed in nearly all circumstances. Where DoD processes are not amenable to this approach, the Department should modify its processes, not the software.

- **Type B (Customized Software):** The second class of software constitutes those applications that consist of commercially available software that is customized for DoD-specific usage. Customization can include the use of configuration files, parameter values, or scripted functions tailored for DoD missions. These applications generally require (ongoing) configuration by DoD personnel, contractors, or vendors.

- **Type C (COTS Hardware/Operating Systems):** The third class of software applications is those that are highly specialized for DoD operations but run on commercial hardware and standard operating systems (e.g., Linux or Windows). These applications will generally be able to take advantage of commercial processes for software development and deployment, including the use of open source code and tools. This class of software includes applications written by DoD personnel as well as those that are developed by contractors.

- **Type D (Custom Software/Hardware):** This class of software focuses on applications involving real-time, mission-critical, embedded software whose design is highly coupled to its customized hardware. Examples include primary avionics or engine control, or target tracking in shipboard radar systems. Requirements such as safety, target discrimination, and fundamental timing considerations demand that extensive formal analysis, test, validation, and verification activities be carried out in virtual and "iron bird" environments before deployment to active systems. These considerations also warrant care in the way application programming interfaces (APIs) are potentially presented to third parties.

We note that these classes of software are closely related to those described in the 1987 Defense Science Board (DSB) study on military software, which categorized software as "standard" (roughly capturing types A and B), "extended" (type C), "embedded" (type D), and "advanced" (which the study categorized as "advanced and exploratory systems," which are not so relevant here).

1.3 What Kind of Software Practices Will We Have to Enable?

The competitor that can realize software-defined military capability the fastest is at an advantage in future conflicts. We must shorten our development cycles from years to months so that we can react and respond within the observe–orient–decide–act (OODA) loop of the threats we face. Agile methodologies such as DevSecOps enable this rapid cycle approach (see "Detecting Agile BS" in Appendix E for more information about agile methodologies), and in addition to development we will need to test and validate software in real time as part of the integrated approach that DevSecOps demands. Quality assurance must be a continuous and fully integrated process throughout every phase of the software cycle. We need to build software pipelines that are able to develop and deploy software and provide updates as quickly as modern-day commercial companies so that we can respond to new threats (especially when the target will be our software). We must treat software as a continuous service rather than as block deliverables. It is important to have the agility in our procurement approach that will allow program managers to change priorities based on the needs and timing of the end users.

In the near future, DoD's acquisition and use of business systems should closely mirror industry and the private sector. DoD should modify its processes to mimic industry's best practices rather than try to contract for and maintain customized software. Figure 1.4 illustrates how this looks at Facebook (see also Section 2.1 for examples of best practices in industry).

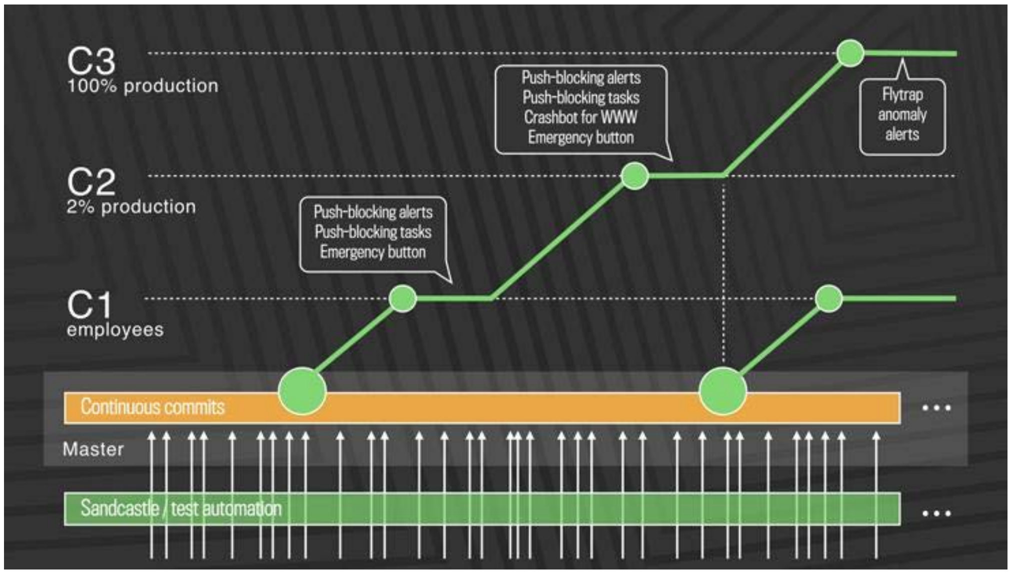


Figure 1.4. Facebook's continuous delivery process. Code updates that have passed a series of automated internal tests (bottom) land in the master development branch and are pushed out to employees (C1). In this stage, push-blocking alerts are generated if there are problems, and an emergency stop button keeps the release from going any further. If everything is OK, changes are pushed to 2 percent of production (C2), where signal and monitor alerts are again collected, especially for edge cases that testing or employee use may not have picked up. Finally, changes are rolled out to 100 percent of production (C3), where the "Flytrap" tool aggregates user reports and provides alerts on any anomalies. The cycle time between updates can be as short as a few hours. [Diagram and caption adapted from Facebook Engineering Blog, 31 Aug 2017 post on "Rapid release at massive scale"]

DoD should also adopt commercial logistics and mission planning software (COTS) wherever possible and reduce its reliance on government off-the-shelf (GOTS) solutions. Good logistics and mission software reduces process complexity, improves situational awareness, reduces costs, and simplifies planning while improving speed of delivery and streamlining performance.

For software that is closely tied to hardware, software-defined systems should be easier to develop, maintain, and upgrade than classic embedded systems. A well-designed system would allow new capabilities to be delivered directly to the edges of the network from the cloud in the same way new capabilities are delivered to consumer mobile devices.

DoD should manage software by measuring value delivered to the user rather than by monitoring compliance with requirements. Accountability should be based on delivering value to the user and solving user needs, not on complying with obsolete contracts or requirements documents.

Program managers must identify potential problems earlier (ideally, within months) and take corrective action quickly. Troubled programs must fail quickly, and the Department needs to learn from them. As we witnessed throughout our work on this study, many software programs are too big, are too complex, and take too long to deliver any value to users. Development must be staged and follow the best practice of smaller deliverables faster, with higher frequency of updates and new features. Initially, program development should focus on developing the “minimum viable product” (MVP) and getting it delivered to the customer more quickly than traditionally run programs. (The MVP for a software program represents the first point at which the code can start doing useful work and also at which feedback can be gathered that supports refinement of features.)

Software developers within the defense community need the same modern tools, systems, environments, and collaboration resources that commercial industry has adopted as standard. Without these, the Department undermines the effectiveness of its software developer base, and its ability to attract and retain our software human capital, both within DoD and among its suppliers. With the introduction of new technologies like ML and AI and the ever-increasing interdependence among networked heterogeneous systems, software complexity will continue to increase logarithmically. DoD needs to continuously invest in new development tools and environments including simulation environments, modeling, automated testing, and validation tools. DoD must invest in research and development (R&D) into new technologies and methodologies for software development to help the Department keep up with the ever-growing complexity of defense systems.

1.4 What Challenges Do We Face (and Consequences of Inaction)?

The world is changing. The United States used to be the dominant supplier of software and the world leader in software innovation. That is no longer the case. Due to the global digital revolution driven by the consumer and commercial markets, countries are building their own indigenous software capabilities and their own technology clusters. Countries like China are making huge investments in AI and cyber. China's 2030 plan envisions a $1 trillion AI industry in China.[2] China wants to become a cyber superpower and is investing in its capital markets, universities, research centers, defense industry, and commercial software companies to reach that goal.[3]

The potential long-term consequences of inaction are that our adversaries' software capabilities could catch and surpass those of the United States. If that happens, our adversaries would be able to develop new capabilities and potentially iterate faster than we can. They could respond to our defense systems faster than we can respond to theirs. If their algorithms and AI become superior to ours, they could hold a decisive advantage when any of our systems go up against any of theirs. And if their cyber capability becomes superior to ours, they could shut us down, cause chaos, continue to steal our secrets as they choose and without repercussions—especially if we could not attribute those attacks. Our adversaries' software capabilities are growing rapidly. If we do not keep pace, we could lose our defense technology advantage within a decade or much sooner.

[2] Vikram Barhat, "China Is Determined to Steal A.I. Crown from US and Nothing, Not Even a Trade War, Will Stop It," CNBC, May 4, 2018, https://www.cnbc.com/2018/05/04/china-aims-to-steal-us-a-i-crown-and-not-even-trade-war-will-stop-it.html.
[3] "China Is Seeking to Become a Cyber Superpower," *The Economist*, March 20, 2018, https://www.economist.com/graphic-detail/2018/03/20/china-is-seeking-to-become-a-cyber-superpower; and Rogier Creemers, Paul Triolo, and Graham Webster, "Translation: Xi Jinping's April 20 Speech at the National Cybersecurity and Informatization Work Conference," New America Blog Post, April 30, 2018, https://www.newamerica.org/cybersecurity-initiative/digichina/blog/translation-xi-jinpings-april-20-speech-national-cybersecurity-and-informatization-work-conference/.

Chapter 2. What Does It Look Like to Do Software Right?

Deliver performance at the speed of relevance. Success no longer goes to the country that develops a new technology first, but rather to the one that better integrates it and adapts its way of fighting. Current processes are not responsive to need; the Department is over-optimized for exceptional performance at the expense of providing timely decisions, policies, and capabilities to the warfighter. Our response will be to prioritize speed of delivery, continuous adaptation, and frequent modular upgrades. We must not accept cumbersome approval chains, wasteful applications of resources in uncompetitive space, or overly risk-averse thinking that impedes change. Delivering performance means we will shed outdated management practices and structures while integrating insights from business innovation.

— U.S. Department of Defense, "Summary of the 2018 National Defense Strategy of the United States of America: Sharpening the American Military's Competitive Edge," (Washington, DC: U.S. Department of Defense, 2018), 10

In many cases, the software acquisition approaches and practices in place within DoD today look strange and perplexing to those familiar with commercial software practices. While the mission-, security-, and safety-critical nature of DoD's software in the context of embedded weapons will have an impact on practices, the extreme degree of divergence from contemporary commercial practice has been an area of our focus. Our case studies, site visits, and other study activities allowed a closer look into the reasons for divergence and whether the absence of many commercial best practices is justified.

2.1 How It Works in Industry (and Can/Should Work in DoD): DevSecOps

Figure 2.1 A former U.S. Marine Corps sergeant, now a Microsoft field engineer, works with an IT support specialist with the Navy as part of his job to travel to commercial companies and military bases across the country and train IT staff about a systems management product. [Photo by Sgt. Shellie Hall]

Modern software companies must develop and deliver software quickly and efficiently in order to survive in a hyper-competitive environment. While it is difficult to characterize the entire software sector, in this section we outline a set of practices—based on documented approaches in industry[4]—that are representative of commercial environments where the delivery of software capability determines the success or failure of the company. These practices generally hold true in other industries where companies have unexpectedly found themselves in the software business due to an increasing reliance on software to provide their key offerings, such as automotive, banking, healthcare, and many others. In any

[4] Fergus Henderson, "Software Engineering at Google" (arXiv:1702.01715 [cs.SE], January 31, 2017).

environment, software engineering practices must be matched with the recruitment and retention of talented software expertise. These practices must be honed over time and adapted to lessons learned.

At a high level, DoD must move from waterfall and spiral development methods to more modern software development practices such as Agile, DevOps, and DevSecOps. "DevOps" represents the integration of software development and software operations, along with the tools and culture that support rapid prototyping and deployment, early engagement with the end user, automation and monitoring of software, and psychological safety (e.g., blameless reviews). "DevSecOps" (as depicted in figure 2.2) adds the integration of security at all stages of development and deployment, which is essential for DoD applications. DoD should adopt these techniques, with appropriate tuning of approaches used by the Agile/DevSecOps community for mission-critical, national security applications. DoD should use open source software when possible to speed development and deployment and leverage the work of others.

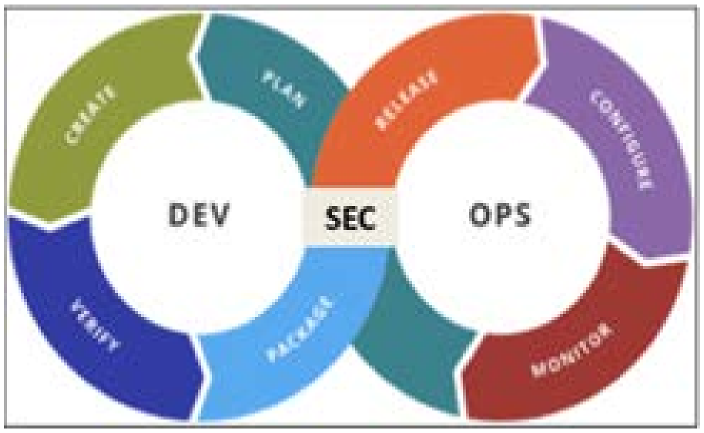


Figure 2.2. Continuous integration of development, security, and deployment (DevSecOps). [Adapted from an image by Kharnagy, licensed under CC BY-SA 4.0]

Generally, successful software companies have developed best practices in three categories:

Software development. These are software engineering practices that include source code management, software build, code review, testing, bug tracking, release, launch, and postmortems. Key best practices applicable to DoD software programs include the following:

- All source code is maintained in a single repository that is available to all software engineers. There are control mechanisms to manage additions to the repository, but in some cases all engineers are culturally encouraged to fix problems, independent of program boundaries.
- Developers are strongly encouraged to avoid "forking" source code (creating independent development branches) and focus work on the main branch of the software development.
- Code review tools are reliable and easy to use. Changes to the main source code typically require review by at least one other engineer, and code review discussions are open and collaborative.
- Unit test is ubiquitous, fully automated, and integrated into the software review process. Integration, regression, and load testing are also widely used, and these activities should be an integrated, automated part of daily workflow.
- Releases are frequent—often weekly. There is an incremental staging process over several days, particularly for high-traffic, high-reliability services.

- Postmortems are conducted after system outages. The focus of the postmortem is on how to avoid problems in the future and not on affixing blame.

Project management. Software projects must contribute to the overall aim of the business, and efforts must be aligned to that end goal.

- Individuals and teams set goals, usually quarterly and annually. Progress against those goals is tracked, reported, and shared across the organization. Goals are mechanisms to encourage high performance but can be decoupled from performance appraisal or compensation.
- The project approval process is organic. Significant latitude to initiate projects is given at all levels, with oversight responsibility given to managers and executives to allocate resources or cancel projects.

People management. Given the scarce number of skilled software engineers, successful software companies know how to encourage and reward good talent. Examples include the following:

- Engineering and management roles are clearly separated, with advancement paths for both. Technical career progression (e.g., for advanced and senior developers, fellows and senior fellows) parallels management career ladders; technical professionals receive similar compensation and accrue comparable respect within the organization. Similar distinctions are made between technical management and people management. The ratio of software engineers to product managers and program managers ranges from 4:1 to 30:1.
- Mobility throughout the organization is encouraged. This allows for the spread of technology, knowledge, and culture throughout the company.

In addition to these specific software development practices, another common approach to managing programs in industry is to move away from the specifications and requirements approach towards a feature management approach. This approach allows program managers to make agile decisions based on evolving needs and capabilities. Using a feature management approach, a program manager has a list of features and capabilities ranked by need, risk, cost, resources, and time. This list of capabilities is two to three times larger than what generally can be accomplished within a given time frame, a given budget, and a set of resources. Program managers make decisions about the feature mix, match investments to needs, and balance risk against performance. Capabilities are tested and delivered on a continuous basis, and maximum automation is leveraged for testing.

In industry, software programs initially start as an MVP. An MVP has just enough features to meet basic minimum functionality. It provides the foundational capabilities upon which improvements can be made. MVPs have significantly shorter development cycles than traditional waterfall approaches. The goal of MVPs is to get basic capabilities into users' hands for evaluation and feedback. Program managers use the evaluation and feedback results to rebalance and re-prioritize the software capability portfolio.

Portfolio success is measured based on performance of the *delivery* of capabilities as measured against user needs and strategic objectives within an investment cycle. Value is determined by output measurements rather than process measurements. Portfolio value is the aggregate of the total value of all of the capabilities delivered divided by total cost invested within a period of time. Blending higher risk/higher reward capabilities with lower risk/lower reward capabilities is the art of good portfolio management. Within a given period of time, program managers use diversification to spread risk and rewards. Good program managers identify troubled projects early and are encouraged either to quickly correct the problems or to quickly abandon failing efforts so that remaining resources can be husbanded and then reallocated to other priorities.

Software budgets are driven by time, talent, compute resources, development environment, and testing capabilities required to deliver capabilities. The capability and cost of talent vary greatly between software engineers, designers, programmers, and managers. The quality of engineering talent is the single largest variable that determines cost, risk, and duration of a software project. Good portfolio managers must take inventory of the range of software talent within a program and carefully allocate that talent across the portfolio of capabilities development.

2.2 Empowering the Workforce: Building Talent Inside and Out

One of the biggest barriers to realizing the software capabilities the Department so desperately needs is the way the Department manages the people necessary to build that capability. DoD cannot compete and dominate in defense software without a technical and design workforce within the Department that can both build software natively and effectively manage vendors to do the same, using the proven principles and practices described above. Some of the Department’s human capital practices actively work against this critical goal.

If the Department wants to be good at software, it must be good at recruiting, retaining, leveraging, managing, and developing the people who make it. When we look at private-sector organizations and institutions that effectively use software to fulfill their mission, they

- understand the software professionals that they have, understand their workforce needs at a high level, and understand the gap between the two. (We say “at a high level” because we believe the gap is large enough that it is much more important to begin closing the gap than it is to measure the gap with too much precision.)
- have a strategy to recruit the people and skills they need to fulfill their mission, understanding what they uniquely have to offer in a competitive market.
- clearly understand the competencies required by software professionals in their organizations and the expectations of these professionals at each level in the organization.
- define career ladders for technical professionals that map software competencies and expectations from entry level to senior technical leadership and management.
- offer opportunities for learning and mentorship from more senior engineering and design leaders.
- count engineering and design leaders among their most senior leadership, with the ability to advocate across silos for the needs of the software and software acquisition workforce and support other senior leaders in understanding how to work with both.

- support a cadre of leadership able and empowered to create a culture of software management and promote common approaches, practices, platforms, and tools, while retaining the ability to use judgement about when to deviate from those common approaches and tools.
- reward software professionals based on merit and demonstrated contribution rather than time in grade.

Unfortunately, these are not the common descriptors for the software workforce practices in today's DoD.

DoD has long recognized that medicine and law require specialized skills, continuing education, and support and made it not only possible but desirable and rewarding to have a career as a doctor or lawyer in the armed forces. In contrast, software developers, designers, and managers in the Services must practice their skills intermittently and often without support as they endure frequent rotations into other roles. DoD does not expect a trained physician to constantly rotate into deployments focused on aviation maintenance, nor does it interrupt the training of a lawyer to teach him or her HR skills. Who would be comfortable being treated by a physician who worked in an institution that lacked common standards of care and provided no continuing education? And though software is often a matter of life and death, DoD's current human capital practices include all of these counterproductive features.

Figure 2.3. Airmen participate in Kessel Run's pair programming. [U.S. Air Force photo by Rick Berry]

The process to retool human capital practices to meet the challenge of software competency in DoD must start with the people the Department already has who have software skills or who are interested in acquiring them. Unlike medicine, software skills can be acquired through self-directed and even informal training resources such as on-demand, online webinars and coding boot camps, etc., and the Department has military and civilian individuals who have taken it upon themselves to gain technical skills outside of or in addition to formal DoD training. This kind of initiative and aptitude, especially when it results in real contribution to the mission, should be rewarded with appropriate opportunities for career advancement in this highly sought-after specialty. As we have witnessed during site visits for this study, there are also many individuals with more formally recognized software skills who are working with determination and even courage to try to deliver great software in service of the mission, but whose efforts to practice modern software techniques are poorly supported, and often actively blocked. Changes to policy that make clear the Department's support for these practices will help, but they must be married with support for the individuals to stay and grow within their chosen field. DoD could leverage several possible human capital pathways:

- Core military occupational series (MOS) and civilian occupational series for software development that include subcategories to address the various duties found in modern software development (e.g., developers/engineers, product owners, and designers).
- A secondary specialty series/designator for military members for software development. Experts come from various backgrounds, and a special secondary designator or occupational series for Service Members would be invaluable to tapping into their expertise even if they are not part of the core "Information Technology" profession.
- A Special Experience Identifier or other Endorsement for military and civilian acquisition professionals that indicates they have the necessary experience and training to serve on a software acquisition team. This Identifier or Endorsement should be a requirement to lead an acquisition team for a software procurement. Furthermore, this Identifier or Endorsement needs to be expanded to the broader team working the software procurement to include legal counsel, contract specialists, and financial analysts.

2.3 Getting It Right: Better Oversight AND Superior National Security

Getting software right in the Department requires more than changing development practices; oversight (and budgeting and finance) must also change. Those responsible for oversight of DoD software projects will need to learn to ask different questions and require different kinds of information on different tempos, but their reward will be more clarity, greater satisfaction with military software investments, and, ultimately, stronger national security.

Rules of thumb for those in appropriations and oversight roles over DevSecOps projects include the following:

Expect value to the user earlier. Oversight of monolithic, waterfall projects has generally focused on whether the team hit pre-determined milestones that may or may not represent actual value or even working code, and on figuring out what to do when they do not. When evaluating and appropriating funds to DevSecOps projects, it is more suitable to judge the project on the speed by which it delivers working code and actual value to users. In a waterfall project, changes to the plan generally reflect the team falling behind and are a cause for concern. In a project that is agile and takes advantage of the other approaches this study recommends (including software reuse), the plan is intended to be flexible because the team should be learning what works as they code and test.

Ask for meaningful metrics. Successful projects will develop metrics that measure value to the user, which involves close, ongoing communication with users. Source lines of code (SLOC) is not a measure of value and should not be used to evaluate projects in any case, as its use creates perverse incentives.

Assign a leader and hold him or her accountable. Part of the role of oversight is to ensure that there is a single leader who is qualified to lead in a DevSecOps framework and has the authority and responsibility to make the decisions necessary for the project to succeed. That person should have the authority to assign tasks and work elements; make business, product, and technical decisions; and manage the feature and bug backlogs. This person is ultimately responsible for how well the software meets the needs of its users, which is how the project should be evaluated.

Clarity and quality of leadership has long been tied to successful defense programs. Consider Kelly Johnson with the U-2, F-104, and SR-71. Paul Kaminski with stealth technology. Admiral Hyman Rickover with the nuclear Navy. Harry Hillaker with the F-16; and Bennie Schriever with the intercontinental ballistic missile. The list goes on. The United States Digital Service recognized this with Play 6 of the *Digital Services Playbook*—Assign One Leader and Hold That Person Accountable.[5] DoD would do well to remember this part of its history and work this practice into its oversight plan.

Speed increases security. Conventional wisdom in DoD says that programs must move slowly because moving quickly would threaten security. Often, the opposite is true. As we have learned from the cyber world, when we are facing active threats, our ability to achieve faster detection, response, and mitigation reduces the consequences of an attack or breach. In the digital domain, where attacks can be launched at machine speeds, where AI and ML can probe and exploit vulnerabilities in near real time, our current ability to detect, respond, and mitigate against digital threat leaves our systems completely vulnerable to our adversaries.

> The Department of Defense (DoD) faces mounting challenges in protecting its weapon systems from increasingly sophisticated cyber threats. This state is due to the computerized nature of weapon systems; DoD's late start in prioritizing weapon systems cybersecurity; and DoD's nascent understanding of how to develop more secure weapon systems. DoD weapon systems are more software dependent and more networked than ever before…. Potential adversaries have developed advanced cyber-espionage and cyber-attack capabilities that target DoD systems. (U.S. Government Accountability Office, Weapon Systems Cybersecurity: DoD Just Beginning to Grapple with Scale of Vulnerabilities [Washington, DC: U.S. Government Accountability Office, Oct 9, 2018], 2)

DoD must operate within its adversaries' digital OODA loop. Much like today's consumer electronic companies, the Department needs the ability to identify and mitigate evolving software and digital threats and to push continuous updates to fielded systems in near-real time.

DoD must be able to deploy software faster without sacrificing its abilities to test and validate software. To accomplish this, the Department needs to reimagine the software development cycle as a continuous flow rather than discrete software block upgrades. It should not only modernize to use a DevSecOps approach to software development but should also modernize its entire suite of development and testing tools and environments. DoD needs to be able to instrument its fielded systems so that we can build accurate synthetic models that can be used in development and test. The Department needs to be able to patch, update, enhance, and add new capabilities faster than our adversaries' abilities to exploit vulnerabilities.

Colors of money doom software projects. The foundational reasons for specific Congressional guidance on how money is to be spent make sense. But because software is in continuous

[5] "Digital Services Playbook," U.S. Digital Service, https://playbook.cio.gov/#plays_index_anchor.

development (it is never “done”—see Windows, for example), colors of money tend to doom programs. We need to create pathways for “bleaching” funds to smooth this process for long-term programs.

Do not pay for the factory every time you need a car. Appropriators must realize that DoD desperately needs common infrastructure if it is to increase the speed and quality of the software it produces. Today, it is as if the Department were buying cars but paying for the entire factory to build each car separately. Appropriators should fund the smart development of common infrastructure and reward its use in individual programs and projects. Evaluators should be wary of programs and projects that fail to articulate how they are taking advantage of common infrastructure and reusable components.

Standard is better than custom. In the same vein as the above, appropriators and evaluators should understand the benefits of using standards from the software development industry. Standards enable quality, speed, adoption, cost control, sustainability, and interoperability.

Technical debt is normal, and it is worth investing to pay it down. “Technical debt” refers to the cost incurred by implementing a software solution that is expedient rather than choosing a better approach that would take longer. Appropriators and evaluators should understandably expect to see progress in terms of features on a regular basis. The exceptions are when software teams must pay down technical debt or refactor code for greater performance. (This often results in fewer lines of code but higher performance, which is why it is a mistake to judge a software project based on the number of lines of code.) These periodic investments are to be expected on a DevSecOps project and are necessary to ensure the overall quality and stability of the project.

Use data as a compass, not a grade. Too often, evaluators and appropriators receive data about a program that suggests it is failing, but by the time they receive it, there is not much to be done about it. Data is collected manually, then processed and presented, and by the time it is being discussed, it is out of date. Mostly what happens at this point is that the project is given a poor grade, which makes the teams increasingly risk averse and demoralized. Instead, projects should be instrumented—equipped with built-in ways of seeing how and where they are going—so that the data is available both to the teams and to evaluators in time to make adjustments. In this model, the data is more like a compass, helping all parties make small corrections quickly to avoid the poor grade. An effective oversight function will help steer projects and hold them accountable, rather than punish poor performance.

2.4 Eye on the Prize: What Is the R&D Strategy for Our Investment?

The nature of software development may radically change in the near future. It is essential that the DoD adequately fund R&D programs to advance the fields of computer science, including computer programming, AI and ML, autonomy, quantum computing, networks and complex systems, man–machine interfaces, and cybersecurity.

Today, computers are controlled by programs that are comprised of sets of instructions and rules written by human programmers. AI and ML change how humans teach computers. Instead of providing computers with programmed instructions, humans will train or supervise the learning algorithm being executed on the computer. Training is inherently different than programming. Data becomes more important than code. Training errors are very different than programming errors. Hacking AI is very different than hacking code. The use of synthetic environments and "digital twins" (simulation-based emulators of physical components) may also become increasingly important tools to train a computer. The impact of AI and ML on software development will be profound and necessitates entirely new approaches and methods of developing software.

New computing technologies are also on the horizon. Experts may agree that we are many years away from developing a universal quantum computer (UQC), a generally programmable computer combining both classical and quantum computing elements. Nevertheless, the United States cannot afford to come in second in the race to develop the first UQC. The challenge is not only confined to development of the UQC hardware, but includes developing quantum computing programming languages and software. We also need to continue to invest in new quantum-resistant technologies such as cryptography and algorithms and apply those technologies as soon as possible to protect today's data and information from tomorrow's UQC attacks.

The field of computer science continues to advance with the discovery and development of new computer architectures and designs. We have already seen the impact of new architectures such as cloud computing, GPUs (graphics processing units), low-power electronics, and Internet of Things (IoT) on computing. New architectures are being studied and developed by both industry and academia. DoD should not only continue to invest in the development of new architectures but also to invest in new methods for quicker adoption of these technologies.

Given today's challenge of cybersecurity and software assurance, R&D must continue developing more trusted computing to thwart future cyber attacks and creating abilities to execute software with assurance on untrusted networks and hardware.

DoD should invest in new approaches to software development (beyond Agile), including the use of computer-assisted programming and project management. While agile development is currently a best practice in industry, managing the software cycle is still more art form than science. New analytical approaches and next-generation management tools could significantly improve software performance and schedule predictability. The Department should fund ongoing research as well as support academic, commercial, and development community efforts to innovate the software process.

Chapter 3. Been There, ~~Done~~ Said That: Why Hasn't This Already Happened?

Probably the most dangerous phrase you could ever use in any computer installation is that dreadful one: "but we've always done it that way." That's a forbidden phrase in my office.

— Rear Admiral Grace Hopper (1906-1992), computer programmer, presentation at MIT Lincoln Laboratory on 25 April 1985, 23m41s

DoD and Congress have a rich history of asking experts to assess the state of DoD software capabilities and recommend how to improve them. A DoD joint task force chaired by Duffel in 1982 started its report by saying,

> Computer software has become an important component of modern weapon systems. It integrates and controls many of the hardware components and provides much of the functional capability of a weapon system. Software has been elevated to this prominent role because of its flexibility to change and relatively low replication cost when compared to hardware. It is the preferred means of adding capability to weapon systems and of reacting quickly to new enemy threats. (Report of the DoD Joint Service Task Force on Software Problems, 1982)

Indeed, this largely echoes our own views, although the scope of software has now moved well beyond weapon systems, the importance of software has increased even further, and the rate of change for software is many orders of magnitude faster, at least in the commercial world.

Five years later, a task force chaired by Fred Brooks began its executive summary as follows:

> Many previous studies have provided an abundance of valid conclusions and detailed recommendations. Most remain unimplemented. … [T]he Task Force is convinced that today's major problems with military software development are not technical problems, but management problems. (Report of the Task Force on Military Software, Defense Science Board, 1987)

This particular assessment, from over 30 years ago, referenced over 30 previous studies and is largely aligned with the assessments of more recent studies, including this one.

And finally, in its 2000 study on DoD software, Defense Science Board (DSB) Chair Craig Fields commented that,

> Numerous prior studies contain valid recommendations that could significantly and positively impact DoD software development programs. However the majority of these recommendations have not been implemented. Every effort should be made to understand the inhibitors that prevented previous recommendations. (Defense Science Board Task Force on Defense Software, 2000)

So to a large extent the problem is not that we do not know what to do, but that we simply are not doing it. In this chapter we briefly summarize some of the many reports that have come before ours and attempt to provide some understanding of why the current state of affairs in defense software is still so problematic. Using these insights, we attempt to provide some level of confidence that our recommendations might be handled differently (remembering that "hope is not a strategy").

3.1 37 Years of Prior Reports on DoD Software

The following table lists previous reports focused on improving software acquisition and practices within DoD.

Date	Org	Short title / Summary of contents
Jul'82	DoD	Joint Service Task Force on Software Problems 37 pp + 192 pp Supporting Information (SI); 4 major recommendations The opportunities and problems posed by computer software embedded in DoD weapon systems were investigated by a joint Service task force. The task force members with software experience combined existing studies with the observations of DoD project managers. The task force concluded that software represents an important opportunity in regard to the military mission. Further, it was concluded that technological excellence in software is an important factor in maintaining U.S. military superiority, but that many problems facing DoD in software endangers this superiority.
Sep'87	DSB	Task Force on Military Software 41 pp + 36 pp SI; 38 recommendations The task force reviewed current DoD initiatives in software technology and methodology, including the Ada effort, the STARS program, DARPA's Strategic Computing Initiative, the Software Engineering Institute (SEI), and a planned program in the Strategic Defense Initiative. The five initiatives were found to be uncoordinated, and the task force recommended that the Undersecretary of Defense (Acquisition) establish a formal program coordination mechanism for them. In spite of the substantial technical development needed in requirements setting, metrics and measures, tools, etc., the Task Force was convinced that the major problems with military software development were not technical problems, but management problems. The report called for no new initiatives in the development of the technology, some modest shift of focus in the technology efforts underway, but major re-examination and change of attitudes, policies, and practices concerning software acquisition.
Dec'00	DSB	Task Force on Defense Software 36 pp + 10 pp SI; 6 major recommendations The Task Force determined that the majority of problems associated with DoD software development programs are a result of undisciplined execution. Accordingly the Task Force's recommendations emphasized a back-to-the-basics approach. The Task Force also noted that numerous prior studies contain valid recommendations that could significantly and positively impact DoD software development programs. The fact that the majority of these recommendations have not been implemented should lead to efforts designed to understand the inhibitors preventing these recommendations from being enacted.
2004	RAND	Attracting the Best: How the Military Competes for Information Technology Personnel 149 pp; no explicit recommendations Burgeoning private-sector demand for IT workers, escalating private-sector pay in IT, growing military dependence on IT, and faltering military recruiting all led to a concern that military capability was vulnerable to a large shortfall in IT personnel. This report examined the supply of IT personnel compared to the military's projected future manpower requirements. It concluded that IT training and experience, augmented by enlistment bonuses and educational benefits as needed, seemed sufficient to ensure an adequate flow of new recruits into IT. However, sharp increases in military IT requirements had the potential to create difficulties.
Feb'08	NCMA	Generational Inertia: An Impediment to Innovation? 7 pp; no explicit recommendations This article cites data to the effect that approximately 50 percent of the acquisition workforce is within 5 years of retirement. Rather than being a problem, the article feels that retirement of senior contracting specialists could effectively lead to acquisition reform: "Senior contracting specialists' resistance to change and indifference to professional development is the elephant

		in the room that acquisition reformers are unwilling to acknowledge.”
Mar’09	DSB	Task Force on Department of Defense Policies and Procedures for the Acquisition of Information Technology 68 pp + 2 pp dissent + 15 pp SI; 4 major recommendations with 13 subrecommendations The primary conclusion of the task force is that the conventional DoD acquisition process is too long and too cumbersome to fit the needs of the many IT systems that require continuous changes and upgrades. The task force recommended a unique acquisition system for information technology.
2010a	NRC	Achieving Effective Acquisition of Information Technology in the Department of Defense 164 pp + 16 major recommendations This study board was asked to assess the efficacy of DoD’s acquisition and test and evaluation (T&E) processes as applied to IT. The study concluded that DoD is hampered by “a culture and acquisition-related practices that favor large programs, high-level oversight, and a very deliberate, serial approach to development and testing (the waterfall model).” This was contrasted with commercial firms, which have adopted agile approaches that focus on delivering smaller increments rapidly and aggregating them over time to meet capability objectives. Other approaches that run counter to commercial, agile acquisition practices include “the DoD’s process-bound, high-level oversight [that] seems to make demands that cause developers to focus more on process than on product, and end-user participation often is too little and too late.”
2010b	NRC	Critical Code: Software Producibility for Defense 148 pp + 15 major recommendations This study was charged to examine the nature of the national investment in software research and ways to revitalize the knowledge base needed to design, produce, and employ software-intensive systems for tomorrow’s defense needs. The study notes the continued reliance by DoD on software capabilities in achieving its mission and notes that there are important areas where DoD must push the envelope beyond mainstream capability. In other areas, however, DoD benefits by adjusting its practices to conform to government and industry conventions, enabling it to exploit a broader array of more mature market offerings.
Jul’16	CRS	The Department of Defense Acquisition Workforce: Background, Analysis, and Questions for Congress 14 pp; no explicit recommendations The increase in the size of the acquisition workforce has not kept pace with increased acquisition spending, which has signified an increase not only in the workload but also in the complexity of contracting work. This report summarized four Congressional efforts aimed at enhancing the training, recruitment, and retention of acquisition personnel.
Dec’16	CNA	Independent Study of Implementation of Defense Acquisition Workforce Improvement Efforts 147 pp + 30 pp SI; 21 major recommendations This report examines the strategic planning of the Department of Defense regarding the acquisition workforce (AWF). The study found significant improvements in several areas that “not only reversed the decline in AWF capacity from the 1990s, but also reshaped the AWF by increasing the number of early and mid-career personnel.”
Feb’17	SEI	DoD’s Software Sustainment Study Phase I: DoD’s Software Sustainment Ecosystem 101 pp; 5 major recommendations Since the time in the early 1980s when software began to be recognized as important to DoD, software sustainment has been considered a maintenance function. After almost four decades, DoD is also at a tipping point where it needs to deal with the reality that software sustainment is not about maintenance, but rather it is about continuous systems and software engineering for the life cycle to evolve the software product baseline. This report recommends

		changing that paradigm to enable the innovation needed to address a rapidly changing technology environment, specifically through investments in human capital, better performance measurement of software sustainment, and better visibility for the software portfolio.
Mar'17	BPC	Building a F.A.S.T. Force: A Flexible Personnel System for a Modern Military 82 pp + 15 pp SI; 4 major themes with 39 recommendations This study describes today's DoD personnel system as out of step with contemporary needs and issues: "the current system is typically poorly coordinated, lacks accountability, is unable to quickly obtain specialized talent, and fosters a groupthink mentality within the force." It concludes that an effective personnel system has to build a force that is adaptable to new threats as they arise and technically proficient (among other characteristics).
Feb'18	DSB	Design and Acquisition of Software for Defense Systems 28 pp + 22 pp SI; 7 (high-level) recommendations + ~32 subrecommendations The Task Force assessed best practices from commercial industry as well as successes within DoD. Commercial embrace of iterative development has benefited bottom lines and cost, schedule, and testing performance, while the Department and its defense industrial base partners are hampered by bureaucratic practices and an existing government-imposed reward system. The Task Force concluded that the Department needs to change its internal practices to encourage and incentivize new practices in its contractor base. The assessment of the Task Force is that the Department can leverage best practices of iterative development even in its mission-critical software systems.
2018	2016 NDAA	Section 809 Panel - Streamlining and Codifying Acquisition 1,275 pp; 93 recommendations The Section 809 Panel was established by Congress in the FY 2016 NDAA to address issues with the way DoD buys what it needs to equip its warfighters. The panel published an Interim Report and a three-volume Final Report, containing a total of 93 recommendations aimed at changing the overall structure and operations of defense acquisition both strategically and tactically. Some changes hold potential for immediate effect, such as those that remove unnecessary layers of approval in the many steps contracting officers and program managers must take and those that remove unnecessary and redundant reporting requirements. Other changes require a large shift in how the system operates, such as buying readily available products and services in a manner similar to the private sector and managing capabilities from a portfolio, rather than program, perspective.
Apr'19	DIB	Software Is Never Done; Refactoring the Acquisition Code for Competitive Advantage (this document) 78 pp + 207 pp SI; 4 main lines of effort, 10 primary and 0x10 additional recommendations In this report, we focus on three overarching 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 matters; and (3) software is different than hardware (and not all software is the same). We provide a set of major recommendations that focus on four main lines of effort: (A) refactoring statutes, regulations, and processes specifically for software—including acquisition, development, assurance, deployment, and maintenance—to remove hardware-centric bottlenecks while providing more insight and better oversight; (B) creating and maintaining interoperable (cross-program/cross-Service) digital infrastructure to enable continuous and rapid deployment, scaling, testing, and optimization of software as an enduring capability; (C) creating new paths for digital talent and increasing the level of understanding of modern software within the acquisition workforce; and (D) changing the practice of how software is procured and developed by adopting modern software development approaches.

As the table shows, studies dating back to at least 1982 have identified software as a particular area of growing importance to DoD—and software acquisition as requiring improvement—and the frequency and urgency of such studies identifying software acquisition as a major issue requiring reform has increased markedly since 2010. Notable recent examples include the 2010 studies by the National Research Council on *Achieving Effective Acquisition of Information Technology in the Department of Defense* and *Critical Code: Software Producibility for Defense*, the 2017 study conducted by the Carnegie Mellon University Software Engineering Institute (SEI) on DoD's Software Sustainment Ecosystem, and the 2018 DSB study on *Design and Acquisition of Software for Defense Systems*.

The properties of software that contribute to its unique and growing importance to DoD are summarized in this quote from the 2010 *Critical Code* study:

> Software is uniquely unbounded and flexible, having relatively few intrinsic limits on the degree to which it can be scaled in complexity and capability. Software is an abstract and purely synthetic medium that, for the most part, lacks fundamental physical limits and natural constraints. For example, unlike physical hardware, software can be delivered and up-graded electronically and remotely, greatly facilitating rapid adaptation to changes in adversary threats, mission priorities, technology, and other aspects of the operating environment. The principal constraint is the human intellectual capacity to understand systems, to build tools to manage them, and to provide assurance—all at ever-greater levels of complexity. (*Critical Code: Software Producibility for Defense*, NRC, 2010)

Prior studies have observed that much of DoD software acquisition policy is systems- and hardware-oriented and largely does not take these unique properties into account.[6]

The lack of action on most of the software recommendations from these studies has also been a subject of perennial comment. The DSB's 2000 study noted this phenomenon:

> [Prior] studies contained 134 recommendations, of which only a very few have been implemented. Most all of the recommendations remain valid today and many could significantly and positively impact DoD software development capability. The DoD's failure to implement these recommendations is most disturbing and is perhaps the most relevant finding of the Task Force. Clearly, there are inhibitors within the DoD to adopting the recommended changes. (Task Force on Defense Software, Defense Science Board, 2000)

The situation has not changed significantly since then despite additional studies and significant numbers of new recommendations. There is little to suggest that the inhibitors to good software practice have changed since 2000, and it is likely that the pace of technological change and addition of new capabilities provided by software have only increased since then.

Major categories of prior recommendations. The SWAP study team conducted a literature review of prior work on DoD software acquisition and extracted the specific recommendations that had been made, binning them according to major topics. The focus of the effort was on recent studies, with the bulk of the work since 2010, resulting in 139 recommendations that were extracted and categorized.

[6] For example, "DoD's Software Sustainment Study Phase I: DoD's Software Sustainment Ecosystem," SEI, 2017.

A few prevailing themes stood out from this body of work, representing issues that were commented upon in multiple studies:

- Contracts: contracts should be modular and flexible.
- Test and evaluation: test and evaluation (T&E) should be incorporated throughout the software process with close user engagement.
- Workforce: software acquisition requires specific skills and knowledge along with user interaction and senior leadership support.
- Requirements: requirements should be reasonable and prioritized; X (the focus of each report) should advocate for the need to move from compliance-based, overly prescriptive requirements to more iterative approaches.
- Acquisition strategy/oversight: DoD should encourage agencies to pursue business process innovations.
- Software process: the Department should adopt spiral/agile development approaches to reduce cost, risk, and time.

The three areas that were dealt with most often in the prior studies were acquisition oversight, contracting, and workforce. These three topics alone accounted for 60 percent of all of the recommendations we compiled. We summarize the major recurring prior recommendations in each of those areas as follows:

Recommendations from recent work in acquisition oversight:

- Ensure non-interruption of funding of programs that are successfully executing to objective (rather than budget), while insulating programs from unfunded mandates.
- Ensure that durations be reasonably short and meaningful and allow for discrete progress measurement.
- Design the overall technology maturity assessment strategy for the program or project.
- Encourage program managers to share bad news, and encourage collaboration and communication.
- Require program managers to stay with a project to its end.
- Empower program managers to make decisions on the direction of the program and to resolve problems and implement solutions.
- Follow an evolutionary path toward meeting mission needs rather than attempting to satisfy all needs in a single step.

Recommendations from recent work in contracting:

- Requests for proposals (RFPs) for acquisition programs entering risk reduction and full development should specify the basic elements of the software framework supporting the software factory, including code and document repositories, test infrastructure, software tools, check-in notes, code provenance, and reference and working documents informing development, test, and deployment.

- Establish a common list of source selection criteria for evaluating software factories for use throughout the Department.
- Contracting Officers (KOs) must function as strategic partners tightly integrated into the program office, rather than operate as a separate organization that simply processes the contract paperwork.
- Develop and maintain core competencies in diverse acquisition approaches and increase the use of venture capital–type acquisitions such as Small Business Innovative Research (SBIR), Advanced Concept Technology Development (ACTD), and Other Transaction Authority (OTA) as mechanisms to draw in nontraditional companies.

Recommendations from recent work on workforce issues:

- Service acquisition commands need to develop workforce competency and a deep familiarity with current software development techniques.
- The different acquisition phases require different types of leaders. The early phases call for visionary innovators who can explore the full opportunity space and engage in intuitive decision making. The development and production phases demand a more pragmatic orchestrator to execute the designs and strategies via collaboration and consensus decisions.
- U.S. Special Operations Command (USSOCOM) must develop a unique organizational culture that possesses the attributes of responsiveness, innovation, and problem solving necessary to convert strategic disadvantage into strategic advantage.
- Encourage employees to study statutes and regulations and explore innovative and alternative approaches that meet the statutory and regulatory intent.
- Rapid acquisition succeeds when senior leaders are involved in ensuring that programs are able to overcome the inevitable hurdles that arise during acquisition, and empower those responsible with achieving the right outcome with the authority to get the job done while minimizing the layers in between.

To help illustrate the continuity of the history of these issues and the lack of progress despite consistent, repeated similar findings, we consider the case of recommendations related to software capabilities of the acquisition workforce (areas where we are also recommending change).

Calls to improve DoD's ability to include software expertise in its workforce have a long history. DoD studies dating back to 1982 have raised concerns about the technical competencies and size of DoD's software workforce [DSB'82, DSB'87]. In 1993, the DoD Acquisition Management Board identified a need to review the DoD's software acquisition management education and training curricula. This study concluded that no existing DoD workforce functional management group was responsible for the software competencies needed in the workforce and that software acquisition competencies were needed in many different acquisition career fields. However, the Board asserted that no new career field was needed for Software Acquisition Managers.

In 2001, the same concerns regarding the software competencies of the DoD acquisition workforce once again surfaced. The DoD Software Intensive Systems Group conducted a software education and training survey of the acquisition workforce.[7] This survey demonstrated that less than 20 percent of the ACAT program staff had taken the basic Software Acquisition Management course (SAM 101) and that less than 20 percent of the ACAT program staff had degrees in computer science, software engineering, or information technology. The specific recommendations from this analysis included (1) instituting mandatory software-intensive systems training for the workforce; (2) developing a graduate-level program for software systems development and acquisition; and (3) requiring ACAT 1 programs to identify a chief software/ systems architect.

A year later, Congress mandated that the Secretary of each military department establish a program to improve the software acquisition processes of that military department.[8] Subsequently each Service established a strategic software improvement program (Army 2002, Air Force 2004, and Navy 2006). These Service initiatives have continued at some level. However, with the sunsetting of the Software Intensive Systems Group at the Office of the Secretary of Defense (OSD) level, the enterprise focus on software waned. During this same period, the Navy started the Software Process Improvement Initiative (SPII), which identified issues preventing software-intensive projects from meeting schedule, cost, and performance goals. This initiative highlighted the lack of adequately educated and trained software acquisition professionals and systems engineers.

In 2007, OSD issued guidance to create the Software Acquisition Training and Education Working Group (SATEWG) with a charter to affirm required software competencies, identify gaps in Defense Acquisition Workforce Improvement Act (DAWIA) career fields, and develop a plan to address those gaps. This group was composed of representatives from the Services, OSD, and other organizations, including the SEI. The group developed a software competency framework that identified four key knowledge areas and 29 competencies that could inform the different acquisition workforce managers about the software competencies to be integrated into their existing career field competency models. There has been no follow-on effort to evaluate the progress of the SATEWG or its outcomes.

Today, in the absence of a DoD-wide approach to describing, managing, and setting goals against a common understanding of needed software skills, each Service (as well as each software sustainment organization) has evolved its own approach or model for identifying software competencies for its workforce.

This historical context highlights two key points. First, DoD has long recognized the challenges of addressing the technical competencies and size of the software workforce across the life cycle. However, there is limited evidence of the outcomes from these different efforts. Second, this history clearly indicates that acquiring software human capital and equipping that workforce with the necessary competencies are persistent and dynamic challenges that demand a continuous enterprise strategy.

[7] Dennis Goldenson, & Matthew Fisher, *Improving the Acquisition of Software Intensive Systems* (CMU/SEI-2000-TR-003), (Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2000), http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=5171.
8 Public Law 107-314, Section 804, 2 December 2002, https://www.govinfo.gov/content/pkg/PLAW-107publ314/html/PLAW-107publ314.htm.

3.2 Breaking the Spell: Why Nothing Happened Before, but Why This Time Could Be Different

Given the long and profound history of inaction on past studies, we have attempted to create our own "Theory of (Non)Change." Why does the Department struggle to step up to rational, generally agreed-upon change? We offer the following three drivers:

The (Patriotic and Dutifully) Frozen Middle. Our process in executing this study has been to talk to anyone and everyone we could within various departments of DoD and the Services, to gather as many different perspectives as possible on what is needed, and to find out what is working and what needs to be stomped upon. As with many change management opportunities, we find significant top-down support for what we are trying to do, especially from those who see the immediate need for more, better, faster mission capability and those at the command level who are directly frustrated by the current processes that are just not working. At the other end, we see digital natives demanding change but with limited power to make it happen—people who are fully enmeshed in how the tech world works, people who have all the expectations that have been created by their private-sector lifestyle and economy. And then we have *the middle,* who are dutifully following the rules and have been trained and had success defined for a different world. For *the middle,* new methodologies and approaches introduce unknown risks, while the old acquisition and development approaches built the world's best military. We question neither the integrity nor the patriotism of this group. They are simply not incentivized to the way we believe modern software should be acquired and implemented, and the enormous inertia they represent is a profound barrier to change.

Unrequited Congress. Congress is responsible for approving and overseeing DoD's development programs. While it is clear that Congress takes its oversight role seriously, it does so knowing that to have oversight requires something to oversee, and it understands its fundamental responsibility is to enable the Department to execute its mission. But oversight matters, and recommendations for change that do not also provide insight into how new ways of doing things will allow Congress to perform its role are a very tough sell. In addition, there is a sense of unrequited return from past changes and legislation such as Other Transaction Authorities (OTAs), pilot programs, and special hiring authorities. In many cases, Congress believes it has already provided the tools and flexibilities for which DoD has asked. It is perhaps unreasonable to expect a positive response to ask for more when current opportunities have not been fully exploited.

Optimized Acquisition (for Something Else!).

> *Knowing was a barrier which prevented learning. — Frank Herbert*

While some may (justifiably) argue that the current acquisition system is not optimized for anything, it is the product of decades of rules upon rules, designed to speak to each and every edge case that might crop up in the delivery of decades-long hardware systems, holds risk elimination at a premium, and has a vast cadre of dedicated practitioners exquisitely trained to prosper within that system. This is a massive barrier to change and informs our recommendations that argue for major new ways of acquiring software and not just attempt to re-optimize to a different local maximum.

What we are trying to do that we think is different. Given the long history of DoD and Congressional reports that make recommendations that are not implemented, why do we think that this report will be any different? Our approach has been to focus not on the report and its recommendations *per se*, but rather on the series of discussions around the ideas in this report and the people we have interacted with inside the Pentagon and at program site visits. The recommendations in this report thus serve primarily as documentation of a sequence of iterative conversations, and the real work of the study is the engagements before and after the report is released.

We also believe that there are some ideas in the report that, while articulated in many places in different ways, are emphasized differently here. In particular, a key point of focus in this report is the use of speed and cycle time as the key drivers for must change and the need to optimize statutes, regulations, and processes to allow management and oversight of software. We believe that optimizing for the speed at which software can be utilized for competitive advantage will create an acquisition system that is much better able to provide security, insight, and scale.

Finally, we have tried to make this report shorter and pithier than previous reports, so we hope people will read it. It also is staged so that each reader, with his or her specific levels of authority and responsibility, can navigate an efficient path to reaching his or her own conclusions on how best to support what is contained here.

3.3 Consequences of Inaction: Increasing Our Attack Surface and Shifting Risk to the Warfighter

So what happens if history does, in fact, repeat itself and we again fail to step up to the changes that have been so clearly articulated for so long? Certainly by continuing to follow acquisition processes designed to limit risk for the hardware age, we will not reduce risk but instead will simply transfer that risk to the worst possible place—the warfighter who most needs the tools in her arsenal to deliver the missions we ask her to perform. But in addition, as we have continually stressed throughout this study, there are several real differences in today's world compared to the environment in which past efforts were made.

First, and most important, weapon systems, and the bulk of the operational structure on which DoD executes its mission, are now fundamentally software (or software-defined) systems, and as such, delays in implementing change amplify the capability gaps that slow, poor, or unsupportable software creates. Second, the astonishing growth of the tech sector has created a very different competitive environment for the talent most needed to meet DoD's needs. Decades ago, DoD was the leading edge of the world's coolest technology, and passionate, skilled software specialists jumped at the chance to be at that edge. That is simply not the case today, and while a commitment to national security is a strong motivator, if the changes recommended in this study are not implemented, the competitive war for talent, *within our country,* will be lost.

The modern software methodologies enumerated in this report—and the recommendations concerning culture, regulation and statute, and career trajectories that enable those methodologies—are the best path to providing secure, effective, and efficient software to users.

Cyber assurance, resilience, and relevance are all delivered much more effectively when done quickly and incrementally, using the tools and methods recommended in this study.

Finally we call attention back to Section 1.4 (What are the challenges that we face [and consequences of inaction]?). To summarize: “The long-term consequence of inaction is that our adversaries’ software capabilities can catch and surpass ours. … Our adversaries’ software capabilities are growing as ours are stagnating.”

Chapter 4. How Do We Get There from Here: Three Paths for Moving Forward

The history of technology is the story of man and tool-hand and mind-working together. If the hardware is faulty or if the software is deficient, the sounds that emerge will be discordant; but when man and machine work together, they can make some beautiful music.

— Melvin Kranzberg, Technology and History: Kranzberg's Laws, (Technology and Culture, 27[3]:1986), 558

The previous three chapters provided the rationale for why we need to *do* (not just say) something different about how DoD develops, procures, assures, deploys, and continuously improves software in support of defense systems. The private sector has figured out ways to use software to accelerate their businesses and DoD should accelerate its incorporation of those techniques to its own benefit, especially in ensuring that its warfighters have the tools they need in a timely fashion to execute their missions in today's hardware-enabled, software-defined environment. In this chapter, we lay out three different paths for moving forward, each under a different set of assumptions and objectives. A list of some representative, high-level steps is provided for each path, along with a short analysis of advantages and weaknesses.

4.1 Path 1: Make the Best of What We've Got

Congress has provided DoD with substantial authority and flexibility to implement the mission of the Department. Although difficult and often inefficient, it is possible to implement the recommendations outlined in this report making use of the existing authorities and, indeed, there are already examples of the types of activities that we envision taking place across OSD and the Services. In this section, we attempt to articulate a path that builds on these successes and does not require any change in the law nor major changes in regulatory structure. The primary steps required to implement this path should focus on changing the practices by which software is developed, procured, assured, and deployed as well as updating some of the regulations and processes to facilitate cultural and operational changes.

To embark on this first path, DoD should streamline its processes, allowing more rapid procurement, deployment, and updating of software. OSD and the Services should also work together to allow better cross-service and pre-certified Authorization to Operate (ATO), easier access to large-scale cloud computing, and use of modern toolchains that will benefit the entire software ecosystem. The acquisition workforce, both within OSD and the Services, should be provided with better training and insight on modern software development (one of the more frequent recommendations over the past 37 years) so that they can take advantage of the approaches that software allows that are different than hardware. Most importantly, government and industry must come together to implement a DevSecOps culture and approach to software, building on practices that are already known and used in industry.

The following list provides a summary of high-level steps that require changes to DoD culture and processes, but could be taken with no change in current law and relatively minor changes to existing regulations:

- Make use of existing authorities such as OTAs and mid-tier acquisition (Sec 804) to implement a DevSecOps approach to acquisition to the greatest extent possible under existing statutes, regulations, and processes.
- Require cost assessment and performance estimates for software programs (and software components of larger programs) to be based on metrics that track speed and cycle time, security, code quality, and useful capability delivered to end users.
- Create a mechanism for ATO reciprocity between Services and industrial base companies to enable sharing of software platforms, components, and infrastructure and rapid integration of capabilities across (hardware) platforms, (weapons) systems, and Services.
- Remove obstacles to DoD usage of cloud computing on commercial platforms, including Defense Information System Agency (DISA) cloud access point (CAP) limits, lack of ATO reciprocity, and access to modern software development tools.
- Expand the use of (specialized) training programs for chief information officers (CIOs), Service acquisition executives (SAEs), program executive officers (PEOs), and program managers (PMs) that provide (hands-on) insight into modern software development (e.g., Agile, DevOps, DevSecOps) and the authorities available to enable rapid acquisition of software.
- Increase the knowledge, expertise, and flexibility in program offices related to modern software development practices to improve the ability of program offices to take advantage of software-centric approaches to acquisition.
- Require access to source code, software frameworks, and development toolchains, with appropriate intellectual property (IP) rights, for all DoD-specific code, enabling full security testing and rebuilding of binaries from source.
- Create and use automatically generated, continuously available metrics that emphasize speed, cycle time, security, and code quality to assess, manage, and terminate software programs (and software components of hardware programs).
- Shift the approach for acquisition (and development) of software (and software-intensive components of larger programs) to an iterative approach: start small, be iterative, and build on success—or be terminated quickly.
- Make security a first-order consideration for all software-intensive systems, recognizing that security-at-the-perimeter is not enough.
- Shift from a list of requirements for software to a list of desired features and required interfaces/characteristics to avoid requirements creep or overly ambitious requirements.
- Maintain an active research portfolio into next-generation software methodologies and tools, including the integration of ML and AI into software development, cost estimation, security vulnerabilities, and related areas.
- Invest in transition of emerging approaches from academia and industry to creating, analysis, verification, and testing of software into DoD practice (via pilots, field tests, and other mechanisms).

- Automatically collect all data from DoD weapon systems and make the data available for machine learning (via federated, secured enclaves, not a centralized repository).
- Mandate a full program review within the first 6–12 months of development to determine if a program is on track, requires corrective action, or deserves cancellation.

This path has the advantage that the authorities required to undertake it are already in place and the expertise exists within the Department to begin moving forward. We believe that the there is strong support for these activities at the top and bottom of the system, and several groups (e.g., the Defense Digital Service [DDS], the Joint Improvised Threat Defeat Organization [JIDO], and Kessel Run) have demonstrated that the flexibilities exist within the current system to develop, procure, assure, deploy, and update software more quickly. The difficulty in this path is that it requires individuals to figure out how to go beyond the default approaches that are built into the current acquisition system. Current statutes, regulations, and processes are very complicated; there is a "culture of no" that must be overcome; and hence using the authorities that are available requires substantial time, effort, and risk (to one's career, if not successful). The risk in pursuing this path is that change occurs too slowly or not at scale, and we are left with old software that is vulnerable and cannot serve our needs. Our adversaries have the same opportunities that we do for taking advantage of software and may be able to move more quickly if the current system is left in place.

4.2 Path 2: Tune the Defense Acquisition System to Optimize for Software

While the first steps to refactoring the defense acquisition system can be taken without necessarily having to change regulations, the reality of the current situation is that Congress and DoD have created a massive "spaghetti code" of laws and regulations that are simply slowing things down. This might be OK for some types of long-development, long-duration hardware, but as we have articulated in the previous three chapters it is definitely not OK for (most types of) software.

This path takes a more active approach to modifying the acquisition system for software by identifying those statutes, regulations, and processes that are creating the worst bottlenecks and modifying them to allow for faster delivery of software to the field. We see this path as one of removing old pieces of code (statutory, regulatory, or process) that are no longer needed or that should not be applied to software, as well as increasing the expertise in how modern software development works so that software programs (and software-centric elements of larger programs) can be optimized for speed and cycle time.

The following list provides a set of high-level steps that require some additional changes to DoD culture and process, but also modest changes in current law and existing regulations. These steps build on the steps listed in path 1 above, although in some cases they can solve the problems that the previous actions were trying to work around.

- Refactor and simplify Title 10 and the defense acquisition system to remove all statutory, regulatory, and procedural requirements that generate delays for acquisition, development, and fielding of software while adding requirements for continuous (automated) reporting of cost, performance (against updated metrics), and schedule.

- Create streamlined authorization and appropriation processes for defense business systems (DBS) that use commercially available products with minimal (source code) modification.
- Plan, budget, fund, and manage software development as an enduring capability that crosses program elements and funding categories, removing cost and schedule triggers that force categorization into hardware-oriented regulations and processes.
- Replace the Joint Capabilities Integration and Development System (JCIDS), the Planning, Programming, Budgeting and Execution (PPB&E) process, and the Defense Federal Acquisition Regulation Supplement (DFARS) with a portfolio management approach to software programs, assigned to "PEO Digital" or an equivalent office in each Service that uses direct identification of warfighter needs to decide on allocation priorities.
- Create, implement, support, and require a fully automatable approach to T&E, including security, that allows high-confidence distribution of software to the field on an iterative basis (with frequency dependent on type of software, but targeting cycle times measured in weeks).
- Prioritize secure, iterative, collaborative development for selection and execution of all new software programs (and software components of hardware programs) (see DIB's Detecting Agile BS as an initial view of how to evaluate capability).
- For any software developed for DoD, require that software development be separated from hardware in a manner that allows new entrants to bid for software elements of the program on the basis of demonstrated capability.
- Shift from certification of executables, to certification of code, to certification of the development, integration, and deployment toolchain, with the goal of enabling rapid fielding of mission-critical code at high levels of information assurance.
- Require CIOs, SAEs, PEOs, PMs, and any other acquisition roles involving software development as part of the program to have prior experience in software development.
- Restructure the approach to recruiting software developers to assume that the average tenure of a talented engineer will be 2–4 years, and make better use of highly qualified experts (HQEs), intergovernmental personnel act employees (IPAs), reservists, and enlisted personnel to provide organic software development capability.
- Establish a Combat Digital Service (CDS) unit within each Combatant Command (COCOM) consisting of software development talent that can be used to manage Command-specific IT assets, at the discretion of the combatant commander. DDS, operating at the OSD level, is a good model for what a CDS can do for each COCOM.

Pursuing this path will allow faster updates to software and will improve security and oversight (via increased insight). In many cases, the Department is already executing some of the actions required to enable this path. The weakness in this path is that software would generally use the same basic approach to acquisition as hardware, with various carve-outs and exceptions. This approach runs the risk that software programs still move too slowly due to the large number of people who have to say yes and the need to train a very large acquisition force to understand how software is different than hardware (and not all software is the same).

4.3 Path 3: A New Acquisition Pathway and Appropriations Category for Software to Force Change in the Middle

The final path is the most difficult and will require dozens of independent groups to agree on a common direction, approach, and set of actions. At the end of this path lies a new defense acquisition system that is optimized for software-centric systems instead of hardware-centric systems and that prioritizes security, speed, and cycle time over cost, schedule, and (rigid) requirements.

To undertake this path, Congress and OSD must write new statutes and regulations for software, providing increased (and automation-enabled) insight to reduce the risk of slow, costly, and overgrown programs and enabling rapid deployment and continuous improvement of software to the field. Laws will have to be changed, and management and oversight will have to be reinvented, focusing on different measures and a quicker cadence. OSD and the Services will need to create and maintain interoperable (cross-program/cross-Service) digital infrastructure that enables rapid deployment, scaling, testing, and optimization of software as an enduring capability; manage it using modern development methods; and eliminate the existing hardware-centric regulations and other barriers for software (and software-intensive) programs. Finally, the Services will need to establish software development as a high-visibility, high-priority career track with specialized recruiting, education, promotion, organization, incentives, and salary.

The following list of high-level steps are required to pursue this path, builds on the steps listed in the previous paths:

- Establish one or more new acquisition pathways for software that prioritize continuous integration and delivery of working software in a secure manner, with continuous oversight from automated analytics.
- Create a new appropriations category that allows (relevant types of) software to be funded as a single budget item, with no separation between RDT&E, production, and sustainment.
- Establish and maintain digital infrastructure within each Service or Agency that enables rapid deployment of secure software to the field, and incentivize its use by contractors.
- Plan and fund computing hardware (of all types) as consumable resources, with continuous refresh and upgrades to the most recent, most secure operating system and platform components.
- Create software development groups in each Service consisting of military and/or civilian personnel who write code that is used in the field, and track individuals who serve in these groups for future DoD leadership roles.

This path attempts to solve the longstanding issues with software by creating an acquisition pathway and an appropriations category that are fine-tuned for software. It will require a very large effort to get the regulations, processes, and people in place that are required to execute it effectively, and there will be missteps along the way that generate controversy and unwanted publicity. In addition, it will likely be opposed by those currently in control of selling or making software for DoD, since it will require that they retool their business to a very new approach that is not well defined at the outset. But if successful, this path has the potential to enable DoD to develop, procure, assure, deploy, and continuously improve software at a pace that is relevant for modern missions and builds on the substantial success of the U.S. private sector.

Chapter 5. What Would the DIB Do: Recommendations for Congress and DoD

It takes a lot of hard work to make something simple, to truly understand the underlying challenges and come up with elegant solutions.

— Steve Jobs as quoted by Walter Isaacson, "How Steve Jobs' Love of Simplicity Fueled a Design Revolution," (Smithsonian Magazine, September 2012)

In this final chapter we lay out our recommendations for what Congress and DoD should do to implement the type of software acquisition and practices reform that we believe is needed for the future. Our recommendations are organized according to four lines of effort, each of which bring together different parts of the defense ecosystem as stakeholders:

A. Congress and OSD should refactor statutes, regulations, and processes for software
B. OSD and the Services should create and maintain cross-program/cross-Service digital infrastructure
C. The Services and OSD should create new paths for digital talent (especially *internal* talent)
D. DoD and industry must change the practice of how software is procured and developed

For each of these lines of effort, we have identified the 2–3 most important recommendations that we believe Congress and DoD should undertake. These "Top Ten" primary recommendations were chosen not because they solve the entire problem but because they will make the biggest difference; without them, substantial change is not likely. In addition, we have identified 16 additional recommendations for consideration once the execution of the first 10 recommendations is successfully underway. For each recommendation, a draft implementation plan is provided in Appendix A that gives a list of actions that can be used to implement the recommendation, as well as more detail on the rationale, supporting information, and similar recommendations from other studies. Potential legislative and regulatory language to implement selected recommendations is included in Appendix B. While we have tried hard to provide specific actions, owners, and target dates that will drive an implementation plan for each recommendation, we recognize that in the end, owners will be decided by the Department's response to our study and owners will use our actions as a starting point to their own implementation plans.

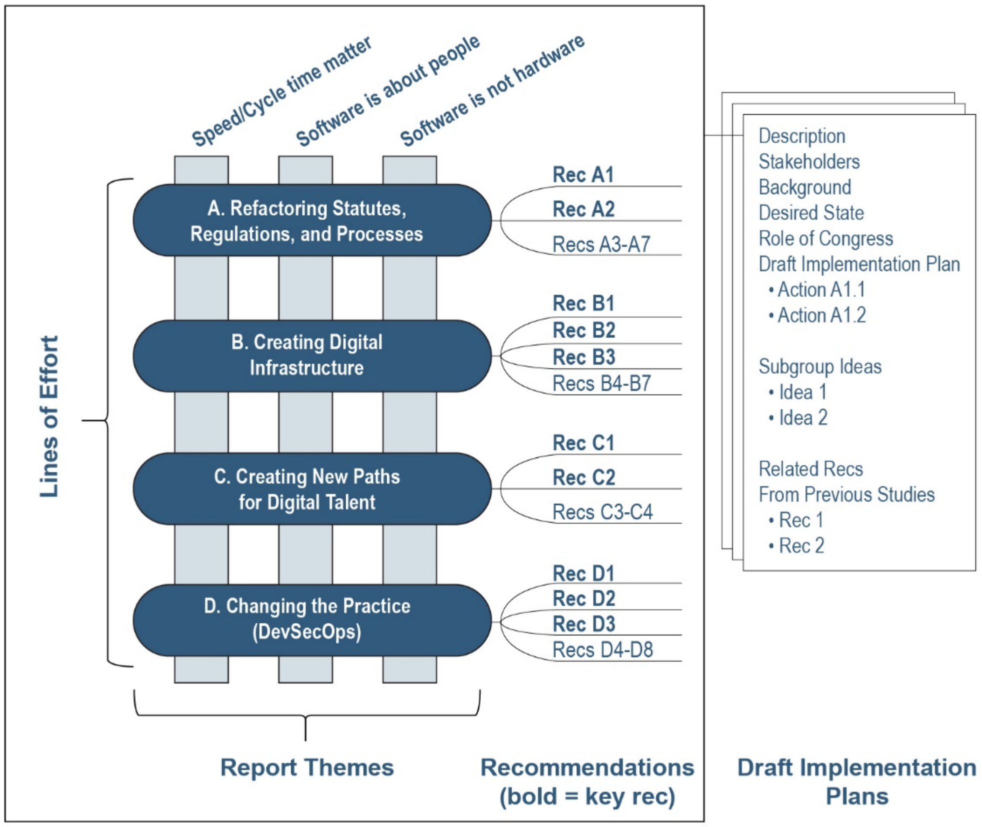


Figure 5.1 Recommendation structure. For each line of effort, a set of primary recommendations (bold) is provided, along with a set of additional recommendations for consideration. Each recommendation contains a draft implementation plan that includes background information on the rationale, vision, and stakeholders.

5.1 The Ten Most Important Things to Do (Starting Now!)

In this section we lay out what we believe are the most important steps for Congress and DoD to take to fully leverage the opportunities presented by software and the private sector's strength in modern development practices. Our commitment to these steps will directly impact the Department's ability to achieve the 2018 National Defense Strategy[9] goals of increased lethality, stronger alliances while positioning for new partnerships, and reformed business practices for better performance and affordability.

[9] U.S. Department of Defense, *Summary of the 2018 National Defense Strategy: Strengthening the American Military's Competitive Edge*, (Washington, DC: U.S. Department of Defense), https://dod.defense.gov/Portals/1/Documents/pubs/2018-National-Defense-Strategy-Summary.pdf.

Line of Effort A. Congress and OSD should refactor statutes, regulations, and processes for software, providing increased insight to reduce the risk of slow, costly, and overgrown programs and enabling rapid deployment and continuous improvement of software to the field. Reinvent management and oversight, focusing on different measures and a quicker cadence.

Figure 5.2. The West Front of the U.S. Capitol. [Photo by Architect of the Capitol]

Recommendation A1. Establish one or more new acquisition pathways for software that prioritize continuous integration and delivery of working software in a secure manner, with continuous oversight from automated analytics

Current law, regulation, policy, and internal DoD processes make DevSecOps-based software development extremely difficult, requiring substantial and consistent senior leadership involvement. Consequently, DoD is challenged in its ability to scale DevSecOps software development practices to meet mission needs. The desired state is that programs have the ability to rapidly field and iterate new functionality in a secure manner, with continuous oversight based on automated reporting and analytics, and utilize IA-accredited commercial development tools.

Implementation of this recommendation could be accomplished by having USD(A&S), in coordination with USD(C) and Cost Assessment and Program Evaluation (CAPE), submit a legislative proposal using Sec 805 to propose new acquisition pathways for two or more classes of software (e.g., application, embedded), optimized for DevSecOps, for approval by the House and Senate Armed Services Committees. A draft of such language, in response to 2016 NDAA Section 805, is included in Appendix B. If approved, USD(A&S) could develop and issue a Directive-Type Memorandum (DTM) for new software acquisition pathways, and the SAEs could issue Service-level guidance for new acquisition pathways. USD(A&S), with SAEs, should select an initial set of programs that are using DevSecOps to convert to or utilize the new software acquisition pathways at the same time as developing and implementing training at Defense Acquisition University (DAU) on new software acquisition pathways for all acquisition communities (FM, Costing, PM, IT, SE, etc.). As the pathways become better understood, the DTM can be converted to a DoD Instruction (5000.SW?), incorporating lessons learned during initial program implementation.

This recommendation is supported by the ideas for change listed by the Acquisition & Strategy subgroup and is aligned with the recommendations of the 1987 and 2009 DSB studies.

Recommendation A2. Create a new appropriation category for software capability delivery that allows (relevant types of) software to be funded as a single budget item, with no separation between RDT&E, production, and sustainment

Current law, regulation, and policy treat software acquisition as a series of discrete sequential steps; accounting guidance treats software as a depreciating asset. These processes are at odds with software being continuously updated to add new functionality and create significant delays in fielding user-needed capability. The desired state is the establishment of a new appropriation (major force program category) so that programs are better able to prioritize how effort is spent on new capabilities versus fixing bugs/vulnerabilities, improving existing capabilities, etc. Such prioritization can be made based on warfighter/user needs, changing mission profiles, and other external drivers, not constrained by available sources of funding.

Implementation of this recommendation could be accomplished by having USD(A&S) submit a legislative proposal to create a new appropriations category for software and software-intensive programs for approval by the House and Senate Armed Services Committees and funding by the House and Senate Appropriations Committees. A draft of such language, linked to the acquisition pathway described in Recommendation A1, is included in Appendix B. The DoD Comptroller, working with CAPE, would need to make necessary modifications in supporting PPB&E systems to allow use and tracking of the new software appropriation. USD(A&S), in coordination with the SAEs, should select the initial programs that will use the new software appropriation from among those that are currently using DevSecOps-compatible development approaches. Budget exhibits for the new software appropriation, replacing the current P-Forms and R-Forms, should be prepared by USD(A&S) working with USD(C), CAPE, and the Appropriations Committees, and those programs selected to use the new appropriation category should begin using the exhibits upon selection into the category (see Appendix C). Finally, the Federal Accounting Standards Advisory Board in coordination with USD(A&S) and USD(C) will need to change the audit treatment of software for this category to : (1) create a separate category for software instead of characterizing software as property, plant, and equipment; (2) establish a default setting that software is an expense, not an investment; and (3) ensure that “sustainment” is an integrated part of the software life cycle.

This recommendation builds on the recommendations in the DIB’s Ten Commandments of Software (at Appendix E) and our Visit Observations and Recommendations that budgets for software (and software-intensive) programs should support the full, iterative life cycle of the software. In addition, the Acquisition & Strategy, Appropriations, Contracting, and Sustainment & Modernization subgroups all had recommendations that support this approach. The basic approach advocated here was also articulated in the 1987 DSB task force on military software and Government Accountability Office (GAO) studies in 2015 and 2017, and is consistent with the Portfolio Management Framework Recommendations 41 and 42 of the Section 809 Panel.

Line of Effort B. OSD and the Services should create and maintain cross-program/ cross-Service digital infrastructure that enables rapid deployment, scaling, and optimization of software as an enduring capability, managed using modern development methods in place of existing (hardware-centric) regulations and providing more insight (and hence better oversight) for software-intensive programs.

Figure 5.3. Soldiers review the Army's Command Post Computing Environment, a software system that consolidates tools, programs, and tasks into an integrated, interoperable, and cybersecure computing infrastructure framework. [U.S. Army photo by Dan Lafontaine, PEO C3T]

Recommendation B1. Establish and maintain digital infrastructure within each Service or Agency that enables rapid deployment of secure software to the field, and incentivize its use by contractors

Currently, each DoD program develops its own development and test environments, which requires redundant definition and provisioning, replicated assurance (including cyber), and extended lead times to deploy capability. Small companies have difficulties providing software solutions to DoD because those software and development test environments are not available outside the incumbent contractor or they have to build (and certify) unique infrastructure from scratch. The desired state is that defense programs will have access to, and be stakeholders in, a cross-program, modern digital infrastructure that can benefit from centralized support and provisioning to lower overall costs and the burden for each program. Development infrastructure supporting continuous integration/continuous delivery (CI/CD) and DevSecOps is available as best-of-breed, and government off-the-shelf (GOTS) is provided so that contractors want to use it, though DoD programs or organizations that want or need to go outside that existing infrastructure can still do so.

Recommendation B2. Create, implement, support, and use fully automatable approaches to testing and evaluation (T&E), including security, that allow high-confidence distribution of software to the field on an iterative basis

To deliver software at speed, rigorous, automated testing processes and workflows are essential. Current DoD practices and procedures often see operational test and evaluation (OT&E) as a tailgate process, sequentially after development has been completed, slowing down delivery of useful software to the field and leaving existing (potentially poorly performing and/or vulnerable) software in place. The desired state is that development systems, infrastructure, and practices are focused on continuous, automated testing by developers (with users). To the maximum extent possible, system operational testing is integrated (and automated) as part of the development cycle using data, information, and test protocols delivered as part of the development environment. Testing and evaluation/certification of COTS components occurs once (if justified), and then ATO reciprocity (Rec B3) is applied to enable use in other programs, as appropriate.

Recommendation B3. Create a mechanism for Authorization to Operate (ATO) reciprocity within and between programs, Services, and other DoD agencies to enable sharing of software platforms, components, and infrastructure and rapid integration of capabilities across (hardware) platforms, (weapon) systems, and Services

Current software acquisition practice emphasizes the differences among programs: perceptions around different missions, different threats, and different levels of risk tolerance mean that components, tools, and infrastructure that have been given permission to be used in one context are rarely accepted for use in another. The lack of ATO reciprocity drives each program to create its own infrastructure, repeating time- and effort-intensive activities needed to certify elements as secure for their own specific context. The desired state is that modern software components, tools, and infrastructure, once accredited as secure within the DoD, can be used appropriately and cost-effectively by multiple programs. Programs can then spend a greater percentage of their budgets on developing software that adds value to the mission rather than spending time and effort on basic software infrastructure. COTS components are accredited once and then made available for use in other programs, as appropriate.

Line of Effort C. The Services and OSD should create new paths for digital talent (especially internal talent) by establishing software development as a high-visibility, high-priority career track and increasing the level of understanding of modern software within the acquisition workforce. Increased internal capability is necessary both to allow organic (internal) development and to enable the Department to best serve as a knowledgeable partner for software acquired from commercial sources.

Figure 5.4. Airmen assigned to the 707th Communications Squadron, which supports more than 5,700 personnel around the world, update software for Air Force networks. [U.S. Navy photo by Rick Naystatt/Released]

Recommendation C1. Create software development units in each Service consisting of military and civilian personnel who develop and deploy software to the field using DevSecOps practices

DoD’s capacity to apply modern technology and software practices to meet its mission is required to remain relevant in increasingly technical fighting domains, especially against peer adversaries. While DoD has both military and civilian software engineers (often associated with maintenance activities), the IT career field suffers from a lack of visibility and support. The Department has not prioritized a viable recruiting strategy for technical positions, and has no comprehensive training or development program that prepares the technical and acquisition workforce to adequately deploy modern software development tools and methodologies. The desired state is that DoD recruits, trains, and retains internal capability for software development, including by Service Members, and maintains this as a separate career track (like DoD doctors, lawyers, and musicians). Each Service has organic development units that are able to create software for specific needs and that serve as an entry point for software development capability in military and civilian roles (complementing work done by contractors). The Department’s workforce embraces commercial best practices for the rapid recruitment of talented professionals, including the ability to onboard quickly and provide modern tools and training in state-of-the-art training environments. Individuals in software development career paths are able to maintain their technical skills and take on DoD leadership roles.

Recommendation C2. Expand the use of (specialized) training programs for CIOs, SAEs, PEOs, and PMs that provide (hands-on) insight into modern software development (e.g., Agile, DevOps, DevSecOps) and the authorities available to enable rapid acquisition of software

Acquisition professionals have been trained and had success in the current model, which has produced the world’s best military, but this model does not serve well for software. New methodologies and approaches introduce unknown risks, and acquisition professionals are often not incentivized to make use of the authorities available to implement modern software methods. At the same time, senior leaders in DoD need to be more knowledgeable about modern software development practices so they can recognize, encourage, and champion efforts to implement modern approaches to software program management. The desired state is that senior leaders, middle management, and organic and contractor-based software developers are aligned in their view of how modern software is procured and developed. Acquisition professionals are aware of all of the authorities available for software programs and use them to provide flexibility and rapid delivery of capability to the field. Program leaders are able to assess the status of software (and software-intensive) programs and spot problems early in the development process, as well as provide continuous insight to senior leadership and Congress. Highly specialized requirements are scrutinized to avoid developing custom software when commercial offerings are available that are less expensive and more capable.

Line of Effort D. DoD and industry must change the practice of how software is procured and developed by adopting modern software development approaches, prioritizing speed as the critical metric, ensuring cybersecurity is an integrated element of the entire software life cycle, and purchasing existing commercial software whenever possible.

Figure 5.5. Connected battle command suites. [U.S. Army photo]

Recommendation D1. Require access to source code, software frameworks, and development toolchains—with appropriate IP rights—for all DoD-specific code, enabling full security testing and rebuilding of binaries from source

Source code for many DoD systems is not available to DoD for inspection and testing, and DoD relies on suppliers to write code for new compute environments. As code ages, suppliers are not required to maintain codebases without an active development contract, and "legacy" code is not continuously migrated to the latest hardware and operating systems. The desired state is that DoD has access to source code for DoD-specific software systems that it operates and uses to perform detailed (and automated) evaluation of software correctness, security, and performance, enabling more rapid deployment of both initial software releases and (most important) upgrades (patches and enhancements). DoD is able to rebuild executables from scratch for all of its systems and has the rights and ability to modify (DoD-specific) code when new conditions and features arise. Code is routinely migrated to the latest computing hardware and operating systems, and routinely scanned against currently known vulnerabilities. Modern IP language is used to ensure that the government can use, scan, rebuild, and extend purpose-built code, but contractors are able to use licensing agreements that protect any IP that they have developed with their own resources. Industry trusts DoD with its code and has appropriate IP rights for internally developed code.

Recommendation D2. Make security a first-order consideration for all software-intensive systems, recognizing that security-at-the-perimeter is not enough

Current DoD systems often rely on security-at-the-perimeter as a means of protecting code from unauthorized access. If this perimeter is breached, then a large array of systems can be compromised. Multiple reports by the GAO, the Department of Defense Office of Inspector General (DoDIG), and other agencies have identified cybersecurity as a major issue in acquisition programs. The desired future state is that DoD systems use a zero-trust security model in which it is not assumed that anyone who can gain access to a given network or system should have access to anything within that system. DoD uses regular and automated penetration testing to track down vulnerabilities, and engages red teams to attempt to breach our systems before our adversaries do.

Recommendation D3. Shift from the use of rigid lists of requirements for software programs to a list of desired features and required interfaces/characteristics to avoid requirements creep, overly ambitious requirements, and program delays

Current DoD requirements processes significantly impede its ability to implement modern software development practices by forcing programs to spend years establishing requirements and insisting on satisfaction of requirements before a project is considered "done." This impedes rapid implementation of features that are of greatest value to the user. The desired state is that rather than a list of requirements for every feature, programs should establish a minimum set of requirements required for initial operation, security, and interoperability, and place all other desired features on a list that will be implemented in priority order, with the ability for DoD to redefine priorities on a regular basis.

5.2 The Next Most Important Things to Tackle

DoD must make a large number of changes to fully realize the vision that 37 years of studies have articulated. This study solicited input from a wide range of stakeholders in the defense software enterprise, including OSD and Service leaders, industry participants in our visits and roundtables, and FFRDC personnel who helped put together our report and identify the recommendations that we should make. The list of recommendations below are the next 0x10 (16) recommendations that we believe can be implemented after actions on the 10 above are solidly underway (like software, implementing recommendations is never "done"). We list these second not because they are dependent on the primary recommendations but simply to emphasize the urgency of the Top Ten.

ID	Recommendation
A3	Require cost assessment and performance estimates for software programs (and software components of larger programs) of appropriate type be based on metrics that track speed and cycle time, security, code quality, and functionality
A4	Refactor and simplify Title 10, DFARS, and DoDI 5000.02/5000.75 to remove statutory, regulatory, and procedural requirements that generate delays for acquisition, development, and fielding of software; while adding requirements for continuous (automated) reporting of cost, performance (against updated metrics), and schedule
A5	Create streamlined authorization and appropriation processes for defense business systems (DBS) that use commercially available products with minimal (source code) modification
A6	Plan, budget, fund, and manage software development as an enduring capability that crosses program elements and funding categories, removing cost and schedule triggers associated with hardware-focused regulations and processes
A7	Replace JCIDS, PPB&E, and DFARS with a portfolio management approach to software programs, assigned to "PEO Digital" or an equivalent office in each Service that uses direct identification of warfighter needs to determine allocation priorities for software capabilities

B4	Prioritize secure, iterative, collaborative development for selection and execution of new software development programs (and software components of hardware programs), especially those using commodity hardware and operating systems
B5	Remove obstacles to DoD usage of cloud computing on commercial platforms, including DISA CAP limits, lack of ATO reciprocity, and access to modern software development tools
B6	Shift from certification of executables for low- and medium-risk deployments to certification of code/architectures and certification of the development, integration, and deployment toolchain
B7	Plan and fund computing hardware (of all appropriate types) as consumable resources, with continuous refresh and upgrades to current, secure operating systems and platform components
C3	Increase the knowledge, expertise, and flexibility in program offices related to modern software development practices to improve the ability of program offices to take advantage of software-centric approaches to acquisition
C4	Restructure the approach to recruiting digital talent to assume that the average tenure of a talented engineer will be 2–4 years, and make better use of HQEs, IPAs, special hiring authorities, reservists, and enlisted personnel to provide organic software development capability, while at the same time incentivizing and rewarding internal talent
D4	Create and use automatically generated, continuously available metrics that emphasize speed, cycle time, security, user value, and code quality to assess, manage, and terminate software programs (and software components of hardware programs)
D5	Shift the approach for acquisition and development of software (and software-intensive components of larger programs) to an iterative approach: start small, be iterative, and build on success—or be terminated quickly
D6	Maintain an active research portfolio into next-generation software methodologies and tools, including the integration of ML and AI into software development, cost estimation, security vulnerabilities, and related areas
D7	Invest in transition of emerging tools and methods from academia and industry for creating, analyzing, verifying, and testing of software into DoD practice (via pilots, field tests, and other mechanisms)
D8	Automatically collect all data from DoD national security systems, networks, and sensor systems, and make the data available for machine learning (via federated, secured enclaves, not a centralized repository).

5.3 Monitoring and Oversight of the Implementation Plan

It would be naive to believe that just listing the recommendations above will somehow ensure they are quickly and easily implemented after 37 years of previous, largely consistent recommendations have had relatively minor impact. We believe that DoD should use these recommendations (and the ones that preceded them) to create an implementation plan for review by stakeholders (including the DIB, if there is interest). This implementation plan might use as its starting point the proposed implementation plans that we have articulated in Appendix A, with agreement by the Secretary of Defense, the Undersecretaries of Defense, the Service Chiefs, CAPE, and DOT&E to support the creation and execution of the next iteration of the implementation plan.

We propose the following timeline for implementing the recommendations proposed here:

- (Immediately): Define, within 60 days after delivery of this report to Congress, a detailed implementation plan and assign owners to begin each of the top recommendations.

- FY19 (create): High-level endorsement of the vision of this report, and support for activities that are consistent with the desired end state (i.e., DevSecOps and enterprise-level architecture and infrastructure). Identify and launch programs to move out on the priority recommendations (start small, iterate quickly).
- FY20 (deploy): Initial deployment of authorities, budgets, and processes for reform of software acquisition and practices. Execute representative programs according to the main lines of effort and primary recommendations in this report. Implement these recommendations in the way we implement modern software: implement now, measure results, and modify approaches.
- FY21 (scale): Streamlined authorities, budgets, and processes enabling reform of software acquisition and practices at scale. In this time frame, adopt a new methodology to estimate as well as determine the value of software capability delivered (and not based on lines of code).
- FY22 (optimize): Conditions established so that all DoD software development projects transition (by choice) to software-enabled processes, with the talent and ecosystem in place for effective management and insight.

5.4 Kicking the Can Down the Road: Things That We Could Not Figure Out How to Fix

Despite the fairly comprehensive view that we have attempted to take in this study regarding how to improve the defense software enterprise, there are a number of challenges remaining that we were not able to address. We summarize these here for the next study (or perhaps one 37 years from now) to consider as DoD continues this path forward.

Over-oversight. DoD's sprawling software enterprise has many oversight actors, spanning Congress, OSD, Service or Component leadership, and other executive branch actors like the GAO. These actors each take frequent oversight action in attempts to improve the software in specific programs and also make well-intentioned efforts to improve the health of the overall system. However, these oversight actions focus primarily on addressing the behavior of the people developing and maintaining the software, overlooking the fact that the oversight itself is equally part of DoD's software problem. Ultimately, we cannot fix software without fixing oversight.

There are at least two categories of problems when it comes to software oversight: structural and substantive.

From a structural perspective, there are too many actors involved in oversight. A program manager, tasked with leading a software development effort, may have as many as 17 other actors who can take some form of oversight action on the program. Most of these individuals do not possess the authority to cancel a program unilaterally, but all have the ability to delay progress or create uncertainty while seeking corrective action for their concerns. These oversight actors often have overlapping or unclear roles and authorities, as well as competing interests and incentives. This means that in addition to the necessary checks and balances required between organizations, there is debate and active competition inside each of the organizations with, for example, various offices in OSD arguing among themselves in addition to arguing with Congress and the Services. Further, there is significant personnel turnover within these positions, meaning that any consensus tends to be short lived.

Substantively, the various oversight actors often do not possess a shared understanding of what constitutes good practice for software or its oversight. Further, these actors may not share a common vision for what DoD's software enterprise should look like today or in the future. The majority of oversight attention and action is placed on individual programs than on considering portfolios in the aggregate or the performance of the system as a whole. This program oversight is highly subjective in nature, relying on reports and PowerPoint slides presenting narratives and custom-created data. Worse, this oversight operates primarily according to conventional wisdom associated with the oversight of hardware programs, using decades-old heuristics when considering cost, schedule, and performance.

Without understanding what good looks like, or the right questions to ask, oversight actors risk enacting poor fixes. These actions can also be at odds with stated policy. Oversight actions are always more powerful than written policy, meaning that disparities between the two create the risk of cognitive dissonance or a shadow policy environment. Disparities also put program leadership in the unfair position of having to resolve the competing priorities of others, with the knowledge that failure to do so will lead to more blame and action from above.

Structural and substantive problems lead to oversight that is inconsistent and confusing, making it essentially impossible to systematically identify symptoms, determine root causes, or implement scalable fixes. This, in turn, allows everyone involved in DoD software development and maintenance to feel aggrieved, blame everyone other than themselves for systemic issues, and continue their behavior without reflection or change, thus perpetuating the cycle.

The approach by oversight organizations both on the Hill and in DoD should be that policy is treated as the current hypothesis for how best to ship code that DoD's users need. Through the use of data-driven governance, each program should then be tested against that policy while also being a test of the policy. The hypothesis, and policy, must be continually updated based on standard data that is recognized by, and accessible to, all oversight actors. Implementing such an approach is within the power of the oversight community but would be challenging and appears unlikely given current culture and practices. Regardless, those involved in the oversight of DoD software should not expect meaningfully improved outcomes for that software until the oversight practices used to improve that software are themselves improved.

Promotion practices. Software is disproportionately talent driven. Access to strong engineering talent is one of the most important factors that determine the success or failure of software projects. All that our rivals have to do to surpass us in national security applications of software such as AI, autonomy, or data analytics is to leverage their most talented software engineers to work on those applications. And yet in DoD, as much as we struggle to attract those with technical talent, we also struggle to elevate the talent we have.

The companies and institutions that are winning the software game recognize the importance of identifying and cultivating talented software leaders (whether they are engineers, managers, or strategists working closely with contractors) and actively promote and reward employees based on merit and demonstrated contributions. In contrast, human capital practices in DoD, sometimes by design and sometimes by habit and culture, narrowly limit how technical talent can be evaluated and often prioritize time in grade. The Department needs to figure out how to recognize when civilians and Service Members show an aptitude for software and software management and be able to promote, reward, and retain these individuals outside of the current constraints.

Using commercial software whenever possible. DoD should not build something that it can buy. If there is an 80 percent commercial solution, it is better to buy it and adjust—either the requirements or the product—rather than build it from scratch. It is generally not a good idea to over-optimize for what we view as "exceptional performance," because counter-intuitively this may be the wrong thing to optimize for as the threat environment evolves over time. Similarly, DoD should take actions to ensure that both the letter and spirit of commercial preference laws (e.g., 10 USC 2377, which requires defense agencies to give strong preference to commercial and non-developmental products) are being followed.

There is a myth that the U.S. private sector—where much of the world's software talent is concentrated—is unwilling to work on national security software. The reality is that DoD has failed to award meaningful government contracts to commercial software companies, which has generally led to companies making a *business decision* to avoid it. DoD's existing efforts to target the commercial software sector are governed by a "spray and pray" strategy, rather than by making concentrated investments.[10] DoD seems to love the idea of innovation, but does not love taking sizeable bets on new entrants or capabilities. It is interesting that Palantir and SpaceX are the only two examples since the end of the Cold War of venture-backed, DoD-focused businesses reaching multibillion dollar valuations. By contrast, China has minted around a dozen new multibillion dollar defense technology companies over the same time period. Some of these problems are purely cultural in nature and require no statutory/regulatory changes to address. Others likely will require the changes detailed in our recommendations.

That said, in many cases, there will not be an obvious "buy" option on the table. DoD and the Services should also work together to prioritize interoperable approaches to software and systems that enable rapid deployment, scaling, testing, and optimization of software as an enduring capability; manage them using modern development methods; and eliminate selected hardware-centric regulations and other particularly problematic barriers. The Services should find ways to better recognize software as a key area of expertise and provide specialized education and organizational structures that are better tuned for rapid insertion and continuous updates of software in the field and in the (back) office.

[10] While the overall funding commitments are large—$2 billion from DARPA for AI, for example—those commitments have resulted in few, if any, contracts for private companies other than traditional defense contractors. They have therefore failed to create significant incentives for the commercial tech sector to invest in government applications of AI.

Acknowledgments

The SWAP study members are indebted to a large number of individuals who helped provide valuable input, guidance, and support for the study and for the creation of this report.

We would first like to thank the SWAP study team, who coordinated the many activities associated with the study, including arranging for visits, briefings, and meetings; running the SWAP working group activities; and assisting with the production of the final report. Our initial study director, Bess Dopkeen, was detailed to the study from CAPE and provided outstanding leadership to the overall study. Her vision, energy, and knowledge of the Department were essential in establishing the interactive nature of this activity and helping us obtain insight into the many previously unknown aspects of DoD. She was succeeded by Jeff Boleng, the USD(A&S) Special Assistant for Software, who initially served as our liaison to A&S and took over as study director when Bess departed the Pentagon. Bess and Jeff were assisted by three outstanding members of the core SWAP team: Courtney Barno, Devon Hardy, and Sandra O'Dea. The study and the report could not have come together without the tireless (and patient!) efforts of Bess, Courtney, Devon, Sandy, and Jeff, who participated in every aspect of the report and helped us shape its content, style, and tone.

The SWAP study was also assisted by individuals from the Institute for Defense Analyses (IDA), SEI, and MITRE who served as our experts on the acquisition process and were invaluable in working through the detailed recommendations. Their knowledge of past studies, the acquisition regulations, the many novel approaches to acquisition reform, and the language of the acquisition community helped us better understand the challenges and opportunities for software acquisition and reform. We would particularly like to thank Kevin Garrison (IDA), Nick Guertin (SEI), Tamara Marshall-Keim (SEI), Forrest Shull (SEI), and Craig Ulsh (MITRE) for their help, encouragement, and constant advice.

A major element of the study was the participation of a large SWAP working group consisting of DoD employees who worked with Bess and the SWAP team to provide input to the study and to articulate pain points, ideas for changes, and proposed updates to legislation and regulations. A full list of individuals who participated in the working groups is listed in Appendix J, but we would particularly like to thank John Bergin, Ben FitzGerald, Bill Greenwalt, Amy Henninger, Paul Hullinger, Peter Levine, Melissa Naroski Merker, Jane Rathbun, Ed Wolski, and Philomena Zimmerman.

The Defense Innovation Board (DIB) staff were tightly linked to the SWAP study, which took place under the auspices of the Science and Technology (S&T) Committee. Josh Marcuse was instrumental in initiating the study (including identifying and hiring Bess) and providing keen insights into the report contents and recommendations. Mike Gable, Janet Boehnlein, and Christopher "Bruno" Brunett served as our designated federal officers (DFOs), accompanying us on trips, visits, and meetings and helping us uphold the Federal Advisory Committee Act (FACA) guidelines in a manner that enabled us to interact in a transparent and interactive way with members of the public, the Department, and Congress.

Many high-ranking officials within the Pentagon took the time to meet with us and provide their input, views, and encouragement for our efforts. Chief among these was Ellen Lord, Under Secretary of Defense for Acquisition & Sustainment, who provided input to our study and support for our meetings, while always being careful to help protect the independence of the study team in support of the charge from Congress. We would also like to thank Bob Daigle (CAPE), Dana Deasy (CIO), Bob Behler (DOT&E), Hondo Guerts (USN), and Will Roper (USAF) for their willingness to meet with us on multiple occasions.

Finally, we are indebted to the many individuals working on DoD programs with whom we met, both in industry and in government. On our many visits and in countless briefings, individuals who were working within the current system, and often pushing the boundaries of what is possible, gave us their honest insights and feedback. We are particularly grateful for the help we received from Tory Cuff, Leo Garciga, and CAPT Bryan Kroger, for their willingness to speak with us and help us understand what the future could look like.

SWAP Vignettes

To help illustrate some of the issues facing the Department in the area of software acquisition and practices, the SWAP study solicited a set of “vignettes” on different topics of relevance to the study. These vignettes represent “user stories” contributed by study team members and collaborators; the views expressed here do not necessarily reflect the views of the SWAP study (though they are consistent with the overarching themes contained in the report). The intent of these vignettes is to provide some additional points of view and insights that are more specific and, in some cases, more personal.

List of vignettes:

- Implementing Continuous Delivery: The JIDO Approach
- F22: DevOps on a Hardware Platform
- Making It Hard to Help: A Self-Denial of Service Attack for the SWAP Study
- DDS: Fighting the Hiring Process Instead of Our Adversaries
- Kessel Run: The Future of Defense Acquisitions Is #AgileAF
- JMS: Seven Signs Your Software (Program) Is in Trouble

Vignette 1 – Implementing Continuous Delivery: The JIDO Approach

Forrest Shull

One theme that emerges from the work in this study is that DoD certainly does have successes in terms of modern, continuous delivery of software capability; however, in too many cases, these successes are driven by heroic personalities and not supported by the surrounding acquisition ecosystem. In fact, in several cases the demands of the rest of the ecosystem cause friction that, at best, adds unnecessary overhead to the process and slows the delivery of capability. The Joint Improvised-Threat Defeat Organization (JIDO), within the Defense Threat Reduction Agency, is a compelling example.

JIDO describes itself as "the DoD's agile response mechanism, a Quick Reaction Capability (QRC) as a Service providing timely near-term solutions to the improvised threats endangering U.S. military personnel around the world."[11] As such, the speed of delivery is a key success criterion, and JIDO has made important improvements in this domain. Central to accomplishing these successes has been the adoption of a DevSecOps solution along with a continuous ATO process, which exploits the automation provided by DevSecOps to quickly assess security issues.

At least as important as the tooling are the tight connections that JIDO has enabled among the stakeholder groups that have to work together with speed to deliver capability. JIDO has personnel embedded in the user communities associated with different COCOMs, referred to as Capability Data Integrators (CDIs). These personnel are required to be familiar with the domain, familiar with the technology, and forward-leaning in terms of envisioning technical solutions to help warfighter operations. Almost all CDIs have prior military experience and are deployed in the field, moving from one group of users to another, helping to train them on the tools that are available, and at the same time understanding what they still need. CDIs have tight reachback to JIDO and are able to identify important available data that can be leveraged by software functionality and can be developed with speed through the DevSecOps pipeline.

JIDO has also focused on knocking down barriers among contractors and government personnel. JIDO finds value in relying on contractor labor that can flex and adapt as needed to the technical work, with effort spent on making sure that the mix of government personnel and multiple contractor organizations can work together as a truly integrated team. To accomplish this, JIDO has created an environment with a great deal of trust between government and contractors. There are responsibilities that are inherently governmental and tasks that can be delegated to the contractor. Finding the right mix requires experimentation, especially since finding the personnel with the right skillset on the government side is difficult.

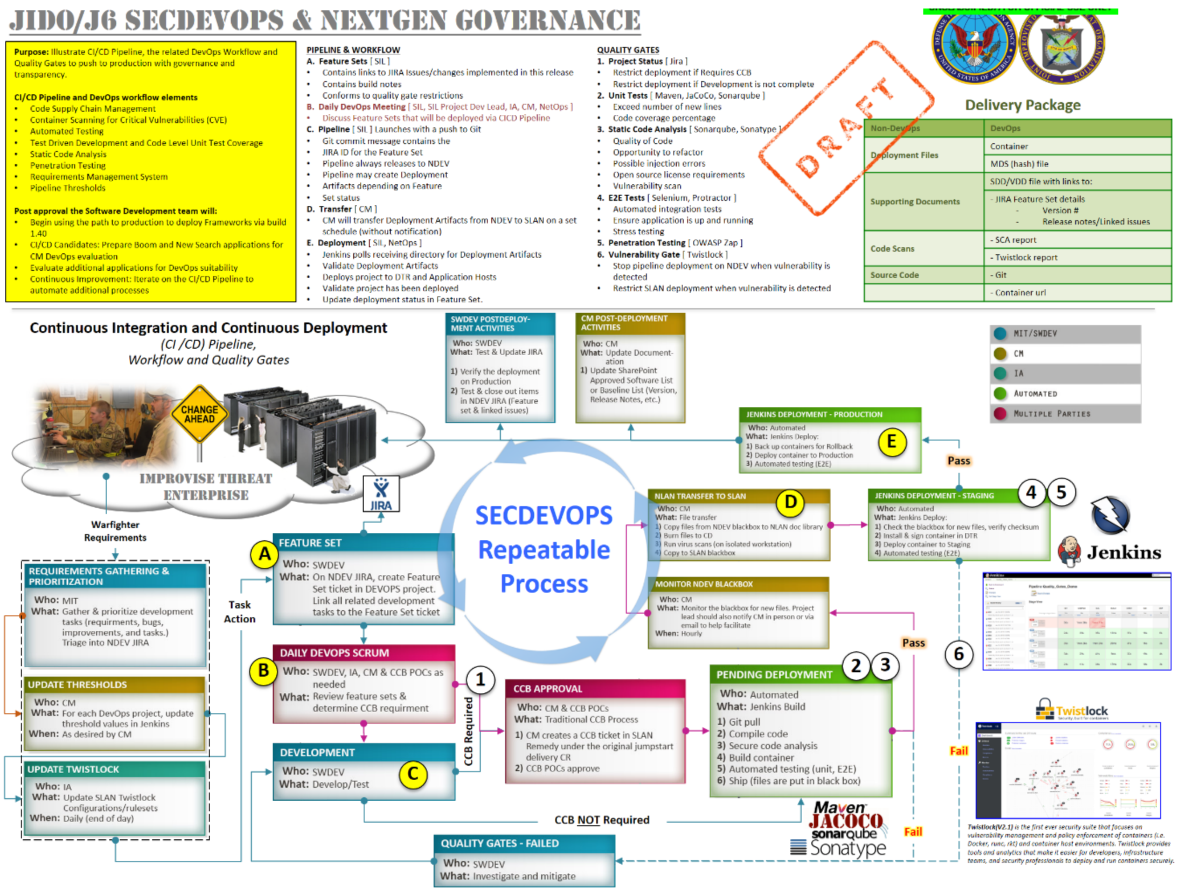
Despite these successes at bringing together stakeholders within the JIDO team, stakeholders in the program management office (PMO) sometimes describe substantial difficulties in working with the rest of the acquisition ecosystem, since on many dimensions the Agile/DevSecOps approach does not work well with business as usual. For example, they describe instances where the Services or the Joint Chiefs push back on solutions that were created to address requirements from the field. Thanks to the CDIs, JIDO can create a technical solution that answers identified

[11] JIDO SecDevOps Concept of Operations, v1.

requirements from warfighters in the field, but that does not mean it will get approval for deployment. There is a mismatch and potential for miscommunication when the organizations that control deployment don't own the requirements themselves.

Also, because JIDO operates in an agile paradigm in which requirements can emerge and get re-prioritized, it is difficult for the organization to justify budget requests upfront in the way that their command chain requires. JIDO addresses this today by creating notional, detailed mappings of functionality to release milestones. Since a basic principle of the approach is that capabilities being developed can be modified or re-prioritized with input from the warfighter, this predictive approach provides little or no value to the JIDO teams themselves. Even though JIDO refuses to map functionality in this way more than 2 years out, given that user needs can change significantly in that time, the program has had to add headcount just to pull these reports together.

JIDO has no problem showing value for the money spent. It is able to show numbers of users and, because it has personnel embedded with user communities, can discuss operational impact. As mentioned above, JIDO's primary performance metric is "response from the theater." Currently, JIDO faces a backlog of tasks representing additional demand for more of its services, as well as a demand for more CDIs. Despite these impactful successes, the surrounding ecosystem unfortunately provides little in the way of support and much that hinders the core mission. It is difficult to see how these practices can be replicated in other environments where they can provide positive impact, until these organizational mismatches can be resolved.



JIDO/J6 SECDEVOPS & NEXTGEN GOVERNANCE

Purpose: Illustrate CI/CD Pipeline, the related DevOps Workflow and Quality Gates to push to production with governance and transparency.

CI/CD Pipeline and DevOps workflow elements

- Code Supply Chain Management
- Container Scanning for Critical Vulnerabilities (CVE)
- Automated Testing
- Test Driven Development and Code Level Unit Test Coverage
- Static Code Analysis
- Penetration Testing
- Requirements Management System
- Pipeline Thresholds

Post approval the Software Development team will:

- Begin using the path to production to deploy Frameworks via build 1.40
- CI/CD Candidates: Prepare Boom and New Search applications for CM DevOps evaluation
- Evaluate additional applications for DevOps suitability
- Continuous Improvement: Iterate on the CI/CD Pipeline to automate additional processes

PIPELINE & WORKFLOW

A. **Feature Sets** [SIL]
- Contains links to JIRA Issues/changes implemented in this release
- Contains build notes
- Conforms to quality gate restrictions

B. **Daily DevOps Meeting** [SIL, SIL Project Dev Lead, IA, CM, NetOps]
- Discuss Feature Sets that will be deployed via CICD Pipeline

C. **Pipeline** [SIL] Launches with a push to Git
- Git commit message contains the
- JIRA ID for the Feature Set
- Pipeline always releases to NDEV
- Pipeline may create Deployment
- Artifacts depending on Feature
- Set status

D. **Transfer** [CM]
- CM will transfer Deployment Artifacts from NDEV to SLAN on a set schedule (without notification)

E. **Deployment** [SIL, NetOps]
- Jenkins polls receiving directory for Deployment Artifacts
- Validate Deployment Artifacts
- Deploys project to DTR and Application Hosts
- Validate project has been deployed
- Update deployment status in Feature Set.

QUALITY GATES

1. **Project Status** [Jira]
- Restrict deployment if Requires CCB
- Restrict deployment if Development is not complete

2. **Unit Tests** [Maven, JaCoCo, Sonarqube]
- Exceed number of new lines
- Code coverage percentage

3. **Static Code Analysis** [Sonarqube, Sonatype]
- Quality of Code
- Opportunity to refactor
- Possible injection errors
- Open source license requirements
- Vulnerability scan

4. **E2E Tests** [Selenium, Protractor]
- Automated integration tests
- Ensure application is up and running
- Stress testing

5. **Penetration Testing** [OWASP Zap]

6. **Vulnerability Gate** [Twistlock]
- Stop pipeline deployment on NDEV when vulnerability is detected
- Restrict SLAN deployment when vulnerability is detected

DRAFT

Delivery Package

Non-DevOps	DevOps
Deployment Files	Container
	MDS (hash) file
Supporting Documents	SDD/VDD file with links to:
	- JIRA Feature Set details - Version # - Release notes/Linked issues
Code Scans	- SCA report
	- Twistlock report
Source Code	- Git
	- Container url

Continuous Integration and Continuous Deployment
(CI /CD) Pipeline,
Workflow and Quality Gates

REQUIREMENTS GATHERING & PRIORITIZATION
Who: MIT
What: Gather & prioritize development tasks (requirments, bugs, improvements, and tasks.) Triage into NDEV JIRA

UPDATE THRESHOLDS
Who: CM
What: For each DevOps project, update threshold values in Jenkins
When: As desired by CM

UPDATE TWISTLOCK
Who: IA
What: Update SLAN Twistlock Configurations/rulesets
When: Daily (end of day)

FEATURE SET (A)
Who: SWDEV
What: On NDEV JIRA, create Feature Set ticket in DEVOPS project. Link all related development tasks to the Feature Set ticket

DAILY DEVOPS SCRUM (B)
Who: SWDEV, IA, CM & CCB POCs as needed
What: Review feature sets & determine CCB requirment

DEVELOPMENT (C)
Who: SWDEV
What: Develop/Test

CCB APPROVAL
Who: CM & CCB POCs
What: Traditional CCB Process
1) CM creates a CCB ticket in SLAN Remedy under the original jumpstart delivery CR
2) CCB POCs approve

PENDING DEPLOYMENT (2) (3)
Who: Automated
What: Jenkins Build
1) Git pull
2) Compile code
3) Secure code analysis
4) Build container
5) Automated testing (unit, E2E)
6) Ship (files are put in black box)

QUALITY GATES - FAILED
Who: SWDEV
What: Investigate and mitigate

MONITOR NDEV BLACKBOX
Who: CM
What: Monitor the blackbox for new files. Project lead should also notify CM in person or via email to help facilitate
When: Hourly

NLAN TRANSFER TO SLAN (D)
Who: CM
What: File transfer
1) Copy files from NDEV blackbox to NLAN doc library
2) Burn files to CD
3) Run virus scans (on isolated workstation)
4) Copy to SLAN blackbox

JENKINS DEPLOYMENT - STAGING (4) (5)
Who: Automated
What: Jenkins Deploy:
1) Check the blackbox for new files, verify checksum
2) Install & sign container in DTR
3) Deploy container to Staging
4) Automated testing (E2E)

JENKINS DEPLOYMENT - PRODUCTION (E)
Who: Automated
What: Jenkins Deploy:
1) Back up containers for Rollback
2) Deploy container to Production
3) Automated testing (E2E)

SWDEV POSTDEPLOY-MENT ACTIVITIES
Who: SWDEV
What: Test & Update JIRA
1) Verify the deployment on Production
2) Test & close out items in NDEV JIRA (Feature set & linked issues)

CM POST-DEPLOYMENT ACTIVITIES
Who: CM
What: Update Document-ation
1) Update SharePoint Approved Software List or Baseline List (Version, Release Notes, etc.)

Twistlock(V2.1) is the first ever security suite that focuses on vulnerability management and policy enforcement of containers (i.e. Docker, runc, rkt) and container host environments. Twistlock provides tools and analytics that make it easier for developers, infrastructure teams, and security professionals to deploy and run containers securely.

Slide image received from former DTRA-JIDO chief technology officer.

Vignette 2 – F22: DevOps on a Hardware Platform

Craig Ulsh and Maj Zachary McCarty

The F-22A Raptor program recognized a need for greater speed and agility and took action. In mid-2017, the F-22 Program Office realized the F-22A Raptor modernization efforts were not delivering at a speed that would keep pace with emerging threats. Program leadership secured the expertise of the Air Force Digital Service (AFDS). A joint team assessed the program and captured a series of observations and recommendations. The overarching assessment was:

> The Air Force must move faster, accept a greater amount of risk, and commit to radical change with how the F-22A modernization effort is managed and technology is implemented. Competitors are moving faster, and blaming poor vendor performance will not help the F-22A Raptor remain the dominant air superiority platform.

The F-22A Program Office realized that change was needed. The F-22 acquisition process, steeped in the traditional DoDI 5000 model, was slow and cumbersome, with initial retrofits taking at least 6 years to deliver. The program recognized the following symptoms:

- Requirements were static and rigidly defined.
- Capability was delivered in large, monolithic releases.
- Change was avoided and treated as a deviation from well-guarded baselines.
- The development team placed too much focus on intensive documentation.
- Separate programs with separate contracts drove inefficiencies and conflicting interests.
- Insufficient automation for incremental testing resulted in marathon test events. More specifically, the team identified a number of issues that are common among weapon systems:

Development practices. Development processes were matched to the traditional acquisition process. Large feature sets, multiple baselines, highly manual developer testing tools, and limited focus on continuous software infrastructure upgrades contributed to the slow capability delivery cycle. The team made several specific recommendations under the overarching recommendation for the software development teams to adopt modern software practices.

Planning. Several inefficiencies were identified in the planning process including lack of metrics for estimation of effort, inability to prioritize, and inefficient use of developer time. Again, the team proposed that the program adopt modern agile software processes.

Organization. Organizational gaps included poor collaboration across teams, lack of incentives for engineering talent, and competing priorities across multiple vendors.

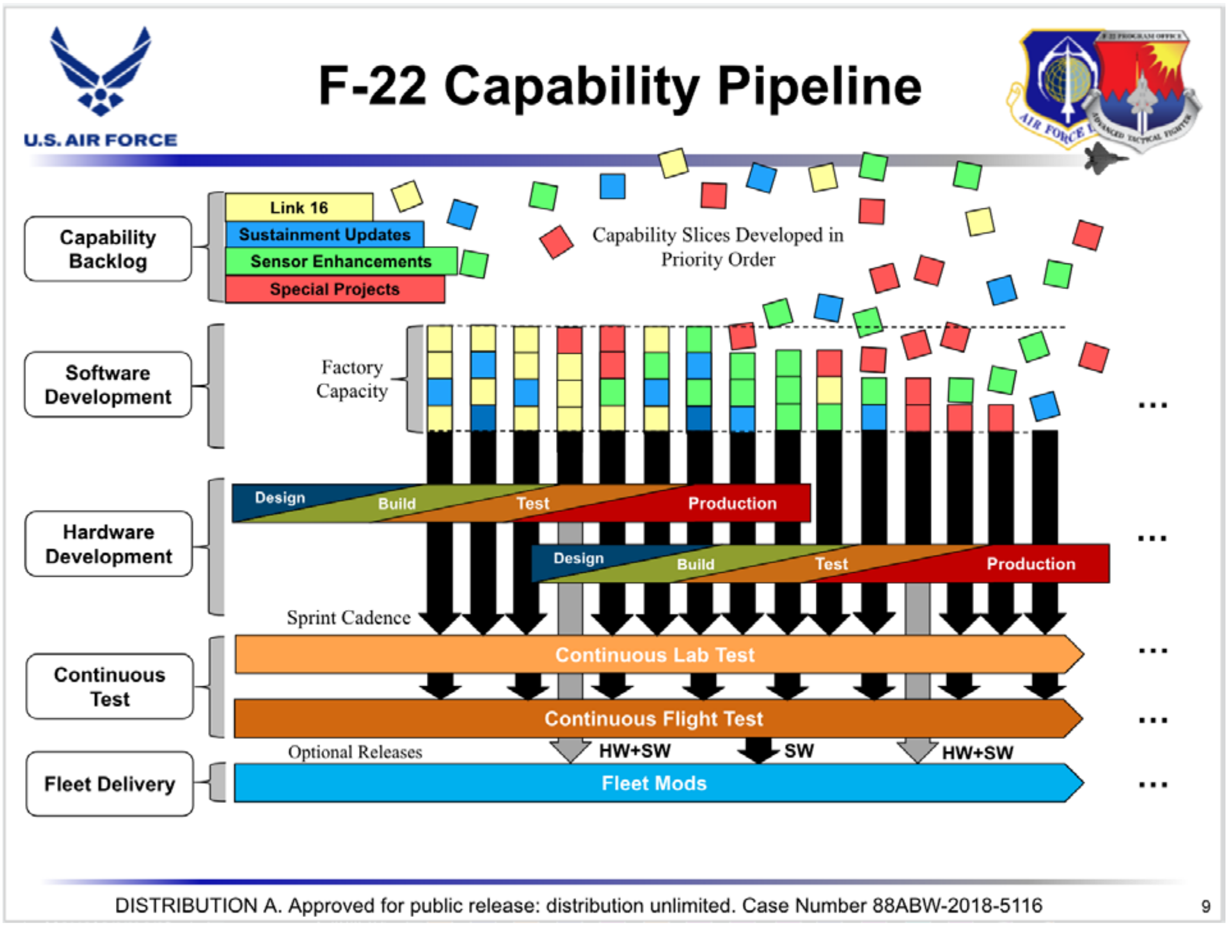
Contracts. The single most significant observation is the failure to prioritize.

In November 2017, the F-22 Program Office took several steps to accelerate the F-22A modernization efforts. In response to outdated development practices, the program office restructured TACLink 16 and TACMAN programs into a single agile development stream. To properly match the contractor effort with a new development approach, a “level of effort” for prime development labor was adopted. To address some of the planning concerns, steps were taken to adjust program alignments and authorities.

The F-22A Raptor program has made positive steps in adopting a more modern approach to both hardware and software acquisition. Perhaps the best example is a new contract structure that allows for quick reaction to emerging requirements and changing user priorities while incentivizing a long-time incumbent contractor for continuous improvement. The Program Office has learned lessons during the transition to more agile approaches, including:

- Culture change has been the biggest hurdle.
- The program must recognize and accept that things will go wrong.
- Security controls limit flexibility and communication.

The program is on the right track with a sound plan to accelerate delivery. But the program office also noted, in the immortal words of Mike Tyson, “Everyone has a plan until they get punched in the face.”



Slide image received for briefing from F22A Raptor Program Office.

Vignette 3 – Making It Hard to Help: A Self-Denial of Service Attack for the SWAP Study

Richard Murray

DoD makes use of advisory committees consisting of a mixture of government, industry, and academic experts, all trying to help. However, the Department can make it extremely difficult for these groups to function, an example of what we refer to on the Defense Innovation Board (DIB) as a "self-denial of service attack."[12] The DIB SWAP study is itself a case in point.

<rant>

The DIB Software Acquisition and Practices (SWAP) study clock started ticking when the 2018 NDAA was signed on 12 December 2017. We had our first SWAP discussion at the Pentagon on 16 January 2018, before we had officially been requested by the Under Secretary for Defense (Acquisition and Sustainment) to start, but knowing this was coming (and using the DIB Science & Technology [S&T] committee to ramp up quickly). We identified potential subcommittee members by 12 February, and we were officially charged to carry out the study on 5 April 2018. The one-year Congressionally-mandated end date was thus set as 5 April 2019. The DIB S&T subcommittee submitted the list of suggested subcommittee members. Then we started waiting…

On 24 May, after a DIB meeting, one of the SWAP co-chairs found out that there had been no movement on these positions. He sent a note to the DIB's Executive Director, expressing disappointment and reiterating the importance of getting these people on board early in the study. The Executive Director tried to use this note to push things along. More waiting…

The first activity in which any new member of the SWAP subgroup participated took place on 1 November 2018— a full 30 weeks after our 52-week countdown started and 9 months after we had identified the people whom we wanted to enlist in to help in our study. Even this took repeated interventions by the DIB staff and, in the end, only two of the four people who we hoped could help were able to participate in the study. The timing was such that we had already visited five of the six programs with which we met, written seven of the eight concept papers that we generated, and held three of the four public meetings that provided input for our report.

Why did things take so long? These people were ready to help, had served in government advisory roles in the past, and provided incredibly valuable input in the end (but only in the end). Maybe we need some sort of "FACA Pre ✓" that allows DoD to make use of people who are willing to help and all we need to do is ask.

Another example: the SWAP study decided to use Google's G Suite as the means for writing our report. It had some nice features for collaboration and several of us were familiar with using it. Setting up a G Suite site is fast and easy, and a member of the study had previously created a site in a matter of minutes and had a fully operational, two-factor authenticated set of accounts

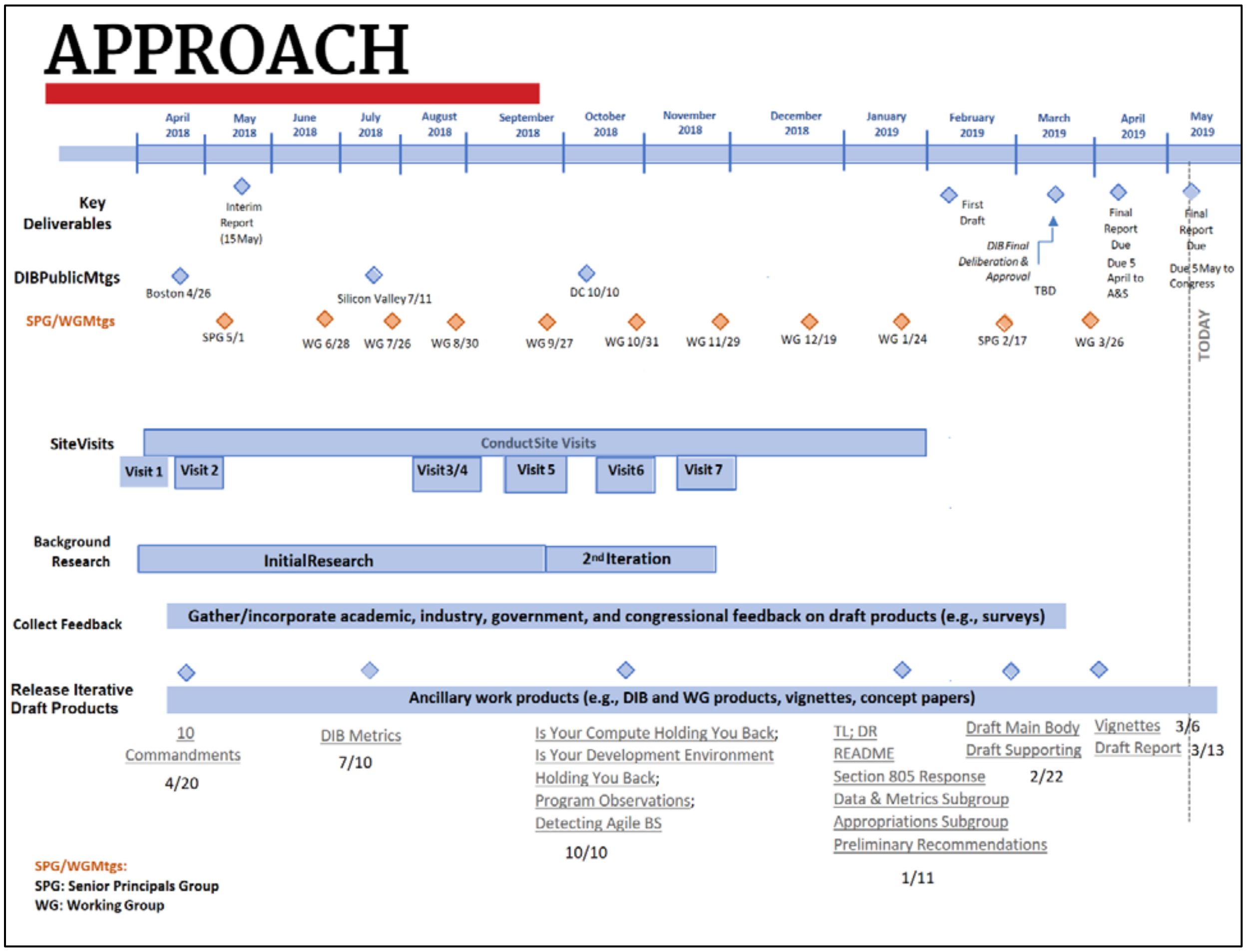
[12] The DIB first heard this term from one of the military instructors at the Air Force Academy and we now use it all the time.

up and running in less than a week. It turns out that the Department has the authority to create official G Suite sites and so we just needed to get permission to use it.

Our request went in ~10 April 2018. The site was created on 8 August 2018, 17 weeks after our request. As near as we can tell, the only thing that happened during the 4 months that it took to get the site working was that people said “no” and then other people had to spend time figuring out why they said no and either convincing them that this really was useful and a good solution for the study’s needs and/or going above their heads.

A major theme from the beginning of the SWAP study, and more generally in the DIB’s overall work, has been that DoD technology must move at the speed of (mission) need, faster than our adversaries and, certainly, not that much slower than what has proven possible and effective in the private sector. If the Department wants to take advantage of people who can help it be more effective in development and delivery of technology for improving national security, it should figure out how to *quickly* put together groups of people from inside and outside government, provide them with modern collaboration environments, and let them spend their time providing service to the Department instead of struggling with the bureaucracy.

</rant>



SWAP study schedule (used for briefings).

Vignette 4 – DDS: Fighting the Hiring Process Instead of Our Adversaries

Sean Brady, Kevin Carter, Justin Ellsworth

In novelist James Patterson and former President Bill Clinton's political thriller, *The President Is Missing*, a terrorist group threatens to unleash cyber-warfare on the Western World, bringing about the "Dark Ages." The President (in the story) must sneak away from the White House incognito, engage in shootouts, survive an ambush on Memorial Bridge, and assemble the best computer scientists from our government and military to take out the impending computer virus before it strikes.

At this point, the novel introduces a top "white hat hacker" who joins the President's team. She impresses the FBI with her hacking abilities and the Bureau hires her on the spot. In a sensational thriller that constantly demands suspended disbelief, this was by far the most unbelievable.

There's no way government hiring works that effectively or efficiently.

We know because we tried.

The Defense Digital Service (DDS) is an organization within the Pentagon tasked with driving a giant leap forward in the way DoD builds and deploys technology and digital services. One of DDS's most visible programs is Hack the Pentagon, the first bug bounty program in the history of the federal government. Bug bounties (also known as crowd-sourced hacking challenges) allow private citizens to harness their diverse range of talents to contribute and strengthen our nation's security posture in exchange for a monetary reward for finding security issues. Bug bounties are an integral part of private-sector security strategies at companies including Microsoft, Google, Twitter, and Facebook.

The winner of one of these Hack the Pentagon challenges was a 17-year-old high school student, who beat out 600 other invited hackers by reporting 30 unique vulnerabilities to the Department. After the challenge, he expressed interest in interning so he could help contribute to our nation's security outside of the challenges.

DDS staff spent the next 8 months and approximately 200 man hours trying to navigate the hiring process to bring the hacker onboard. DDS engaged with the Washington Headquarters Service, the Air Force internship program, and U.S. Army Cyber HR organizations to identify applicable hiring authorities and, more important, the HR specialists who could help drive the hiring actions for a non-traditional, but obviously qualified, candidate.

Unfortunately, what we found was a system ill-equipped to evaluate technical expertise (especially when demonstrated through experience or skill rather than certifications or education) and resistant to leveraging the full flexibilities and authorities provided.

Twice the hacker's resume was rejected as insufficient to qualify him at the necessary grade level for using direct hire authority. Ultimately, the candidate lengthened his resume to a total of five pages, which a classifier reviewed and determined would qualify him for the General Schedule (GS)-4 level, which equates to less than $16 per hour. (For what it's worth, the GS-5 only requires "experience that provided a knowledge of data processing ... gained in work such as a computer operator or assistant, [or] computer sales representative…" according to the OPM GS-2210: Information Technology Management Series General Schedule Qualification Standards). We like to point out that he would have qualified if he had worked a year at Best Buy.

Oh, and did we mention he landed on *TIME*'s List of the 25 Most Influential Teenagers of 2018? He is currently studying computer science at Stanford University.

We recognize that it is unreasonable to expect a classification specialist to understand and translate the experience listed in a resume into the education, demonstrated knowledge, and specialized experience requirements that must be met for each grade level in each job series.

The classification specialist may not have known how this particular candidate's listed experience developing *"mobile applications in IonicJS, mobile applications using Angular, and APIs using Node.js, MongoDB, npm, Express gulp, and Babel,"* met or did not meet the classification requirements of *"experience that demonstrated accomplishment of computer-project assignments that required a wide range of knowledge of computer requirements and techniques pertinent to the position to be filled.*"

This is why DDS provided a supporting memo to the classifier that identified where the candidate's resume and classification guide matched. However, the HR office refused to accept the supporting document despite OPM guidance that *"It is entirely appropriate (and encouraged!) to use Subject Matter Experts (SMEs) outside of HR to rate and rank applicants and determine the most highly qualified candidates for a position."*

Thankfully, our story, like *The President Is Missing*, has a happy ending. When it became clear that we would lose the hacker to a competing offer from the private sector, leaders at some of the highest levels of the Pentagon intervened and ordered their HR office to make the hire. With sufficient visibility and the right people assigned, the hacker's original (one-page) resume was reviewed and used to hire him at a reasonable but still below-market rate. We were ultimately able to hire him, but the process required escalation and is not scalable for more than a small number of hires.

The hacker, now 18, joined DDS as an employee during the summer of 2018 and during that time identified numerous vulnerabilities that threatened the security of information and potentially the safety of our nation.

His story was not isolated to one HR specialist or one service. As a Department, we made it as hard as possible for him to join (all while the private sector offered higher salaries and housing stipends). Hiring him did not require a new law or regulation; it required an understanding of his technical abilities, trust in those who evaluated him, and leadership that prioritizes people over process.

Vignette 5 – Kessel Run: The Future of Defense Acquisitions Is #AgileAF

Dan Ward

I've seen the future, and it's #agileAF.

Kessel Run's lab director welcomes new engineers. [U.S. Air Force photo by Todd Maki]

That's the hashtag used by an Air Force software company known as Kessel Run—the "AF" stands for Air Force, by the way. And I did say "software company," which is how members of this military unit describe their organization. Kessel Run does not look like any other program office the Air Force has ever seen. That is its great strength. That is its great peril. And that is why it is the future.

What's so great about Kessel Run? For starters, it delivers. As one example from many, in less than 130 days Kessel Run fielded an accredited Secret Internet Protocol Router (SIPR) cloud-native DevOps platform at Al Udeid Air Base, then replicated the instance at Shaw Air Force Base and fielded another DevOps platform at Osan Air Base in Japan. Don't worry if that last sentence sounded like technobabble—the point is they put stuff into the field quickly. In contrast, the previous program charged with addressing this need (which went by the catchy name "AOC 10.2") spent $430 million over 10 years before being terminated "without delivering any meaningful capability," to quote Senator John McCain. But while Kessel Run's ability to field operational software is noteworthy, its organizational achievement and the culture the team has built just might be the real breakthrough.

It turns out disruptive new technologies do not merely require cutting-edge tech. They also require new *organizational architectures*, to use Professor Rebecca Henderson's term, and very specific cultural features.

Easier said than done, of course. Building and sustaining these innovative structures inside a large legacy organization like the U.S. military requires replacing existing standards and norms. That's even harder than it sounds and is why so many large companies fail to make the switch.

Despite the difficulty, the Kessel Run team seems to have cracked the code and built a unique organization that operates at warp speed. The most visible difference between Kessel Run and business-as-usual military program offices is their location. Rather than spending all their time on the military base they are *technically* assigned to, Kessel Run personnel operate from a brightly lit We Work office in downtown Cambridge, MA. The conference rooms have Star Wars–themed names instead of Mil-Standard room numbers. The walls are covered in multi-colored sticky notes. The view of Boston is spectacular. You get the picture.

Only slightly less visible is Kessel Run's approach to contracting. Instead of handing the work over to a major defense contractor, team members built a collaborative partnership with a small-ish software company named Pivotal. Together they use DevOps methods like pair programming, where Air Force coders work side-by-side with Pivotal coders to produce software that runs on classified military systems and supports real-world military operations.

Where people sit and how they collaborate are just the tip of the iceberg. The Kessel Run culture is the product of hundreds of thoughtful design decisions that continually reinforce principles of learning, collaboration, critical thinking, and agility. The details of these decisions are beyond the scope of this short vignette, but the fact that Kessel Run continues to do the hard work of deliberately crafting and maintaining its culture is absolutely foundational to its success story.

That story is happening right now, so saying "the future is #agileAF" is actually an observation about the present. Kessel Run's approach is what right looks like *today*. Kessel Run is the new standard of military acquisition excellence, and already the other Services are starting to follow suit. Just last month the U.S. Naval Institute's blog had a post titled The Navy's Kessel Run. When your program office's name gets used in a headline like that, it's a sure sign you're doing something right.

Some skeptical commentators have expressed concern about the risks inherent in a high-speed operation like Kessel Run. In response, let's hear from the four-star commander of U.S. Strategic Command, General John Hyten. He's responsible for the nation's nuclear arsenal and is precisely the type of serious, thoughtful, risk-averse leader we want in charge of nuclear weapons. If anyone has a definitive professional opinion on Kessel Run's risk profile, it's General Hyten.

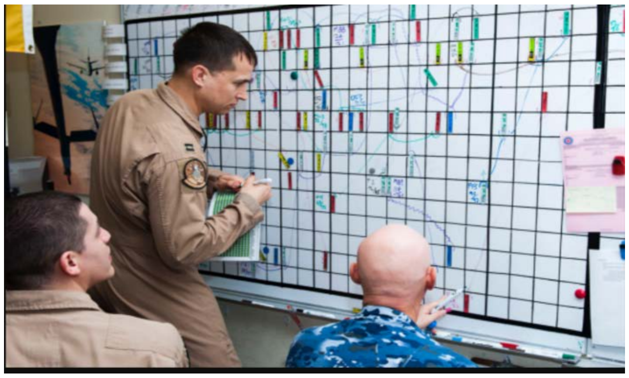
On several occasions General Hyten has stated that what keeps him up at night is the thought that the U.S. military's technology community has "lost the ability to go fast." This inability to move quickly increases the likelihood of operational shortfalls and degrades our nation's overall defense posture. In General Hyten's assessment, going too slow is far riskier than going too fast. He sounds quite comfortable with Kessel Run's pace.

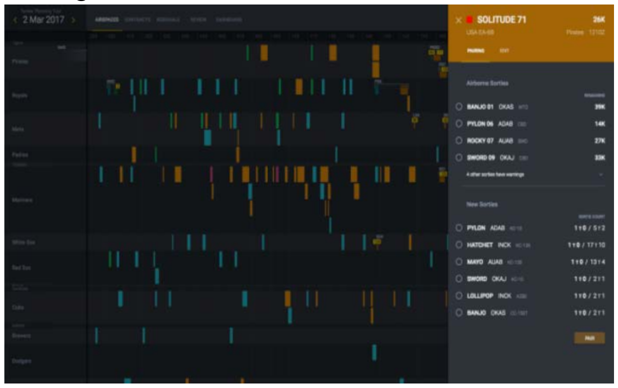
In a similar vein, Secretary of the Air Force Heather Wilson submitted a report to Congress in October 2018 that described Kessel Run's achievements to date. She wrote "The use of Agile DevOps methodologies … is proving successful and we are able to rapidly deliver cloud native applications that increase operational utility. … *We believe we have demonstrated the ability to continuously deliver software that adds value to the warfighter.* " (emphasis added.)

So the question is not whether the Kessel Run team delivers good results or addresses the needs of the operational community. It clearly does. Instead, the question is how long it will take the Department of Defense to adopt this organizational innovation on a larger scale. How long will DoD wait before making Kessel Run-style organizations and culture the default rather than the exception?

Replicating the Kessel Run culture requires more than giving all your conference rooms Star Wars-themed names and putting military personnel into civilian clothes. In fact, the best way to replicate the Kessel Run culture is to *not* replicate it exactly. The wisest imitators will use Kessel Run's example for illumination, not imitation. They will learn from Kessel Run's practices, not simply cut and paste them onto existing organizational structures. The wisest imitators will commit to having the difficult, ongoing conversations about values, attitudes, and beliefs that lead to genuine culture shifts. They will do the hard work of establishing and maintaining a healthy culture that unleashes people's talent and enables them to do their best work.

Kessel Run is not perfect, of course. It has collected a number of critics and skeptics alongside its fans and supporters. Interestingly, no critics see the project's shortcomings more clearly and pointedly than the Kessel Run members themselves. The team members are very aware they are still learning, still experimenting, still making mistakes and identifying opportunities for improvement. They are the first to tell you that Kessel Run has problems and struggles. They are quick to agree with some of their critics about ways the program can and should improve. That is the thing I admire most about this team. That just might be the most important practice for the rest of us to follow. And that is precisely why the future is #agileAF.


Whiteboard on which tanker refueling operations were planned. [Photo by U.S. Air Force]


The tanker refueling planning app that replaced the AOC's whiteboard. [Photo by U.S. Air Force]

Air Force Kessel Run Headquarters in Boston, MA. [U.S. Air Force photo by J.M. Eddins Jr.]

Vignette 6 – JMS: Seven Signs That Your Software (Program) Is in Trouble

Richard Murray

The DIB SWAP study visited the JMS (JSpOC [Joint Space Operations Center] Mission System) program in August 2018. The JMS team was open and cooperative, and the people working on the project were highly capable and well-intentioned. At the same time, our assessment of the program was that it was doomed to failure. Because the JMS program was restructured after our visit, we felt it was OK to spell out the problems as examples of what can go wrong.

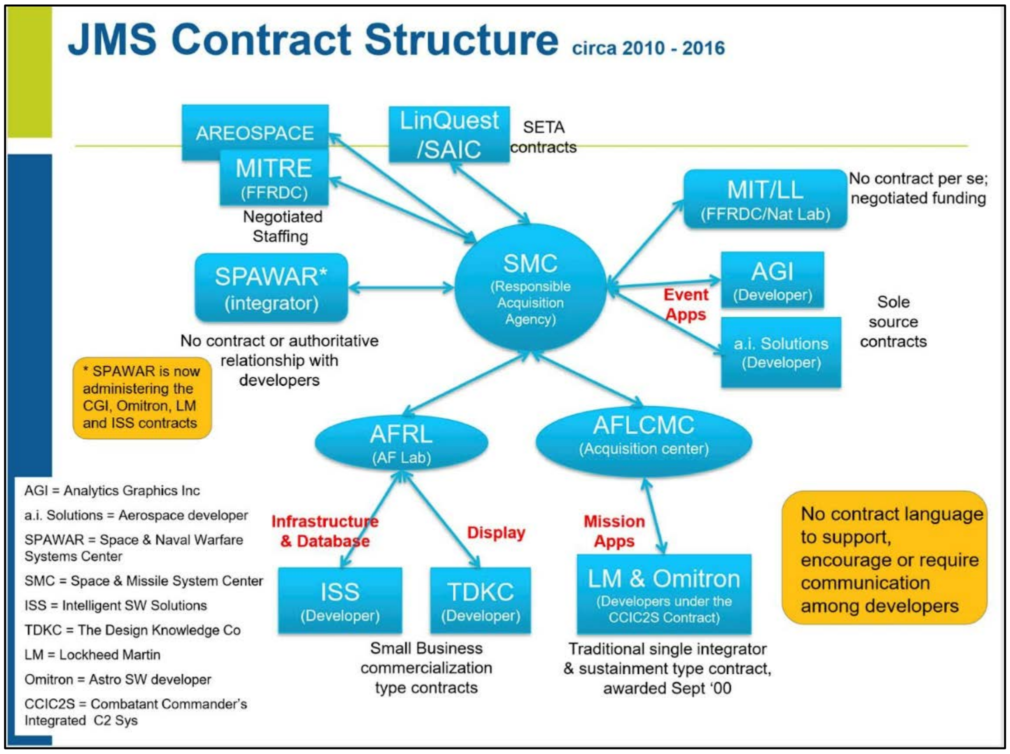
While there were many issues that led to the failure of the JMS program, the following seven are ones that are not a function of that program *per se*, but rather of the process that created it. We thus call these out as general things to look for as indications that your software (program) may be in trouble.

1. The problem is being made harder than it needs to be. JMS increment 2 had a budget of just under $1B. The basic function of the JMS system was to track objects in space. While there are engineering challenges to doing this with the proper precision, the basic problem is *not that hard*. Our sense was that the project could be converted to an "app" within AOC Pathfinder, or something equivalent. Assign 20–30 [50? 100?] programmers (+ 20% program management, administration) to work on it for 3 years at $10–20M/year, with first capability due in 6 months and increments every 2 weeks (based on user feedback). Interface to existing data sources (via software interfaces), run in the cloud, and use a scalable architecture that can get to 1M objects in the next year or two. Make sure that the app architecture can accept a commercial product if one is available that meets the needs of the user (there were some indications this might have already been happening). Target budget: $10–20M/year for first 5 years, $5–15M/year in perpetuity after that.

2. The requirements are outdated. Many of the requirements for JMS increment 2 appeared to trace back to its original inception circa 2000 and/or its restart in 2010. Any software program in which a set of software requirements was established more than 5 years ago should be shut down and restarted with a description of the desired end state (list of features with specifications) and a prioritization of features that should be targeted for simplest usable functionality.

3. The program organizational structure is designed to slow things down. Any software program with more than one layer of indirection between the prime contractor/integrator and the companies doing the engineering work should be shut down and restarted with a set of level-of-effort–style contracts that go directly from the system integrator to the companies delivering code. The system integrator should own the architecture, including the design specifications for the components that plug into that architecture.

4. The program contract structure is designed to slow things down even more. The program had at least a dozen contracts with all sorts of small companies and National Labs. It was apparently treated as a COTS integration problem with lots of pieces, but it was implemented in a way that seemed designed to ensure that nobody could make any progress.


JMS Contract Structure circa 2010 - 2016
AREOSPACE
LinQuest /SAIC
SETA contracts
MITRE (FFRDC)
Negotiated Staffing
MIT/LL (FFRDC/Nat Lab)
No contract per se; negotiated funding
SPAWAR* (integrator)
SMC (Responsible Acquisition Agency)
Event Apps
AGI (Developer)
a.i. Solutions (Developer)
Sole source contracts
No contract or authoritative relationship with developers
* SPAWAR is now administering the CGI, Omitron, LM and ISS contracts
AFRL (AF Lab)
AFLCMC (Acquisition center)
AGI = Analytics Graphics Inc
a.i. Solutions = Aerospace developer
SPAWAR = Space & Naval Warfare Systems Center
SMC = Space & Missile System Center
ISS = Intelligent SW Solutions
TDKC = The Design Knowledge Co
LM = Lockheed Martin
Omitron = Astro SW developer
CCIC2S = Combatant Commander's Integrated C2 Sys
Infrastructure & Database
Display
Mission Apps
ISS (Developer)
TDKC (Developer)
LM & Omitron (Developers under the CCIC2S Contract)
No contract language to support, encourage or require communication among developers
Small Business commercialization type contracts
Traditional single integrator & sustainment type contract, awarded Sept '00

JMS contract structure. [Photo courtesy of former JMS program office]

5. The program is implementing "waterfall with sprints" (otherwise known as Agile BS). The program was implementing "sprints" of ~6–9 months (Agile BS detector alert!). Sprints had hundreds of tasks spread across six development teams. Just coordinating was taking weeks. For a while the program had used 4-week sprints, but infrastructure was not available to support that cadence. Test happened after delivery of software, with very little automation.

6. The program management office is too big and does not know enough about software. We were told there were 200–260 FTEs in the program office. The overall program management should be limited to 10–20% of the size of the program so that resources are focused on the development team (including system architects, user interface designers, programmers, etc.), where the main work gets done. The program office must have expertise in software programs so that it is able to utilize contract and oversight structures that are designed for software (not hardware).

7. OT&E is done as a tailgate process. As an ACAT1 program, JMS was mandated to conduct operational test, a process that nominally required the program to freeze its baseline, do the tests, and then wait 120 days for report. The Operational User Evaluation conducted in early 2018 was terminated early by the Air Force due to poor performance of the system. The OT&E process being used by the program added information to support the termination decision, but it is important to note that had the program not been terminated the tailgate nature of the evaluation was one that would have added further delays.

The JMS program has since undergone major changes to address the issues above, so the criticisms here should be taken as an example of some of the signs that a program is in trouble.

Software Is Never Done:
Refactoring the Acquisition Code for Competitive Advantage

Defense Innovation Board, 3 May April 2019

J. Michael McQuade and Richard M. Murray (co-chairs)
Gilman Louie, Milo Medin, Jennifer Pahlka, Trae' Stephens

Supporting Information

This document contains the supporting information for the Defense Innovation Board's (DIB's) Software Acquisition and Practices (SWAP) study.

Contents

Appendix A: Draft Implementation Plan

The following pages contain summaries for each recommendation that give more detail on the rationale, supporting information, similar recommendations, specific action items, and notes on implementation. The beginning of each recommendation summary includes the recommendation statement, proposed owner, background information, description of the desired state, proposed role for Congress, and a short list of actions describing how the recommendation might be implemented. The remainder of the summary contains a list of recommendations from the DIB Guides (contained in Appendix E of the supporting information), a list of recommendations from the working group reports (Appendix F of the supporting information), and some related recommendations from previous reports.

The recommendations listed here are relatively decoupled, but there are some dependencies between them, as shown to the right. In figure A.1, an arrow leading from one recommendation toward a second recommendation means that the first implementation depends at least somewhat on the implementation of the second. Hence by choosing one recommendation and following the arrows, the list of all recommendations that should also be implemented can be obtained.

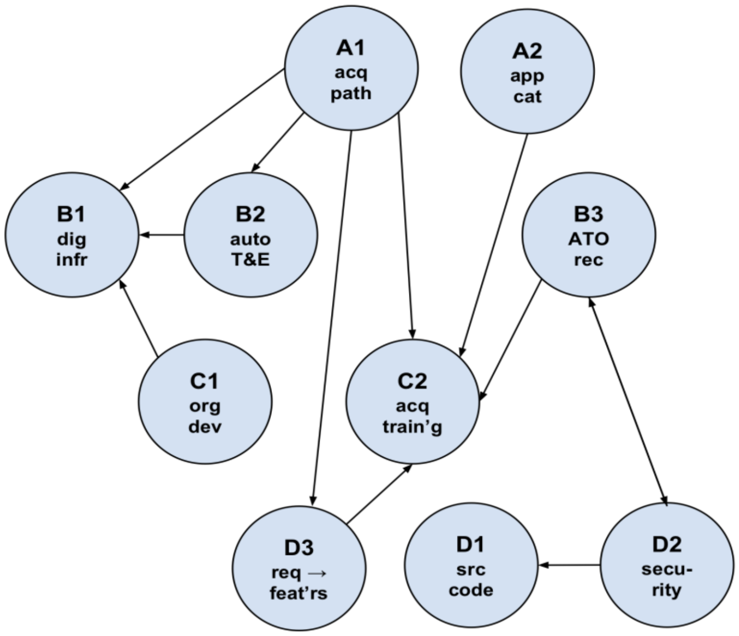


Figure A.1. Interdependency of recommendations.

The recommendations of the report are broken up into four primary lines of effort:

A. Refactor statutes, regulations, and processes for software

B. Create and maintain cross-program/cross-service digital infrastructure

C. Create new paths for digital talent (especially internal talent)

D. Change the practice of how software is procured and developed

For each of the lines of effort, we give a set of two or three primary recommendations (bold) and two to four additional recommendations (see Chapter 5 for insights).

Primary Recommendation A1
New Acquisition Pathway

Line of Effort	Refactor statutes, regulations, and processes for software.
Recommendation	**Establish one or more new acquisition pathways for software that prioritize continuous integration and delivery of working software in a secure manner, with continuous oversight from automated analytics.**
Stakeholders	A&S, HASC/SASC, USD(C), CAPE, DOT&E, R&E/DT, SAE, Service FM & PA&E, Joint Staff
Background	Current law, regulation, policy, and internal DoD processes make DevSecOps software development extremely difficult, requiring substantial and consistent senior leadership involvement. Consequently, DoD is challenged in its ability to scale DevSecOps software development practices to meet mission needs.
Desired State	Tailored, software-specific pathways that provide guidance to acquisition professionals for navigating the acquisition and requirements life cycle to rapidly deliver capabilities. Each pathway streamlines the processes, reviews, and documents based on the type of IT/SW capability. Programs choosing these pathways have the ability to rapidly field and iterate new functionality in a secure manner, with continuous oversight based on automated reporting and analytics, and utilizing IA-accredited commercial development tools. Rapid acquisition authority should be available for software already in use and accredited, especially when purchased as a capability delivery (as a service). Over time, this becomes the default choice for software and software-intensive programs/program elements.
Role of Congress	This acquisition pathway should become the primary pathway that DoD chooses to use for software and software-intensive programs and should provide Congress with the insight required to oversee software projects that move at a much faster pace than traditional HW programs, with traditional metrics and milestones replaced by more software-compatible measures of progress.

Draft Implementation Plan		**Lead Stakeholder**	**Target Date**
A1.1	(optional) Submit legislative proposal using Sec 805 to propose new acquisition pathways for two or more classes of software (e.g., application, embedded), optimized for DevSecOps.	USD(A&S), in coordination with USD(C) and CAPE	Q3 FY19
A1.2	Create new acquisition pathway(s) for two or more classes of software, optimized for DevSecOps (based on A2c.1 or Appendix B.1).	HASC, SASC	FY20 NDAA
A1.3	Develop and issue a Directive-Type Memorandum (DTM) for the new software acquisition pathway.	USD(A&S)	Q1 FY20
A1.4	Issue Service-level guidance for new acquisition pathway.	SAEs	Q2 FY20

A1.5	Select 5 initial programs using modern software development (DevSecOps) to convert to or use new software acquisition pathway.	USD(A&S), with SAEs	Q2 FY20
A1.6	Develop and implement training at Defense Acquisition University on new software acquisition pathway for all acquisition communities (FM, Costing, PM, IT, SE, etc.).	USD(A&S)	Q3 FY20
A1.7	Convert DTM to DoD Instruction (perhaps 5000.SW), incorporating lessons learned during initial program implementation.	USD(A&S)	Q4 FY20

SWAP working group inputs (reflected in Appendix F) related to this recommendation

Acq	Define software as a critical national security capability under Section 805 of FY16 NDAA "Use of Alternative Acquisition Paths to Acquire Critical National Security Capabilities."
Acq	Create an acquisition policy framework that recognizes that software is ubiquitous and will be part of all acquisition policy models.
Acq	Create a clear, efficient acquisition path for acquiring non-embedded software capability. Deconflict supplemental policies.
Acq	Develop an Enterprise-level Strategic Technology Plan that reinforces the concept of software as a national security capability and recognizes how disruptive technologies will be introduced into the environment on an ongoing basis.
Acq	Additionally, take all actions associated with Rec A2a to refactor and simplify those parts of Title 10, DoD 5000 and other regulations and processes that are still in force for software-intensive programs.

Related recommendations from previous studies

DSB87	Rec 13: The Undersecretary of Defense (Acquisition) should adopt a four-category classification as the basis of acquisition policy [standard (COTS), extended (extensions of current systems, both DoD and commercial), embedded, and advanced (advanced and exploratory systems)].
DSB87	Rec 14: USD(A) should develop acquisition policy, procedures, and guidance for each category.
DSB09	The USD(AT&L) should lead an effort, in conjunction with the Vice Chairman, Joint Chiefs of Staff, to develop new, streamlined, and agile capabilities (requirements) development and acquisition processes and associated policies for information technology programs.

Primary Recommendation A2
New Appropriation Category

Line of Effort	Refactor statutes, regulations, and processes for software.
Recommendation	**Create a new appropriation category for software capability delivery that allows (relevant types of) software to be funded as a single budget item, with no separation between RDT&E, production, and sustainment.**
Stakeholders	A&S, HAC-D/SAC-D, HASC/SASC, USD(C), CAPE, SAE, Service FM & PA&E, FASAB, OMB
Background	Current law, regulation, and policy treat software acquisition as a series of discrete, sequential steps; accounting guidance treats software as a depreciating asset. These processes are at odds with software being continuously updated to add new functionality, and they create significant delays in fielding user-needed capability.
Desired State	Appropriations for software and software-intensive programs use a Major Force Program (MFP) category that provides a single budget to support full life cycle costs of software, including development, procurement, assurance, deployment, and continuous improvement. Programs are better able to prioritize how effort is spent on new capabilities versus fixing bugs/vulnerabilities, improving existing capabilities, etc. Such prioritization can be made based on warfighter/user needs, changing mission profiles, and other external drivers, not constrained by available sources of funding.
Role of Congress	This should become the primary pathway that Congress uses to fund software and software-intensive programs and should provide Congress with the insight required to oversee software projects that move at a much faster pace than traditional HW programs, with traditional metrics and milestones replaced by more software-compatible measures of progress.

Draft Implementation Plan		**Lead Stakeholder**	**Target Date**
A2.1	(optional) Submit legislative proposal using Sec 805 to create a new appropriations category for software and software-intensive programs.	USD(A&S), with USD(C) and CAPE	Q3 FY19 for FY20 NDAA
A2.2	Create new appropriation category for software-intensive programs, with appropriate reporting and oversight for software (based on Action A2.1 or Appendix B.1).	HAC-D, SAC-D, with OSD, HASC, SASC	FY20 NDAA, FY20 budget
A2.3	Select initial programs using DevSecOps to convert to or use new SW Appropriation in FY20.	USD(A&S), with Service Acquisition Executives	Q4 FY19
A2.4	Define budget exhibits for new SW appropriation (replacement for P- and R-Forms; see Appendix C).	USD(A&S), with USD(C), CAPE, HAC-D, SAC-D	Q4 FY19

A2.5	Change audit treatment of software with these goals: (1) separate category for software instead of being characterized as property, plant, and equipment; (2) default setting that software is an expense, not an investment; and (3) "sustainment" is an integrated part of the software life cycle.	FASAB, with USD(A&S) and USD(C)	End FY20
A2.6	Make necessary modifications in supporting PPB&E systems to allow use and tracking of new software appropriation.	USD(C) and CAPE	Q1 FY21
A2.7	Ensure programs using new software appropriation submit budget exhibits in the approved format.	SAE with USD(C), CAPE	FY 22 POM

SWAP concept paper recommendations related to this recommendation

10C	Budgets should be constructed to support the full, iterative life cycle of the software being procured with amount proportional to the criticality and utility of the software.
Visits	Construct budget to support the full, iterative life cycle of the software.

SWAP working group inputs (reflected in Appendix F) related to this recommendation

Acq	Revise 10 USC 2214 to allow funding approved by Congress for acquisition of a specific software solution to be used for research and development, production, or sustainment of that software solution, under appropriate conditions.
App	A new multi-year appropriation for Digital Technology needs to be established for each Military Defense Department and the Fourth Estate.
App	Components will program, budget, and execute for information and technology capabilities from one appropriation throughout life cycle rather than using RDT&E, procurement, or O&M appropriations—often applied inconsistently and inaccurately—allowing for continuous engineering.
Con	Congress establishes new authority for contracting for SW development and IT modernization.
M&S	Revise 10 USC 2460 to replace the "software maintenance" with "software sustainment" and use a definition that is consistent with a continuous engineering approach across the life cycle.
M&S	A DoD Working Group should be established to leverage ongoing individual Service efforts and create a DoD contracting and acquisition guide for software and software sustainment patterned after the approach that led to creation of the DoD Open Systems Architecture Contracting Guide.
M&S	Acquisition Strategy, RFP/Evaluation Criteria, and Systems Engineering Plan should address software sustainability and transition to sustainment as an acquisition priority.
Con	Manage programs at budget levels, allow programs to allocate funds at project investment level.
Con	Work with appropriators to establish working capital funds so that there is not pressure to spend funds sooner than when you're ready (iterative contracts may produce more value with less money).

Related recommendations from previous studies

GAO15	When assigning resources to all activities, the schedule should reflect the resources (labor, materials, travel, facilities, equipment, and the like) needed to do the work, whether they will be available when needed, and any constraints on funding or time.
GAO17	Hold suppliers accountable for delivering high-quality parts for their products through activities including regular supplier audits and performance evaluations of quality and delivery.

GAO17	Prioritize investments so that projects can be fully funded and it is clear where projects stand in relation to the overall portfolio.
CSIS18	Performance Based Logistics (PBL) contracts should have a duration that allows for tuning and re-baselining with triggered options and rolling extensions.
Sec809	Rec. 41: Establish a sustainment program baseline, implement key enablers of sustainment, elevate sustainment to equal standing with development and procurement, and improve the defense materiel enterprise focus on weapon system readiness.
Sec809	Rec. 42: Reduce budgetary uncertainty, increase funding flexibility, and enhance the ability to effectively execute sustainment plans and address emergent sustainment requirements.

Additional Recommendation A3
Metrics for Cost Assessment and Performance Estimates

Line of Effort	Refactor statutes and regulations for software.
Recommendation	**Require cost assessment and performance estimates for software programs (and software components of larger programs) of appropriate type be based on metrics that track speed and cycle time, security, code quality, and functionality.**
Stakeholders	CAPE, CMO, USD(A&S), Service CMOs and SAEs
Background	Current software cost estimation and reporting processes and procedures in DoD have proven to be highly inaccurate and time consuming. New metrics are required that match the DevSecOps approach of continuous capability delivery and maintenance and provide continuous insight into program progress.
Desired State	Program oversight will re-focus on the value provided by the software as it is deployed to the warfighter/user and will rely more heavily on metrics that can be collected in a (semi-)automated fashion from instrumentation on the DevSecOps pipeline and other parts of the infrastructure. Specific metrics will depend on the type of software rather than a one-size-fits-all approach.
Role of Congress	Congress needs to emphasize the need for new software acquisition reporting that focuses on value provided for the investment in software and frequency of deployments to the warfighter/user. Congress needs to work with CAPE and USD(A&S) to provide feedback on meaningful content and level of detail in reporting.

Draft Implementation Plan		**Lead Stakeholders**	**Target date**
A3.1	Identify (or hire) a small team (3-4) programmers to implement software for automated collection and analysis of metrics and provide them with a modern development environment.	CAPE, DDS	Q4 FY19
A3.2	Identify low-level metrics that are already part of standard commercial development environments (see Appendix C for reporting approach and Appendix E.2 (DIB's "Metrics for Software") for initial lists).	CAPE, SAO	MVP[1] Q4 FY19, then quarterly
A3.2a	Speed and cycle time: launch → initial use, cycle time	Dev team, users	
A3.2b	Code quality: unit test coverage, bug burn-rate, bugs-in-test:bugs-in-field	Dev team, users	
A3.2c	Security: patch → field, OS upgrade → field, HW/OS age	Dev team, users	
A3.2d	Functionality: user satisfaction, number/type of features/cycle	Dev team, users	
A3.2e	Cost: head count, software license cost, compute costs	Dev team, users	
A3.3	Identify 3-5 ongoing programs that are collecting relevant metrics and that partner with CAPE to collect and use data.	CAPE, A&S, CMO, SAEs	In parallel with A6.2

[1] Minimum viable product (first useful iteration)

A3.4	Create a mechanism to transfer and process low-level metrics from development team to PMO on a continuous basis with selectable levels of resolution across the program.	CAPE, SAEs, PMO	MVP Q4 FY19, then quarterly
A3.5	Begin reporting metrics to Congress as part of annual reporting; iterate on content, level, format.	CAPE, Comp, A&S	FY2020
A3.6	Use initial results to establish expectations for new proposed software or software-intensive projects and integrate use of new cost and performance estimates into contract selection.	A&S, SAEs, CAPE	FY2020
A3.7	Establish ongoing capability within CAPE to update metrics on continuous basis, with input from users (of the data).	CAPE	FY2021
A3.8	Identify and eliminate remaining uses of ESLOC as metric for cost and schedule estimation of software/software-intensive programs.	CAPE, SAEs	FY2022

SWAP working group inputs (reflected in Appendix F) related to this recommendation

Con	Revise estimation models - source lines of code are irrelevant to future development efforts, estimations should be based on the team size and investment focused (Cultural).

Related recommendations from previous studies

SEI01	Effort Estimation: • Utilize most likely effort estimates in proposals and status reports; • Find ways to promote the use of accurate effort estimation and productivity evaluation; • Lowest cost is not equivalent to best value. Question outliers.
OSD06	Adjust program estimates to reflect “high confidence”—defined as a program with an 80 percent chance of completing development at or below estimated cost—when programs are baselined in the Stable Program Funding Account.
SEI10	Don’t require PMO to adopt contractors’ estimate for the program—or else use the difference as PM “reserve.”
SEI10	Change from traditional 50% estimation confidence level to 80% level.
SEI10	DoD should consider use of Vickrey “second price” auction mechanism for acquisition proposal bidding.
SEI15	Use the government’s cost estimates (using perhaps an 80% confidence level) rather than contractors’ estimates as the basis for program budgets and place the difference (if the government’s estimate is larger) in a reserve fund available to program managers with sufficient justification. Contractors’ estimates should be acquired using mechanisms that promote accurate estimates, e.g., using Vickrey auctions, the Truth-Revealing Incentive Mechanism (TRIM), or more standard methods of review and acceptance by independent third parties.
DSB18	Rec 3b: The MDA with the Cost Assessment and Program Evaluation office (CAPE), the USD(R&E), the Service Cost Estimators, and others should modernize cost and schedule estimates and measurements.
DSB18	Rec 3b.1: [DoD] should evolve from a pure SLOC approach to historical comparables as a measurement, and should adopt the National Reconnaissance Office (NRO) approach (demonstrated in Box 5) of contracting with the defense industrial base for work breakdown schedule data to include, among others, staff, cost, and productivity.
DSB18	Rec 3c: The MDA should immediately require the PM to build a program-appropriate framework for status estimation.

Additional Recommendation A4
Simplify Laws and Policies

Line of Effort	Refactor statutes and regulations for software.
Recommendation	**Refactor and simplify Title 10, DFARS, and DoDI 5000.02/5000.75 to remove statutory, regulatory, and procedural requirements that generate delays for acquisition, development, and fielding of software while adding requirements for continuous (automated) reporting of cost, performance (against updated metrics), and schedule.**
Stakeholders	USD(C), CAPE, SAE, Service FM & PA&E, Joint Staff
Background	Current law, regulation, policy, and internal DoD processes make modern software development extremely difficult, requiring substantial and consistent senior leadership involvement. Consequently, DoD is challenged in its ability to scale modern software development practices to meet mission needs. Recommendation A1 (new acquisition pathway) provides a pathway that is optimized for software, but it is also possible to modify existing statutes, regulations, and processes to remove barriers for software.
Desired State	Programs have the ability to rapidly field and iterate new functionality in a secure manner, with continuous oversight based on automated reporting and analytics, and utilizing IA-accredited commercial development tools. Congress has better insight into the status of software programs through improved reporting of relevant metrics (see also Recommendations A3 and D4 on metrics).
Role of Congress	Work with DoD to review current statutes and evaluate their effectiveness for different types of software, removing barriers that add time and interfere with the continuous nature of modern software development. See Appendix F for a list of issues to consider.

Draft Implementation Plan		**Lead Stakeholders**	**Target Date**
A4.1	Submit legislative proposal(s) to simplify Title 10 for software (see also: Sec 809 Panel report).	USD(A&S)	Q3 FY19
A4.2	Convene working group with stakeholders and develop and issue a Directive-Type Memorandum (DTM) for the new simplified software acquisition process.	USD(A&S)	Q1 FY20
A4.3	Issue Service-level guidance for new simplified software acquisition process.	SAE	Q1 FY20
A4.4	Identify initial set of programs using modern software development methods to convert to or utilize new, simplified software acquisition process.	USD(A&S), with SAEs	Q1 FY20
A4.5	Convert DTM to DoD Instruction, incorporating lessons learned during initial program implementation.	USD(A&S)	Q1 FY20
A4.6	Develop and implement training at Defense Acquisition University on new, simplified software acquisition process for all acquisition communities (FM, Costing, PM, IT, SE, etc.).	USD(A&S)	Q1 FY20

SWAP working group inputs (reflected in Appendix F) related to this recommendation

Acq	Ensure appropriate integration of a data strategy and the Department's Cloud Strategy. Examine a Steering Committee approach for management.
Acq	Examine the organizational structure with the intent of achieving a more responsive and flat organizational model that de-conflicts roles and responsibilities between the DoD CIO, the USD(A&S), and the CMO regarding software.
Acq	Re-focus the software acquisition workforce on teaming and collaboration, agility, improved role definition, career path advancement methods, continuing education and training opportunities, incentivization, and empowerment.
Acq	Increase flexibility and agility for software programs by eliminating mandated content for acquisition strategies and authorities in Section 821 of the FY16 NDAA, except for MDAPs.
Acq	Eliminate hardware-centric cost, fielding, and performance goals in 10 USC 2488 (established by Sec 807 of the FY17 NDAA) for software-intensive programs.
Acq	Eliminate Nunn-McCurdy breaches (10 USC 2433) for software-intensive programs and replace with continuous evaluation of software performance metrics.
Acq	Remove statutory definition of "major system" for software-intensive programs in 10 USC 2302 and 2302d to remove confusion, since most software in weapons systems inherently functions together to fulfill a mission need.
Acq	Develop language for 10 USC 2366 that allows exemption for software-intensive programs, where DOT&E must justify adding the program for oversight with the MDA and must streamline the process.
Acq	Only require DOT&E oversight for software-intensive programs when requested by the SAE, USD(A&S), or Congress, or if the program is an MDAP.
Acq	For the Fourth Estate, combine all three authorities for DBS under the DoD CMO. After one year, conduct assessment and make a determination if this should be applied to the Services as well.
Acq	Eliminate the separate annual funding certification process for defense business system from 10 USC 2222 or require that funding certification be merged in to the PPBE process.
Acq	Replace annual configuration steering board (CSBs) for software-intensive programs with board (or equivalent entities) established by the CAE, PEO, or PM [FY09 NDAA Sec 814; DoDI 5000.02].
Acq	Expand the FAR 39 (Acquisition of IT) to allow for one area to drive technology purchases. Unless otherwise stated, no other FAR rules would apply.
Acq	Rewrite FMR Volume 2A, Chapter 1, Section 010212(B) to [1] acknowledge that, for the purpose of modifying or enhancing software, there is no technically meaningful distinction between RDT&E, Procurement, and O&M; [2] eliminate the $250,000 barrier between expenses and investments (i.e., stop explicitly tying to a dollar threshold, the determination of whether software is an expense or an investment).
Acq	Revise or eliminate DoDI 8330.01 to eliminate the following elements for software-intensive programs: [1] NR KPP required; [2] DoD-specific architecture products in the DoDAF format that are labor intensive and of questionable value; [e] Interoperability Support Plans (ISPs) required, where DoD CIO can declare any ISP of "special interest"; [2] requirement of DT authority to provide assessments at MS C; [5] mandates JITC to do interoperability assessments for IT with "joint, multinational, and interagency interoperability requirements."

Acq	Revise PfM policy (DoDD 7045.20) to consider the role of data and metrics, as well as additional portfolios (like NC3), and determine authority for the policy.
Con	Separate Contract requirements (scope, PoP, and price) from technical requirements (backlog, roadmap, and stories).
Con	Use SOO vs. SOW to allow the vendor to solve the objectives how they are best suited.
Con	Establish clear and intuitive guidelines on how and when to apply existing clauses.
Con	Have standard clause applications for each of the above that must be excepted vs. accepted.
D&M	Congress could establish, via an NDAA provision, new data-driven methods for governance of software development, maintenance, and performance. The new approach should require on-demand access to standard (and perhaps real-time) data with reviews occurring on a standard calendar, rather than the current approach of manually developed, periodic reports.
M&S	Title 10 USC 2460 should be revised to replace the term “software maintenance” with the term “software sustainment” and use a definition that is consistent with a continuous engineering approach across the life cycle.
Req	The Joint Staff should consider revising JCIDS guidance to focus on user needs, bypassing the JCIDS process as needed to facilitate rapid software development. Guidance should specifically account for user communities (e.g., Tactical Action Officer (TAO), Maritime Operations Center (MOC) director) that do not have one specific PoR assigned to them, but use multiple systems and data from those systems to be effective.
Req	The Joint Staff should consider revising JCIDS guidance to separate functionality that needs high variability from the functionality that is deemed “more stable” (e.g., types of signals to analyze vs. allowable space for the antenna). Then implement a “software box” approach for each one in which the contours of the box are shaped by the functionality variability.
Req	The Joint Staff should consider revising JCIDS guidance to document stable concepts, not speculative ideas. Acknowledge that software requirement documents will iterate, iterate, iterate. JCIDS must change from a “one-pass” mentality to a “first of many” model that is inherently agile, delegating approval to the lowest possible level.

Related recommendations from previous studies

DSB87	Rec 21: DoD should examine and revise regulations to approach modern commercial practice insofar as practicable and appropriate.
NPS16a	Program offices spend far too much time generating paperwork and navigating the bureaucracy rather than thinking creatively about program risks, opportunities, and key elements of their strategies.
NDU17	Develop and maintain core competencies in diverse acquisition approaches and increase the use of venture-capital-type acquisitions, such as Small Business Innovative Research (SBIR), Advanced Concept Technology Development (ACTD), and Other Transaction Authority (OTA), as mechanisms to draw in non-traditional companies.
NDU17	Encourage employees to study statutes and regulations and explore innovative and alternative approaches that meet the statutory and regulatory intent.
Sec809	Rec. 62: Update the FAR and DFARS to reduce burdens on DoD’s commercial supply chain to decrease cost, prevent delays, remove barriers, and encourage innovation available to the

	Military Services.
Sec809	Rec. 74: Eliminate redundant documentation requirements or superfluous approvals when appropriate consideration is given and documented as part of acquisition planning.
Sec809	Rec. 75: Revise regulations, instructions, or directives to eliminate non-value-added documentation or approvals.
Sec809	Rec. 90: Reorganize Title 10 of the U.S. Code to place all of the acquisition provisions in a single part, and update and move acquisition-related note sections into the reorganized acquisition part of Title 10.

Additional Recommendation A5
Streamlined Processes for Business Systems

Line of Effort	Refactor statutes and regulations for software.
Recommendation	**Create streamlined authorization and appropriation processes for defense business systems (DBS) that use commercially available products with minimal (source code) modification.**
Stakeholders	CMO, USD(A&S), Service CMOs, SAEs, DoD CIO
Background	Current DoD business processes are minimally standardized due to a high number of legacy systems that inhibit business process reengineering. In addition, solicitation for new business systems often insists on customization because DoD is “different,” resulting in hard-to-maintain systems that become obsolete (and possibly insecure) quickly.
Desired State	DoD uses standard commercial packages for enterprise and business services, changing its processes to match those of large industries, allowing its systems to be updated and modified on a much faster cadence. The only specialized defense business systems should be those for which there is no commercial equivalent (to include cases in which minor modifications would be required) and there is a funded internal capability to maintain and update the software at a near-commercial cadence.
Role of Congress	Congressional approval for new software development programs should be based on a clear assessment of the current state of commercial software and the need for DoD-specific customization. In many cases it should be possible to make use of commercial systems and modify the DoD process to be consistent with commercial practice rather than attempting to build and maintain specialized business systems. Support legislative change of 10 USC §2222, as needed.

Draft Implementation Plan		**Lead Stakeholders**	**Target Date**
A5.1	Use a Net Promoter Score (NPS) assessment to identify 10 programs whose customers (soldiers, civilians, or others) believe the functionality could be better executed with commercial software.	CMO, with USD(A&S), Service counterparts	Q4 FY19
A5.2	Using the results of A5.1, select four projects for a more detailed assessment of possible savings and/or efficiency improvements.	CMO, with Service CMOs and business process owners	Q1 FY20
A5.3	Implement COTS opportunities, with contracts in place.	Services, with CMO oversight	Q1 FY21
A5.4	Submit legislative change proposal to modify Title 10 §2222 to reflect the lessons learned through process re-engineering to utilize commercially available system over DoD-specific solutions.	CMO, with USD(A&S) and Service counterparts	FY21

SWAP concept paper recommendations related to this recommendation

10C	Use commercial process and software to adopt and implement standard business practices within the Services.
D&D	For common functions, purchase existing software and change DoD processes to use existing apps.

Related recommendations from previous studies

DSB87	Rec 15: The USD(A) and the ASD(Comptroller) should direct Program Managers to assume that system software requirements can be met with off-the-shelf subsystem and components until it is proved that they are unique.
Sec809	Rec 16: Combine authority for requirements, resources, and acquisition in a single, empowered entity to govern DBS portfolios separate from the existing acquisition chain of command.
Sec809	Rec 18: Fund DBSs [defense business systems] in a way that allows for commonly accepted software development approaches.

Additional Recommendation A6
Enduring Capability

Line of Effort	Refactor statutes, regulations, and processes for software.
Recommendation	**Plan, budget, fund, and manage software development as an enduring capability that crosses program elements and funding categories, removing cost and schedule triggers associated with hardware-focused regulations and processes.**
Stakeholders	USD(A&S), USD(C), SAE, Service FM, HASC, SASC
Background	The current approach to acquiring software is based on projects that have a beginning and end. However, many missions are “enduring capabilities” and need software program and portfolio management that continually and perpetually deliver across the spectrum of new capability, incremental enhancements, and life cycle sustainment. The Department should pilot and then scale methods for appropriating software budgets for these enduring capability programs as an ongoing, regularly evaluated expense, with continuous oversight, rather than large, multi-year development contracts.
Desired State	The Department can manage software acquisition as an activity requiring continuous development, deployment, and sustainment, recognizing that software systems are long-lived and have a continuous need for a level of activity to evolve capabilities and address vulnerabilities. Assessment of progress will be maintained throughout the software lifespan by means of continual user engagement with working software, rather than at large-scale milestone gates that do not map well to the underlying technical activities.
Role of Congress	N/A

Draft Implementation Plan		**Lead Stakeholder**	**Target Date**
A6.1	Modify FMR to implement this continuous funding approach.	USD(C)	Q4 FY19
A6.2	Select and launch five programs to be managed as enduring capability, two-year pilot projects.	USD(A&S) with SAE	Q4 FY19
A6.3	Work with FASAB to create an audit treatment of enduring capability software that has a category distinct from Property, Plant, and Equipment; defaults to treating software as an expense, not an investment; and does not distinguish between development and sustainment.	USD(A&S) with USD(C)	Q4 FY20

SWAP concept paper recommendations related to this recommendation

10C	Budgets should be constructed to support the full, iterative life cycle of the software being procured with amount proportional to the criticality and utility of the software.
D&D	Treat software development as a continuous activity, adding functionality continuously.

Additional Recommendation A7
Portfolio Management

Line of Effort	Refactor statutes, regulations, and processes for software.
Recommendation	**Replace JCIDS, PPB&E, and DFARS with a portfolio management approach to software programs, assigned to "PEO Digital" or an equivalent office in each Service that uses direct identification of warfighter needs to determine allocation priorities for software capabilities.**
Stakeholders	USD(A&S), CAPE, JCS, USD(C), SAE, Service FM & PAE
Background	The current requirements process often drives the development of exquisite requirements that tend to be overly rigid and specific and attempt to describe the properties of systems in dynamic environments years in advance. The speed of requirements development and analysis is out of sync with the pace of technology and mission changes. Most importantly, requirement documents that are developed are often disconnected with the end-user requirements.
Desired State	Software programs are managed using a portfolio approach, in which resources are available for reallocation across programs and funding categories based on the importance and opportunities of given elements of the portfolio. Relevant portfolios are defined based on the linkages between programs of similar function, as defined by OSD and/or Services.
Role of Congress	Congress should approve and monitor metrics of success defined within different portfolios and measure the progress against those metrics in determining allocations of funding to different portfolios (with the decisions within a portfolio made by the portfolio office, which would be held accountable for those decisions).

Draft Implementation Plan		**Lead Stakeholders**	**Target Date**
A7.2	Select initial capability areas in each Service to place under portfolio management by PEO Digital (or equivalent).	SAEs	Q3 FY19
A7.1	Issue guidance for management of software portfolios with a "PEO Digital" or similar office with OSD and/or the Services.	USD(A&S) SAE	Q4 FY19
A7.3	Stand up PEO Digital or equivalent office with necessary resources allocated and aligned.	SAE	Q1 FY20
A7.4	Implement new portfolio management methods for initial program capability areas.	PEO Digital	Q3 FY20
A7.5	Determine intermediate successes of, or required modifications to, portfolio management approach.	PEO Digital	Q1 FY21
A7.6	Establish portfolio management approach as standard work for software.	PEO Digital, SAE	FY22

SWAP working group inputs (reflected in Appendix F) related to this recommendation

App	Within each Component-unique Budget Activity (BA), Budget Line Items (BLINs) align by functional or operational portfolios. The BLINs may be further broken into specific projects to provide an even greater level of fidelity. These projects would represent key systems and supporting activities, such as mission engineering.
App	By taking a portfolio approach for obtaining software-intensive capabilities, the Components can better manage the range of requirements, balance priorities, and develop portfolio approaches to enable the transition of data to information in their own portfolios and data integration across portfolios to achieve mission effects, optimize the value of cloud technology, and leverage and transition to the concept of acquisition of whole data services versus individual systems.
App	This fund will be apportioned to each of the Military Departments and OSD for Fourth Estate execution.
App	Governance: management execution, performance assessment, and reporting would be aligned to the portfolio framework—BA, BLI, project.
Req	OSD and the Joint Staff should consider creating “umbrella” software programs around “roles” (e.g., USAF Kessel Run).

Related recommendations from previous studies

OSD06	Transform the Planning, Programming, and Budgeting, and Execution process and stabilize funding for major weapons systems development programs.
DSB09	The USD(AT&L) aggressively delegate milestone decision authority commensurate with program risk.
DSB09	The USD(AT&L) consider a more effective management and oversight mechanism to ensure joint program stability and improved program outcomes.
DSB09	Consolidate all acquisition oversight of information technology under the USD(AT&L) by moving into that organization those elements of the OASD (NII)/DOD CIO and Business Transformation Agency responsible for IT acquisition oversight. The remainder of OASD (NII)/DOD CIO is retained as it exists today, but should be strengthened as indicated in the previous recommendation.
Sec809	Rec 36: Transition from a program-centric execution model to a portfolio execution model.
Sec809	Rec 37: Implement a defense-wide capability portfolio framework that provides an enterprise view of existing and planned capability, to ensure delivery of integrated and innovative solutions to meet strategic objectives.
Sec809	Rec. 38: Implement best practices for portfolio management.
Sec809	Rec. 39: Leverage a portfolio structure for requirements.

Primary Recommendation B1
Digital Infrastructure

Line of Effort	Create and maintain cross-program/cross-service digital infrastructure.
Recommendation	**Establish and maintain digital infrastructure within each Service or Agency that enables rapid deployment of secure software to the field, and incentivize its use by contractors.**
Stakeholders	A&S, CIO, SAE, USD(C)
Background	Currently, DoD programs each develop their own development and test environments, which requires redundant definition and provisioning, replicated assurance (including cyber), and extended lead times to deploy capability. Small companies and other new entrants have difficulties providing software solutions to DoD because those environments are not available outside the incumbent contractor or because they have to build (and certify) unique infrastructure from scratch.
Desired State	Programs will have access to, and be stakeholders in, a cross-program, modern digital infrastructure that can benefit from centralized support and provisioning to lower overall costs and the burden for each program. Development infrastructure supporting CI/CD and DevSecOps is available as best of breed and GOTS provided so that contractors want to use it, though DoD programs or organizations that want or need to go outside of that existing infrastructure can still do so.
Role of Congress	Congress should track the availability, scale, use, and cost effectiveness of digital infrastructure, with the expectation that overall capacity will expand while unit costs decrease over time. Sufficient funding should be provided on an ongoing basis to maintain and upgrade digital infrastructure and to maintain best-of-breed capability that accelerates software development.

Draft Implementation Plan		**Lead Stakeholder**	**Target Date**
B1.1	Designate organization(s) responsible for creating and maintaining the digital infrastructure for each Service's digital infrastructure. Explore the use of tiered approaches with infrastructure at Service or Program level, as appropriate.	DoD CIO, USD(C) and Services (SAE and Service CIO)	Q3 FY19
B1.2	Designate organization(s) responsible for creating and maintaining digital infrastructure(s) for DoD agencies and organizations, including joint digital infrastructure available to the Services.	USD(A&S), with CIO, CMO	Q3 FY19
B1.3	Provide resources for digital infrastructure, including cloud solutions, pre-approved "drop-ship" local compute capability, approved development environments (see DIB Compute Environment concept paper, Appendix I [Glossary]).	USD(A&S), SAE with CAPE, USD(C)	FY20 budget
B1.4	Define baseline digital infrastructure systems and implement procurement and deployment processes and capability.	Responsible organizations from B1.1, B1.2	Q2 FY20

B1.5	Implement digital infrastructure and provide access to ongoing and new programs.	Responsible organizations from B1.1, B1.2	Q3 FY20
B1.6	Identify acquisition programs to transition to digital infrastructure.	SAE	Q2 FY20
B1.7	Transition programs to digital infrastructure.	SAE, CIO, PEO, PM	Q4 FY20

SWAP concept paper recommendations related to this recommendation

10C	Make computing, storage, and bandwidth, and programmers abundant to DoD developers and users.
D&D	Use validated software development platforms that permit continuous integration & delivery evaluation (DevSecOps platform).
Visits	Separate development of mission-level software from development of IA-accredited platforms.

SWAP working group inputs (reflected in Appendix F) related to this recommendation

T&E	Build the enterprise-level digital infrastructure needed to streamline software development and testing across the full DoD software portfolio.

Related recommendations from previous studies

DSB87	Rec 16: All methodological efforts, especially STARS, should look to see how commercially available software tools can be selected and standardized for DoD needs.
SEI01	Infrastructure: In distributed development activities, get high-quality, secure broadband communications between sites. It is an enabler, not a cost.

Primary Recommendation B2
Automated Testing and Evaluation

Line of Effort	Create and maintain cross-program/cross-service digital infrastructure.
Recommendation	**Create, implement, support, and use fully automatable approaches to testing and evaluation (T&E), including security, that allow high-confidence distribution of software to the field on an iterative basis.**
Stakeholders	DOT&E, USD(A&S), DDR&E(AC), SAE, Service Test Agencies
Background	To deliver SW at speed, rigorous, automated testing processes and workflows are essential. Current DoD practices and procedures often see OT&E as a tailgate process, sequentially after development has completed, slowing down delivery of useful software to the field and leaving existing (potentially poorly performing and/or vulnerable) software in place.
Desired State	Development systems, infrastructure, and practices are focused on continuous, automated testing by developers (with users) with frequency dependent on type of software, but targets cycle times measured in weeks. To the maximum extent possible, system operational testing is integrated (and automated) as part of the development cycle using data, information, and test protocols delivered as part of the development environment. Embedded software in safety-critical systems is tested with high confidence in representative (physical and simulated) environments. Testing and evaluation/certification of COTS components is done once (if justified), and then ATO reciprocity (Rec B3) is applied to enable use in other programs, as appropriate. System-level testing using modeling and simulation ("digital twin") is routinely used.
Role of Congress	DOT&E should provide annual reports to Congress that describe the availability, scale, use, and effectiveness of automated T&E, with the expectation that level/depth of testing will increase at the same time as speed and cycle time are being improved.

Draft Implementation Plan		**Lead Stakeholders**	**Target Date**
B2.1	Establish procedures for fully automated testing on digital infrastructure (Rec B1), updating DoDI 5129.47 and Service equivalents as needed.	USD(A&S), DOT&E, with Service Testers	Q1 FY20
B2.2	Establish processes for automated and red-team-based security testing, including zero-trust assumptions, penetration testing, and vulnerability scanning.	USD(A&S), DOT&E, with Service Testers	Q1 FY20
B2.3	Identify initial programs to use tools and workflows.	SAE	Q1 FY20
B2.4	Implement minimum viable product (MVP) tools and workflows on digital infrastructure (Rec B1).	SAE, DOT&E, with PMOs	Q2 FY20
B2.5	Migrate initial programs to digital infrastructure using automated T&E.	PEO, with Responsible Organizations	Q3 FY20
B2.6	Use tools and workflows, identify lessons learned and improvements (using DevSecOps iterative approach).	Service Testers, with PEO/PM	Q4 FY20

B2.7	Modify tools and workflows; document procedures.	Responsible Organizations, Service Testers	Q4 FY20

SWAP concept paper recommendations related to this recommendation

10C	Automate testing of software to enable critical updates to be deployed in days to weeks, not months or years.
D&D	Create automated test environments to enable continuous (and secure) integration and deployment to shift testing and security left.
Visits	Automate testing of software to enable critical updates to be deployed in days to weeks, not months or years (also requires changes in testing organization).
Visits	Add testing as a service.

SWAP working group inputs (reflected in Appendix F) related to this recommendation

Acq	DOT&E should use test data collected through existing test methodologies present in software-intensive programs and not recommend or prescribe additional independent, one-time test events.
Acq	One-time IOT&Es or cybersecurity test events should not be recommended for software-intensive systems except in specific circumstances if warranted.
T&E	Build the enterprise-level digital infrastructure needed to streamline software development and testing across the full DoD software portfolio.
T&E	DoD should expand DOT&E’s current capability to obtain state-of-the-art cyber capabilities on a fee-for-service basis.

Related recommendations from previous studies

DSB87	Rec 27: Each Service should provide its software Using Commands with facilities to do comprehensive operational testing and life-cycle evaluation of extensions and changes.
SEI12	Merge agile and security best practices (e.g., integrate vulnerability scans into continuous integration process, leverage automated test cases for accreditation validation, adhere to secure coding standards).
SEI16	Employ concurrent testing and continuous integration.
USDS	When issuing a solicitation, it should explain the agile software development process. The solicitation should also describe the required testing of functional requirements and make it clear that testing should be integrated into each sprint cycle.
IDA18a	Analysis of planned operational test lengths indicates that the test scope is generally not long enough, demonstrate operational reliability with statistical confidence.

Primary Recommendation B3
ATO Reciprocity

Line of Effort	Create and maintain cross-program/cross-service digital infrastructure.
Recommendation	**Create a mechanism for Authorization to Operate (ATO) reciprocity within and between programs, Services, and other DoD agencies to enable sharing of software platforms, components, and infrastructure and rapid integration of capabilities across (hardware) platforms, (weapon) systems, and Services.**
Stakeholders	DoD CIO, A&S, Service CIOs, DISA
Background	Current software acquisition practice emphasizes the differences among programs: perceptions around different missions, different threats, and different levels of risk tolerance mean that components, tools, and infrastructure that have been given permission to be used in one context are rarely accepted for use in another. The lack of ATO reciprocity drives each program to create their own infrastructure, repeating time- and effort-intensive activities needed to certify elements as secure for their own specific context.
Desired State	Modern software components, tools, and infrastructure, once accredited as secure within DoD, can be used appropriately and cost-effectively by multiple programs. Programs can spend a greater percentage of their budgets on developing software that adds value to the mission rather than spending time and effort on basic software infrastructure. Accreditation of COTS components is done once and then made available for use in other programs, as appropriate.
Role of Congress	N/A

Draft Implementation Plan		**Lead Stakeholder**	**Target Date**
B3.1	Issue guidance making reciprocity the default practice in DoD with limited exceptions and update DoDI 8510.01 to reflect updated risk management framework. Exceptions should require signoff by the DoD CIO to discourage their use.	DoD CIO, with Service CIOs	Q3 FY19
B3.2	Establish DoD-wide repository for ATO artifacts with tools and access rules that enable Services to identify existing ATOs and utilize them when possible.	DoD CIO, with Service CIOs, DISA	Q4 FY19
B3.3	Implement procedures and access controls so that Authorizing Officials have visibility over other programs that are using compatible ATOs.	DoD CIO, with Service CIOs, DISA	Q2 FY20
B3.4	Implement mechanisms to allow FedRAMP and other non-DoD security certifications to be used for DoD ATO when appropriate based on intended use and environment.	DoD CIO, with FedRAMP	Q4 FY20

SWAP working group inputs (reflected in Appendix F) related to this recommendation

Sec	As security is “baked in” to software during the development process, people must be educated about what that means as different tools look at different security aspects.
Sec	People must learn to appreciate that speed helps increase security. Security is improved when changes and updates can be made quickly to an application. Using automation, software can be reviewed quickly.
Sec	The AO must also be able to review documentation and make a risk decision quickly and make that decision on the process and not the product.

Related recommendations from previous studies

SEI12	Define criteria for reaccreditation early in the project.
SEI12	Leverage long accreditation approval wait time with frequent community previews.
SEI12	Don’t apply all the information assurance controls blindly.

Additional Recommendation B4
Prioritize Modern Software Development Methods

Line of Effort	Create and maintain cross-program/cross-service digital infrastructure.
Recommendation	**Prioritize secure, iterative, collaborative development for selection and execution of new software development programs (and software components of hardware programs), especially those using commodity hardware and operating systems.**
Stakeholders	USD(A&S), USD(C) DOT&E, SAE, Service Test Agencies
Background	Despite 37+ years of recommendations to stop using waterfall development for software programs, DoD continues to make use of hardware-centric approaches to development for software and software-intensive programs. While portions of the DoD 5000.02 Instructions apply to “Defense Unique Software Intensive” programs and “Incrementally Deployed Software Intensive” programs, these are still waterfall processes with years between the cycles of deployments (instead of weeks). These processes may be appropriate for some (though not all) embedded systems, but they are not the right approach for DoD-specific software running on commercial hardware and operating systems.
Desired State	DoD makes use of commercial software (without customization) whenever possible. When DoD-specific software development is required, contractors with demonstrated ability in the implementation of modern software development processes (e.g., Agile, DevOps, DevSecOps) are prioritized in the selection process and a contract structure is used that enables those methods to be successfully applied. For those applications for which hardware and software development are closely coupled, modern methods are still used as appropriate, especially in terms of information assurance testing.
Role of Congress	Congress should review metrics for performance on software (and software-intensive) programs with the expectation that modern methods of software are able to deliver software to the field quickly, provide rapid and continuous updates of capability, perform extensive automated testing, and track metrics for speed and cycle time, security, code quality, and useful capability.

Draft Implementation Plan		Lead Stakeholders	Target Date
B4.1	Establish metrics for evaluation of software development environments, following DSB 2018 recommendations on software factors and the DIB’s “Development Environment” and “Agile BS Detector” concept papers.	USD(A&S)	Q3 FY19
B4.2	Issue Directive-Type Memorandum (DTM) to specify DoD’s default software development approach is secure, iterative, modular, and collaborative.	USD(A&S)	Q3 FY19
B4.3	Create new DoD Instruction (DoDI) 5000.SW (or update DoDI 5000.02 and 5000.75) to specify DoD’s default software development approach is secure, iterative, modular, and collaborative.	USD(A&S)	Q1 FY20

B4.4	Update courseware at Defense Acquisition University to specify DoD's default software development approach is secure, iterative, modular, and collaborative.	USD(A&S)	Q2 FY20

SWAP concept paper recommendations related to this recommendation

10C	Adopt a DevOps culture for software systems.
D&D	Require developers to meet with end users, then start small and iterate to quickly deliver useful code.
Visits	Adopt a DevOps culture: design, implement, test, deploy, evaluate, repeat.

SWAP working group inputs (reflected in Appendix F) related to this recommendation

Con	Use collaborative tools and libraries so that all content is available to all parties at all times.
Con	Use an agile process to manage structure and technical requirements.
Sec	As security is "baked in" to software during the development process, people must be educated about what that means as different tools look at different security aspects.
Wkf	Incentivize defense contractors to demonstrate their ability to leverage modern software methodologies.
Wkf	Contractor Reform. Adjust future NDAA's to add incentives for defense contractors to use modern development practices. (See FY18NDAA / §§873 & 874)

Related recommendations from previous studies

DSB87	Rec 12: Use evolutionary acquisition, including simulation and prototyping, as discussed elsewhere in this report, to reduce risk.
DSB87	Rec 17: DoD should devise increased productivity incentives for custom-built software contracts and make such incentivized contracts the standard practice.
DSB87	Rec 18: DoD should devise increased profit incentives on software quality.
DSB87	Rec 23: The USD(A) should update DoD Directive 5000.29, "Management of Computer Resources in Major Defense Systems," so that it mandates the iterative setting of specifications, the rapid prototyping of specified systems, and incremental development.
DSB87	Rec 24: DoD STD 2167 should be further revised to remove any remaining dependency on the assumptions of the "waterfall" model and to institutionalize rapid prototyping and incremental development.
DSB87	Rec 29: The USD(A) should develop economic incentives, to be incorporated into standard contracts, to allow contractors to profit from offering modules for reuse, even though built with DoD funds.
DSB87	Rec 30: The USD(A) should develop economic incentives, to be incorporated into all cost-plus standard contracts, to encourage contractors to buy modules and use them rather than build new ones.
DSB87	Rec 31: The USD(A) and ASD(Comptroller) should direct Program Managers to identify in their programs those systems, components, and perhaps even modules that may be expected to be acquired rather than built, and to reward such acquisition in the RFPs.
SEI12	Make sure Agile project teams understand the intent behind security requirements and organize the backlog accordingly.
SEI12	Ensure agile development processes produce and maintain "just enough" design documentation.

SEI12	Make sure there is at least one person with strong security analysis expertise on the Agile project team.
SEI12	Foster Agile project team and accrediting authority collaboration.
SEI12	Leverage unclassified environments for agile development and community previews.
SEI12	Agile and the information assurance community must join forces to continue improving information assurance processes.
GAO16a	Establish a department policy and process for the certification of major IT investments' adequate use of incremental development, in accordance with OMB's guidance on the implementation of FITARA.
NPS16a	Systems leveraging open architectures and incremental designs can focus on delivering initial capability quickly and then iterate improvements over time. The DoD can tailor acquisition processes for each major type of system to streamline each program's path through focused guidance.
SEI16	Ensure that the RFP contains language that allows the use of Agile. One promising approach that is consistent with Agile is to make sure the original contract is written with Agile in mind and contains sufficient flexibility to permit a wide scope of activity that could be modified as the situation develops. Agile program managers (PMs) could establish contract vehicles that allow for collaborative discussions to resolve and address dynamic developments over the life of the effort.
DSB18	Requests for proposals (RFPs) for acquisition programs entering risk reduction and full development should specify the basic elements of the software framework supporting the software factory, including code and document repositories, test infrastructure, software tools, check-in notes, code provenance, and reference and working documents informing development, test, and deployment.
DSB18	Rec 1: A key evaluation criterion in the source selection process should be the efficacy of the offeror's software factory.
DSB18	Rec 1a: Establish a common list of source selection criteria for evaluating software factories for use throughout the Department.
DSB18	Rec 1b: Competing contractors should have to demonstrate at least a pass-fail ability to construct a software factory.
DSB18	Rec 1c: Criteria for evaluating software factories should be reviewed and updated every five years.
DSB18	Rec 5e: Defense prime contractors must build internal competencies in modern software methodologies.
DSB18	Rec 2: The DoD and its defense industrial base partners should adopt continuous iterative development best practices for software, including through sustainment.
DSB18	Rec 2c: [DoD should] engage Congress to change statutes to transition Configuration Steering Boards (CSB) to support rapid iterative approaches (Fiscal Year (FY) 2009 National Defense Authorization Act (NDAA), Section 814).
DSB18	Rec 2d: [DoD] should require all programs entering Milestone B to implement these iterative processes for Acquisition Category (ACAT) I, II, and III programs.
DSB18	Rec 4a: For ongoing development programs, the USD(A&S) should immediately task the PMs with the PEOs for current programs to plan transition to a software factory and continuous iterative development.
DSB18	Rec 4c: Defense prime contractors should incorporate continuous iterative development into a long-term sustainment plan.
DSB18	Establish a common list of source selection criteria for evaluating software factories for use

	throughout the Department.
FCW18	Contractors would allow government to develop past performance reports with less documentation and less contractor opportunity to appeal their ratings.
USDS	Agile software development is the preferred methodology for software development contracts that contribute to the creation and maintenance of digital services, whether they are websites, mobile applications, or other digital channels.
USDS	Although Part 39 does not directly speak to agile software development practices, it endorses modular contracting principles where information technology systems are acquired in successive, interoperable increments to reduce overall risk and support rapid delivery of incremental new functionality.
USDS	With agile software development, requirements and priorities are captured in a high-level Product Vision, which establishes a high-level definition of the scope of the project, specifies expected outcomes, and produces high-level budgetary estimates.
USDS	Under agile software development, the Government retains the responsibility for making decisions and managing the process; it plays a critical role in the IPT as the Product Owner by approving the specific plans for each iteration, establishing the priorities, approving the overall plan revisions reflecting the experience from completed iterations, and approving deliverables.
USDS	OMB’s 2012 Contracting Guidance to Support Modular Development states that IDIQ contracts may be especially suitable for agile software development because they provide a high level of acquisition responsiveness, provide flexibility, and accommodate the full spectrum of the system life cycle that provides both development and operational products and services. BPAs may work with agile software development using modular contracting methods. Additionally, stand-alone contracts or single-award contracts may be used.
USDS	The Agile process works only if there are appropriate dedicated resources, as the process can be labor intensive. Agencies need to ensure adequate resources are applied to manage their contracts irrespective of the strategy used. Strong contract management ensures projects stay on course and helps prevent the agency from becoming overly reliant on contractors.

Additional Recommendation B5
Cloud Computing

Line of Effort	Create and maintain cross-program/cross-service digital infrastructure.
Recommendation	**Remove obstacles to DoD usage of cloud computing on commercial platforms, including DISA CAP limits, lack of ATO reciprocity, and access to modern software development tools.**
Stakeholders	DoD CIO, Service CIOs, USD(A&S)
Background	Lack of ATO reciprocity and current DoD procedures for cloud are obstacles to leveraging modern infrastructure and tools.
Desired State	DoD developers and contractors are able to use modern cloud computing environments and commercial development tools quickly, with a single certification that is transferable to other groups using the same environment and tools.
Role of Congress	N/A

Draft Implementation Plan		Lead Stakeholders	Target Date
B5.1	Rescind Cloud Access Point (CAP) policy and replace with policy that ensures security at scale (including end-to-end encryption).	DoD CIO	Q3 FY19
B5.2	In conjunction with primary Rec B3, allow transfer of ATOs for commercial platforms between programs and Services.	DoD CIO	Q3 FY19
B5.3	Create specifications and certification process for approval of standard development tools (w/ ATO reciprocity).	DoD CIO	Q4 FY19
B5.4	In conjunction with Rec B1, establish a common, enterprise ability to develop software solutions in the “easy-to-acquire-and-provision” cloud that is fully accredited by design of the process, tools, and pipeline.	USD(A&S)	Q1 FY20

SWAP working group inputs (reflected in Appendix F) related to this recommendation

Acq	Include an approach for enterprise-level DevSecOps and other centralized infrastructure development and management, approach for shared services, and applications management.
Inf	Establish a DoD enterprise ability to procure, provision, pay for, and use cloud that is no different from the commercial entry points for cloud computing.
Inf	DoD should establish a common, enterprise ability to develop software solutions in the “easy-to-acquire-and-provision” cloud that is fully accredited by design of the process, tools, and pipeline.

Related recommendations from previous studies

Sec809	Rec. 43: Revise acquisition regulations to enable more flexible and effective procurement of consumption-based solutions.

Additional Recommendation B6
Certify Code/Toolchain

Line of Effort	Create and maintain cross-program/cross-service digital infrastructure.
Recommendation	**Shift from certification of executables for low- and medium-risk deployments to certification of code/architectures and certification of the development, integration, and deployment toolchain.**
Stakeholders	USD(A&S), SAE, DoD CIO, Service CIO
Background	Today, the typical focus of security accreditation on programs is to certify each version of the code that is intended for release. This works against the goal of frequent updates because the more versions of software that are created, the more often the time and expense of the certification have to be borne by the program.
Desired State	The Department will accredit software infrastructures that are capable of producing quality code when used appropriately, enabling each version of the code produced on that infrastructure to be treated as certifiably secure (within appropriate limits, e.g., for versions that do not entail major architectural changes). With this change in certification, DoD will enable rapid fielding of mission-critical code at high levels of information assurance.
Role of Congress	N/A

Draft Implementation Plan		**Lead Stakeholders**	**Target Date**
B6.1	Identify and use commercial certification procedures for security assessments and deployment mechanisms that can be used for DoD software programs.	CIO	Q4 FY19
B6.2	Identify three lead programs for initial implementation of certification procedures.	A&S, SAE	Q1 FY20
B6.3	Expand certification procedures to 10 additional sites, spanning all Services and multiple OSD offices; update procedures with each new certification to streamline process.	A&S, SAE with CIO	Q3 FY20
B6.4	Update DoDI 8501.01, Risk Management Framework for DoD Information Technology, to reflect revised certification procedures.	CIO with SAE, A&S	Q4 FY20

SWAP working group inputs (reflected in Appendix F) related to this recommendation

Acq	Exempt the DoD from the Clinger Cohen Act, 40 U.S.C. 1401(3)
Inf	DoD should establish a common, enterprise ability to develop software solutions in the "easy-to-acquire-and-provision" cloud that is fully accredited by design of the process, tools, and pipeline.

Related recommendations from previous studies

SEI12	Use common operating environment (COE), software development toolkits (SDKs), and enterprise services to speed up accreditation time.

SEI12	Apply a risk-based, incremental approach to security architecture.
SEI12	Leverage design tactics such as layering and encapsulation to limit impact of change.
SEI13	For an SoS or for the more likely case of a system or component that participates in an existing SoS, an effective risk management approach should: • scale to size and complexity of systems of systems • incorporate dynamics • integrate across full life cycle: requirements to sustainment • focus on success as well as failure

Additional Recommendation B7
Hardware as a Consumable

Line of Effort	Create and maintain cross-program/cross-service digital infrastructure.
Recommendation	**Plan and fund computing hardware (of all appropriate types) as consumable resources, with continuous refresh and upgrades to current, secure operating systems and platform components.**
Stakeholders	USD(A&S), SAE, DoD CIO, Service CIO, USD(C), CAPE
Background	Current information technology (IT) refreshes take 8-10 years from planning to implementation, which means that most of the time our systems are running on obsolete hardware that limits our ability to implement the algorithms required to provide the level of performance needed to stay ahead of our adversaries. Maintaining legacy code for different variants that have hardware capabilities ranging from 2 to 12 years old is an almost impossibly large spread of capability in computing, storage, and communications. From a contracting perspective, this change would require DoD to provide a stable annual budget that paid for new hardware and software capability (see Commandment #3), but this would very likely save money over the longer term.
Desired State	Whenever possible, applications are run in the cloud, so that algorithms can be run on the latest hardware and operating systems. For weapons systems, a continuous hardware refresh mentality is in place that enables software upgrades, crypto updates, and connectivity upgrades to be rapidly deployed across a fleet on an ongoing basis. The adoption rate of the latest hardware and operating system versions is tracked and targets are set for maintaining hardware and operating system "readiness." The paradigm for computing hardware from current Property, Plant, and Equipment categorization (as investments with depreciation schedules) is modified to treat hardware as an expense.
Role of Congress	Provide funding for ongoing replacement of computing hardware as a consumable with a 2–4-year lifetime. Track "readiness" of currently deployed software capability in part by measuring age of the hardware and operating systems on which software is being run.

Draft Implementation Plan		**Lead Stakeholders**	**Target Date**
B7.1	Establish funds for initial existing weapons platforms involving computing hardware to replace hardware every 2–4 years (like oil).	CIO with USD(C), SAE	Q1 FY20
B7.2	Establish draft guidance for determining when to update hardware and operating systems to balance cost with risk/capability.	CIO	Q2 FY20
B7.3	Work with FASAB to change audit treatment of software/IT with these goals: (1) Separate category for software instead of being characterized as Property, Plant, and Equipment; (2) Default setting that software is an expense, not	USD(A&S), in coordination with USD(C)	Q4 FY20

	an investment; and (3) there is no “sustainment” phase for software.		
B7.4	Modify DoD Financial Management Regulation (FMR) to capture changes in how hardware is purchased and retired from service.	USD(C)	Q1 FY21

SWAP concept paper recommendations related to this recommendation

10C	Move to a model of continuous hardware refresh in which computers are treated as a consumable with a 2-3 year lifetime.
Visits	Make use of platforms (hardware and software) that continuously evolve at the timescales of the commercial sector (3-5 years between HW/OS updates).

Related recommendations from previous studies

Sec809	Rec. 44: Exempt DoD from Clinger–Cohen Act Provisions in Title 40:
Sec809	Rec. 56: Use authority in Section 1077 of the FY 2018 NDAA to establish a revolving fund for information technology modernization projects and explore the feasibility of using revolving funds for other money-saving investments.

Primary Recommendation C1
Organic Development Groups

Line of Effort	Create new paths for digital talent (especially internal talent).
Recommendation	**Create software development units in each Service consisting of military and civilian personnel who develop and deploy software to the field using DevSecOps practices.**
Stakeholders	USD(A&S), USD(P&R), SAE, Service HR
Background	DoD's capacity to apply modern technology and software practices to meet its mission is required in order to remain relevant in increasingly technical fighting domains, especially against peer adversaries. While DoD has both military and civilian software engineers (often associated with maintenance activities), the IT career field suffers from a lack of visibility and support. The Department has not prioritized a viable recruiting strategy for technical positions, and there is no comprehensive training or development program that prepares the technical and acquisition workforce to adequately deploy modern software development tools and methodologies.
Desired State	DoD recruits, trains, and retains internal capability for software development, including by service members, and maintains this as a separate career track (like DoD doctors, lawyers, and musicians). Each Service has organic development units that are able to create software for specific needs and that serve as an entry point for software development capability in military and civilian roles (complementing work done by contractors). The Department's workforce embraces commercial best practices for the rapid recruitment of talented professionals, including the ability to onboard quickly and provide modern tools and training in state-of-the-art training environments. Individuals in software development career paths are able to maintain their technical skills and take on DoD leadership roles.
Role of Congress	Congress should receive regular "readiness" reports that include organic software development capability and provide budget required to maintain desired capability level and resources for modern software development.

Draft Implementation Plan		**Lead Stakeholders**	**Target Date**
C1.1	Exercise existing acquisition and cybersecurity hiring authorities to increase the number of software developers in DoD programs with vacant positions.	SAE, PEO, with CIO (cyber excepted service ability)	Immediately
C1.2	Create new military occupational specialty (MOS) and core occupational series plus corresponding career tracks for each Service; use to grow digital talent for modern software development (e.g., Agile, DevSecOps).	J1 and comparable X1 for each Service with USD(P&R)	Q1 FY20
C1.3	Create regulations to allow standard identification, recruitment, and onboarding of experienced civilian software talent, especially on rotation from private sector roles.	USD(P&R)	Q1 FY20

C1.4	Create mechanism for tracking software development expertise and use as preferred experience for promotion into software engineer and acquisition roles.	A&S, CIO	Q2 FY20
C1.5	Obtain additional manpower authorizations for military and civilian SW developers.	USD(A&S), with USD(P&R), SAE	FY20, FY21
C1.6	Stand up one or more software factories within each Service, tied to field needs that can be satisfied through organic software development groups.	SAEs, with PEOs Digital	FY20 (create), FY21 (scale)

SWAP concept paper recommendations related to this recommendation

10C	Establish Computer Science as a DoD core competency.
D&D	Hire competent people with appropriate expertise in software to implement the desired state and give them the freedom to do so (“competence trumps process”).

SWAP working group inputs (reflected in Appendix F) related to this recommendation

M&S	The definition of “core capabilities” in 10 USC 2464 should be revisited in light of warfighter dependence on software-intensive systems to determine the scope of DoD’s core organic software engineering capability, and we should engage with Congress on the proposed revision to clarify the intent and extent of key terminology used in the current statute.
M&S	Revise industrial base policy to include software and DoD’s organic software engineering capabilities and infrastructure. Start enterprise planning and investment to establish and modernize organic System Integration Labs (SILs), software engineering environments, and technical infrastructure; invest in R&D to advance organic software engineering infrastructure capabilities.
Wkf	Develop a core occupational series based on current core competencies and skills for software acquisition and engineering.
Wkf	Overhaul the recruiting and hiring process to use simple position descriptions, fully leverage hiring authorities, engage subject matter experts as reviewers, and streamline the onboarding process to take weeks instead of months.
Wkf	Embrace private-sector hiring methods to attract and onboard top talent from non-traditional backgrounds that may require special authorities to join the Department.
Wkf	Develop a strategic recruitment program that targets civilians, similar to the recruitment strategy for military members, [including] prioritizing experience and skills over cookie-cutter commercial certifications or educational attainment.
Wkf	The Department should incentivize and provide software practitioners access to modern engagement and collaboration platforms to connect, share their skills and knowledge, and develop solutions leveraging the full enterprise.
Wkf	Allow for greater private-public sector fluidity across the workforce while empowering the existing workforce to create a place where they want to work.
Wkf	Modify Title 10, §1596a to create a new Computer-language proficiency pay statute.
Wkf	Pilot a cyber-hiring team with the necessary authorities to execute report recommendations and that can serve as a Department-wide alternative to organization’s traditional HR offices and will provide expedited hiring and a better candidate experience for top-tier cyber positions.

Related recommendations from previous studies

DSB87	Rec 26: Each Service should provide its software Product Development Division with the ability

	to do rapid prototyping in conjunction with users.
DSB87	Rec 36: Establish mechanisms for tracking personnel skills and projecting personnel needs.
DSB87	Rec 37: Structure some office careers to build a cadre of technical managers with deep technical mastery and broad operational overview.
SEI10	Improve compensation and advancement opportunities to increase tenure.

Primary Recommendation C2
Acquisition Workforce Training

Line of Effort	Create new paths for digital talent (especially internal talent).
Recommendation	**Expand the use of (specialized) training programs for CIOs, SAEs, PEOs, and PMs that provide (hands-on) insight into modern software development (e.g., Agile, DevOps, DevSecOps) and the authorities available to enable rapid acquisition of software.**
Stakeholders	USD(A&S), DoD CIO, SAE, Service CIO
Background	Acquisition professionals have been trained and had success in the current model, which has produced the world's best military, but this model is not serving well for software. New methodologies and approaches introduce unknown risks, and acquisition professionals are often not incentivized to make use of the authorities available to implement modern software methods. At the same time, senior leaders in DoD need to be more knowledgeable about modern software development practices so they can recognize, encourage, and champion efforts to implement modern approaches to software program management.
Desired State	Senior leaders, middle management, and organic and contractor-based software developers are aligned in their view of how modern software is procured and developed. Acquisition professionals are aware of all of the authorities available for software programs and use them to provide flexibility and rapid delivery of capability to the field. Program leaders are able to assess the status of software (and software-intensive) programs and spot problems early in the development process, as well as provide continuous insight to senior leadership and Congress. Highly specialized requirements are scrutinized to avoid developing custom software when commercial offerings are available that are less expensive and more capable.
Role of Congress	Prioritize experience with modern software development environments in approval of senior acquisition leaders.

Draft Implementation Plan		Lead Stakeholders	Target Date
C2.1	Leverage existing training venues to add content about modern software development practices.	USD(A&S), SAEs with DAU	Q4 FY19
C2.2	Create and provide training opportunities via boot camps and rotations for acquisition professionals to obtain hands-on experience in DevSecOps programs.	A&S with SAEs, USD(P&R)	FY20 (MVP)[2] FY21 (scale)
C2.3	Develop additional training opportunities for key leaders about modern software development practices.	USD(A&S), SAE, DAU	Q2 FY20
C2.4	Create software continuing education programs and requirements for CIOs, SAEs, PEOs, and PMs, modeled after MCLE (Minimum Continuing Legal Education) for lawyers.	A&S, DAU	Q3 FY20

[2] Minimum viable product (first useful iteration)

SWAP working group inputs (reflected in Appendix F) related to this recommendation

Con	Provide training to KOs, PMs, and leadership to understand the value and methods associated with Agile and modular implementation.
Wkf	Create a software acquisition workforce fund (similar to the existing Defense Acquisition Workforce Development Fund (DAWDF)) ... to hire and train a cadre of modern software acquisition experts.
Wkf	Pilot development programs that provide comprehensive training for all software acquisition professionals, developers, and associated functions.
Con	Provide training to KOs, PMs, and leadership to understand the value and methods associated with Agile and modular implementation.
Con	Educate PMs and KOs on Open Source, proprietary, and government-funded code.

Related recommendations from previous studies

DSB09	All CIOs should approve IT acquisition program manager training and certification and advise the personnel selection process.
DSB09	The USD(AT&L) shall direct the Defense Acquisition University, in coordination with the Information Resources Management College, to integrate the new acquisition model into their curriculum.
DSB18	USD(A&S) should task the PMs of programs that have transitioned successfully to modern software development practices to brief best practices and lessons learned across the Services.
DSB18	Rec 5d: The USD(A&S) and the USD(R&E) should direct the Defense Acquisition University (DAU) to establish curricula addressing modern software practices leveraging expertise from the DDS, the FFRDCs, and the University Affiliated Research Centers (UARCs).
DSB18	Rec 5g: DoD career functional Integrated Product Team (IPT) leads should immediately establish a special software acquisition workforce fund modeled after the Defense Acquisition Workforce Development Fund (DAWDF), the purpose of which is to hire and train a cadre of modern software acquisition experts across the Services.
DSB18	Rec 5h: PMs should create an iterative development IPT with associated training. The Service Chiefs should delegate the role of Product Manager to these IPTs.
DSB18	Rec 5b: The Service Acquisition Career Managers should develop a training curriculum to create and train [a] cadre [of] software-informed PMs, sustainers, and software acquisition specialists.
Sec809	Rec 27: Improve resourcing, allocation, and management of the Defense Acquisition Workforce Development Fund (DAWDF).
Sec809	Rec. 59: Revise the Defense Acquisition Workforce Improvement Act to focus more on building professional qualifications.

Additional Recommendation C3
Increase PMO Experience

Line of Effort	Create new paths for digital talent (especially internal talent).
Recommendation	**Increase the knowledge, expertise, and flexibility in program offices related to modern software development practices to improve the ability of program offices to take advantage of software-centric approaches to acquisition.**
Stakeholders	USD(A&S), SAE, USD(P&R)
Background	Acquisition professionals do not always have experience and insights into modern software development environments, especially in the opportunities (and limitations) for continuous integration/continuous delivery (CI/CD), automated testing (including security testing), and modern cloud-computing architectures. New methodologies and approaches introduce unknown risks, while the old acquisition and development approaches built the world's best military. Program offices are not incentivized to adopt new approaches to acquisition and implementation of software, and inertia represents a barrier to change.
Desired State	Program management offices have staff available with experience in modern software development environments and who are able to make creative (but legal) use of available authorities for acquisition of software to fit the needs of modern software development solutions. Management of most types of software relies on (continuous) measurement of capability delivered to the field rather than being tied to satisfaction of objectives. Time and cost are used as constraints with schedule of delivery of features replanned at each iteration cycle based on warfighter/user feedback.
Role of Congress	N/A

Draft Implementation Plan		**Lead Stakeholders**	**Target Date**
C3.1	Establish list of skills and experience needed by program office staff to be considered "fully staffed" for a software program.	A&S with SAEs, USD(P&R)	Q4 FY19
C3.2	Modify Position Descriptions for those in leadership positions in software acquisition programs to prioritize and reward prior experience in software development.	USD(A&S), SAE, Service HR	Q1 FY20
C3.3	Create and provide training opportunities via boot camps and rotations for acquisition professionals to obtain hands-on experience in DevSecOps programs.	A&S with SAEs, USD(P&R)	Q2 FY20 (MVP)[3] FY21 (scale)
C3.4	Modify PM training requirements to obtain DAU Level IIII certification to include hands-on experience with modern software development.	USD(A&S), DAU	Q3 FY20
C3.5	Evaluate readiness level of software (and software-intensive) program offices by comparing experience/skill sets available with the list of needed skills from C3.1	A&S with SAEs, USD(P&R)	Q4 FY20 (MVP) FY21 (scale)

[3] Minimum viable product (first useful iteration)

	(hint: consider tracking those skills sets; see Action C1.2).		

SWAP concept paper recommendations related to this recommendation

D&D	Hire competent people with appropriate expertise in software to implement the desired state and give them the freedom to do so ("competence trumps process").

SWAP working group inputs (reflected in Appendix F) related to this recommendation

Acq	Lead tester from either DOT&E or JITC (preferably both, if JITC is being used as test org) must be a subject matter expert in the subject being tested, similar to how qualified test pilots run test flights (health records, financial systems, etc.).
Wkf	Empower a small cadre of Highly Qualified Experts (HQEs) and innovative Department employees to execute the changes from this report.
Wkf	Establish a software acquisition workforce fund, similar to the Defense Acquisition Workforce Development Fund (DAWDF), but the primary use will be for hiring and training a cadre of modern software acquisition experts.
Wkf	Provide Agile, Tech, and DevSecOps coaches in Program Offices to support transformations, adoption of modern software practices, and share lessons across the enterprise.
Wkf	Develop a core occupational series based on current core competencies and skills for software acquisition and engineering.
Wkf	Modify the existing language in 5 USC Part III, Subpart D, Chapter 53 to add a pilot training program for all software acquisition professionals, developers, and associated functions.
Wkf	Modify Title 10 §1746 to include authorities for the development of a modern academy under the Defense Acquisition University; the HQE cadre (see above) should lead its development. Note: Tied with FY18 NDAA §891 (training on agile and iterative development methods.)
Wkf	Modify Title 5, §§3371-3375 to expand the Inter-Government Personnel Act and allow more civil service employees to work with non-Federal Agencies and Educational Institutions. In addition, modify Title 10, §1599g to expand the Public-Private Talent Exchange Program and modify the language to reduce the "repayment" period from 1:2 to 1:1 ratio.

Related recommendations from previous studies

OSD06	Establish a consistent definition of the acquisition workforce with the Under Secretary of Defense for Acquisition Technology and Logistics, working with the Service Secretaries to include in that definition all acquisition-related budget and requirements personnel.
OSD06	Immediately increase the number of federal employees focused on critical skill areas, such as program management, system engineering, and contracting. The cost of this increase should be offset by reductions in funding for contractor support.
OSD06	Request that the White House Liaison Office create a pool of acquisition-qualified, White House pre-cleared, non-career senior executives and political appointees to fill executive positions, to provide leadership stability in the Acquisition System.
OSD06	Seek legislation to retain high-performance military personnel in the acquisition workforce to include allowing military personnel to remain in uniform past the limitations imposed by the Defense Officer Personnel Management Act and augment

	their pay to offset the “declining marginal return” associated with retired pay entitlement.
OSD06	Realign responsibility, authority, and accountability at the lowest practical level of authority by reintegrating the Services into the acquisition management structure.
OSD06	Fully implement the intent of the Packard Commission. Create a streamlined acquisition organization with accountability assigned and enforced at each level.
SEI10	Assign PMs, DPMs, and other key positions for the program’s duration and into deployment. Use civilians if military rotations are not amenable.
SEI10	Improve qualifications of acquisition staff, emphasizing software expertise.
CSIS15	Rapid acquisition succeeds when senior leaders are involved in ensuring that programs are able to overcome the inevitable hurdles that arise during acquisition and empower those responsible with achieving the right outcome with the authority to get the job done while minimizing the layers in between.
CSIS15	Rapid acquisition is fundamentally an ongoing dialogue between the acquisition and operational communities about what the real needs of the warfighter are and what the art of the possible is in addressing them.
SEI15	5. Government Personnel Experience. Government personnel with extensive experience in developing and managing acquisition strategy and technical architecture should be dedicated and available to a program throughout its duration.
NPS16a	The growth of rapid acquisition organizations gives acquisition executives new avenues to meet their top priority and rapid capability demands. However, these organizations may also have negative effects on traditional acquisition organizations. The DoD’s top talent will flock to the rapid acquisition organizations so that they can work on high-priority programs with minimal restrictions and likely achieve greater success.
NPS16a	Contracting Officers (COs) must function as strategic partners tightly integrated into the program office, rather than operate as a separate organization that simply processes the contract paperwork.
NPS16b	Culturally, the acquisition community needs to embrace the available tools as opportunities, while being selective with procurement methods and adaptive to the market environment.
GAO17	Empower program managers to make decisions on the direction of the program and to resolve problems and implement solutions.
GAO17	Hold program managers accountable for their choices.
GAO17	Require program managers to stay with a project to its end.
GAO17	Encourage program managers to share bad news, and encourage collaboration and communication.
DSB18	Rec 5a: The service acquisition commands (e.g., the LCMC, the NAVAIR, the U.S. Naval Sea Systems Command (NAVSEA), and the AMC) need to develop workforce competency and a deep familiarity of current software development techniques.
DSB18	Rec 5a.2: Services acquisition commands should use this cadre early in the acquisition process to formulate acquisition strategy, develop source selection criteria, and evaluate progress.
DSB18	Over the next two years, the service acquisition commands need to develop workforce competency and a deep familiarity of current software development techniques.

Sec809	Rec. 40: Professionalize the requirements management workforce.
Sec809	Rec. 46: Empower the acquisition community by delegating below-threshold reprogramming decision authority to portfolio acquisition executives.

Additional Recommendation C4
Recruiting (Transient) Digital Talent

Line of Effort	Create new paths for digital talent (especially internal talent).
Recommendation	**Restructure the approach to recruiting digital talent to assume that the average tenure of a talented engineer will be 2-4 years, and make better use of HQEs, IPAs, special hiring authorities, reservists, and enlisted personnel to provide organic software development capability, while at the same time incentivizing and rewarding internal talent.**
Stakeholders	USD(A&S), USD(P&R), SAE, A-1/G-1/N-1
Background	Current DoD personnel systems assume that military and government employees will “grow through the ranks” and that individuals will stay in government service for long periods of time. The attractions of the private sector create personnel-retention challenges that are not likely to be overcome, so a different approach is needed.
Desired State	DoD leverages all individuals who are willing to serve, whether for a long period or a short period, and amplifies the ability of individuals to make a contribution during their time in government. Internal talent is recognized and retained through merit-based systems of promotion and job assignment.
Role of Congress	Support and encourage the use of existing authorities to hire digital talent in creative ways that match the intent of Congress and solve the need for more flexible arrangements in which talented individuals move in and out of government service (without creating unnecessary barriers).

Draft Implementation Plan		**Lead Stakeholders**	**Target Date**
C4.1	Exercise existing hiring authorities to increase the number of highly skilled software people in DoD program, such as the Cyber Excepted Workforce.	SAE, PEO, CIO	Starting now
C4.2	In conjunction with Recs C1, create a database of individuals in enlisted, officer, reserve, and civilian positions with software development skills and experience for internal recruiting use to software squadrons & PAOs.	USD(P&R) and Service equivalents	Q3 FY19
C4.3	Within organic software programs, create processes for maintaining release cadence under the assumption of up to 25% turnover per year.	PMOs	Q4 FY19
C4.4	Require software-intensive project proposals to include a plan for maintaining cadence-related metrics in the face of up to 25% turnover of staff.	SAEs	Q4 FY19
C4.5	Identify bottlenecks in providing security clearances for software developers and target granting of interim clearances within 1 month of start date.	DSS	Q1 FY20
C4.6	Revise GS and military promotion guidelines for software developers to allow rapid promotion of highly qualified individuals with appropriate skills, independent of “time in grade.”	USD(P&R)	FY20 for FY21 NDAA

C4.7	Obtain additional funding for military, civilian SW developers, including existing personnel, HQEs, IPAs, reservists, and direct commissioning.	USD(A&S), USD(P&R), SAE	FY21

SWAP concept paper recommendations related to this recommendation

10C	Establish Computer Science as a DoD core competency.

SWAP working group inputs (reflected in Appendix F) related to this recommendation

Wkf	Develop a core occupational series based on current core competencies and skills for software acquisition and engineering.
Wkf	Overhaul the recruiting and hiring process to use simple position descriptions, fully leverage hiring authorities, engage subject matter experts as reviewers, and streamline the onboarding process to take weeks instead of months.
Wkf	Embrace private-sector hiring methods to attract and onboard top talent from non-traditional backgrounds that may require special authorities to join the Department.
Wkf	Develop a strategic recruitment program that targets civilians, similar to the recruitment strategy for military members, [including] prioritizing experience and skills over cookie-cutter commercial certifications or educational attainment.

Related recommendations from previous studies

DSB87	Rec 34: Do not believe that DoD can solve its skilled personnel shortage; plan how best to live with it, and how to ameliorate it.
SEI10	Divide large acquisition development efforts into multiple smaller, shorter duration programs.
Sec809	Rec. 45: Create a pilot program for contracting directly with information technology consultants through an online talent marketplace.

Primary Recommendation D1
Source Code Access

Line of Effort	Change the practice of how software is procured and developed.
Recommendation	**Require access to source code, software frameworks, and development toolchains—with appropriate IP rights—for DoD-specific code, enabling full security testing and rebuilding of binaries from source.**
Stakeholders	USD(A&S), CIO, SAE
Background	For many DoD systems, source code is not available to DoD for inspection and testing, and DoD relies on suppliers to write code for new compute environments. As code ages, suppliers are not required to maintain codebases without an active development contract, and “legacy” code is not continuously migrated to the latest hardware and operating systems.
Desired State	DoD has access to source code for DoD-specific software systems that it operates and uses to perform detailed (and automated) evaluation of software correctness, security, and performance, enabling more rapid deployment of both initial software releases and (most importantly) upgrades (patches and enhancements). DoD is able to rebuild executables from scratch for all of its systems, and it has the rights and ability to modify (DoD-specific) code when new conditions and features arise. Code is routinely migrated to the latest computing hardware and operating systems and routinely scanned against currently known vulnerabilities. Modern IP language is used to ensure that the government can use, scan, rebuild, and extend purpose-built code, but contractors are able to use licensing agreements that protect any IP that they have developed with their own resources. Industry trusts DoD with its code and has appropriate IP rights for internally developed code.
Role of Congress	N/A

Draft Implementation Plan		Lead Stakeholders	Target Date
D1.1	Work with industry to modernize policies for software code ownership, licensing, and purchase. See 2018 Army IP directive as an example.	USD(A&S)	Q3 FY19
D1.2	Modify FAR/DFARS guidance to require software source code deliverables for GOTS and for government-funded software development. Obtain rights for access to source code for COTS wherever possible (and useful).	USD(A&S)	Q3 FY20
D1.3	Modify DoDI 5000.02 and DoDI 5000.75 to make access to code and development environments the default.	USD(A&S)	Q3 FY20
D1.4	Develop a comprehensive source-code management plan for DoD including the safe and secure storage, access control, testing, and field of use rights.	USD(A&S), with CIO	Q4 FY20

SWAP concept paper recommendations related to this recommendation

10C	Every purpose-built DoD software system should include source code as a deliverable.

D&D	Require source code as a deliverable on all purpose-built DoD software contracts. Continuous development and integration, rather than sustainment, should be a part of all contracts. DoD personnel should be trained to extend the software through source code or API access.

Related recommendations from previous studies

DSB87	Rec 22: DoD should follow the concepts of the proposed FAR 27.4 for data rights for military software, rather than those of the proposed DoD 27.4, or it should adopt a new “Rights in Software” Clause as Recommended by Samuelson, Deasy, and Martin in Appendix A6.
DSB18	Rec 6b: Availability, cost, compatibility, and licensing restrictions of [the proposed software factory] framework elements to the U.S. Government and its contractors should be part of the selection criteria for contract award.
DSB18	Rec 6c: All documentation, test files, coding, application programming interfaces (APIs), design documents, results of fault, performance tests conducted using the framework, and tools developed during the development, as well as the software factory framework, should be delivered to the U.S. Government at each production milestone; OR escrowed and delivered at such times specified by the U.S. Government (i.e., end of production, contract reward).
DSB18	Rec 6d: Selection preference should be granted based on the ability of the United States to reconstitute the software framework and rebuild binaries, re-run tests, procedures, and tools against delivered software and documentation.

Primary Recommendation D2
Security Considerations

Line of Effort	Change the practice of how software is procured and developed.
Recommendation	**Make security a first-order consideration for all software-intensive systems, recognizing that security-at-the-perimeter is not enough.**
Stakeholders	USD(A&S), CIO, DDS, SAE, DDR&E(AC), DOT&E
Background	Current DoD systems often rely on security-at-the-perimeter as a means of protecting code for unauthorized access. If this perimeter is breached, then a large array of systems can be compromised. Multiple GAO, DoDIG, and other reports have identified cybersecurity as a major issue in acquisition programs.
Desired State	DoD systems use a zero-trust security model in which it is not assumed that anyone who can gain access to a given network or system should have access to anything within that system. Regular and automated penetration testing is used to track down vulnerabilities, and red teams are engaged to attempt to breach our systems before our adversaries do.
Role of Congress	Review (classified) reporting of vulnerabilities identified in DoD systems and provide the resources required to ensure that hardware and operating systems are at current levels (see Recommendation B7, Hardware as a Consumable).

Draft Implementation Plan		Lead Stakeholders	Target Date
D2.1	Adopt standards for secure software development and testing that use a zero-trust security model.	CIO, with DDS	Q3 FY19
D2.2	Develop, deploy, and require the use of IA-accredited (commercial) development tools for DoD software development.	CIO, PEO Digital	Q4 FY19
D2.3	Establish automated and red-team based penetration testing as part of OT&E evaluation (integrated with program development).	DOT&E	Q1 FY20
D2.4	Establish a red team responsible for ongoing vulnerability testing against any defense software system.	CIO with DDS	Q2 FY20
D2.5	Establish security as part of the selection criteria for software programs.	A&S with CIO, SAEs	Q3 FY20

SWAP concept paper recommendations related to this recommendation

10C	Only run operating systems that are receiving (and utilizing) regular security updates for newly discovered security vulnerabilities.
10C	Data should always be encrypted unless it is part of an active computation.
D&D	Create automated test environments to enable continuous (and secure) integration and deployment to shift testing and security left.

SWAP working group inputs (reflected in Appendix F) related to this recommendation

Sec	People must learn to appreciate that speed helps increase security. Security is improved when

	changes and updates can be made quickly to an application. Using automation, software can be reviewed quickly.
Sec	The AO must also be able to review documentation and make a risk decision quickly and make that decision on the process and not the product.
T&E	Establish a statutory “Live Fire” requirement on software-intensive systems as there is on “Covered Systems” for protecting our warfighters from kinetic threats. “Shoot at it” before design is complete and certainly before it is put into the operational environment.
T&E	Establish a federation of state-of-the-art cyber testing capabilities from non-profit institutions to support trusted, survivable, and resilient defense systems and ensure the security of software and hardware developed, acquired, maintained, and used by the DoD.
T&E	Establish cybersecurity as the “4th leg” in measurement of Acquisition system/program performance: Cost, Schedule, Performance, Cybersecurity.
T&E	Develop mechanisms to enforce existing software and cybersecurity policies (from cradle-to-grave) that are not (now) being adequately enforced.
T&E	Ensure each DoD Component is responsible for representing its own forces and capabilities in a digital modeling environment (e.g., M&S and digital twin), making them available to all other DoD users, subject to a pre-defined architecture and supporting standards. DIA will represent threat forces and capabilities in a digital form consistent with this architecture/standards. Programs are required to use DIA-supplied threat models, unless sufficient justification is provided to use other.

Related recommendations from previous studies

DSB09	In the Services and agencies, the CIOs should also have strong authorities and responsibilities for system certification, compliance, applications development, and innovation.
DSB09	The DOD CIO, supported by CIOs in the Services and agencies, should be responsible for certifying that systems and capabilities added to the enterprise do not introduce avoidable vulnerabilities that can be exploited by adversaries.
Sec809	Rec. 77: Require role-based planning to prevent unnecessary application of security clearance and investigation requirements to contracts.

Primary Recommendation D3
Software Features

Line of Effort	Change the practice of how software is procured and developed.
Recommendation	**Shift from the use of rigid lists of requirements for software programs to a list of desired features and required interfaces/characteristics to avoid requirements creep, overly ambitious requirements, and program delays.**
Stakeholders	USD(A&S), Joint Staff, SAEs
Background	Current DoD requirements processes significantly impede DoD's ability to implement modern SW development practices by spending years establishing program requirements and insisting on satisfaction of requirements before a project is considered "done." This impedes rapid implementation of features that are of the most use to the user.
Desired state	Rather than a list of requirements for every feature, programs should establish a minimum set of requirements required for initial operation, security, and interoperability and place all other desired features on a list that will be implemented in priority order, with the ability for DoD to redefine priorities on a regular basis.
Role of Congress	Modify relevant statutes to allow the use of evolving features over rigid requirements and develop alternative methods for obtaining information on program status (See Rec A2, Action A2.4).

Draft Implementation Plan		**Lead Stakeholders**	**Target Date**
D3.1	Modify requirements guidance by memo to shift from a list of requirements for software to a list of desired features and required interfaces/characteristics.	USD(A&S), with CMO	Q4 FY19
D3.2	Update CJCSI 3170.01H (JCIDS requirements process) to reflect contents of guidance memos.	Joint Staff	Q1 FY20
D3.3	Modify DoDI 5000.02 and DoDI 5000.75 (or integrate into new DoDI 5000.SW).	USD(A&S)	Q2 FY20
D3.4	Define and use new budget exhibits for software programs using evolving lists of features in place of requirements (see also Rec A2).	USD(A&S), with USD(C), CAPE, HAC-D, SAC-D	Q3 FY20

SWAP concept paper recommendations related to this recommendation

10C	Adopt a DevOps culture for software systems.
10C	All software procurement programs should start small, be iterative, and build on success—or be terminated quickly.
D&D	Accept 70% solutions in a short time (months) and add functionality in rapid iterations (weeks).

Related recommendations from previous studies

SEI01	Ensure that all critical functional and interoperability requirements are well specified in the contract (statement of work, Statement of Objectives).
SEI01	Handle requirements that have architectural consequences as systems engineering issues—up front.

SEI12	Ensure requirements prioritization of backlog considers business value and risk.
GAO17	Match requirements to resources—that is, time, money, technology, and people—before undertaking new development efforts.

Additional Recommendation D4
Continuous Metrics

Line of Effort	Change the practice of how software is procured and developed.
Recommendation	**Create and use automatically generated, continuously available metrics that emphasize speed, cycle time, security, user value, and code quality to assess, manage, and terminate software programs (and software components of hardware programs).**
Stakeholders	USD(A&S), CAPE, USD(C), SAE, Service Cost Orgs
Background	Current program reporting requirements are largely manual and time consuming, and they provide limited insight into the SW health of a program. New metrics are required that match the DevSecOps approach of continuous capability delivery and maintenance and provide continuous insight into program progress.
Desired State	Program oversight will re-focus on the value provided by the software as it is deployed to the warfighter/user, and it will rely more heavily on metrics that can be collect in an automated fashion from instrumentation on the DevSecOps pipeline and other parts of the infrastructure. Specific metrics will depend on the type of software rather than a one-size-fits-all approach.
Role of Congress	N/A (but see Rec A3)

Draft Implementation Plan		Lead Stakeholder	Target Date
D4.1	Modify acquisition policy guidance to specify use of automatically generated, continuously available metrics that emphasize speed, cycle time, security, and useful functionality.	USD(A&S)	Q3 FY19
D4.2	Modify cost estimation policy guidance to specify use of automatically generated, continuously available metrics that emphasize speed, cycle time, security, and code.	CAPE	Q3 FY19
D4.3	Develop specific measure of software quality, value, and velocity and the tools to implement the automatic generation and reporting.	DDS, with CAPE, CIO, USD(C)	Q4 FY19
D4.4	Modify DoDI 5000.02, DoDI 5000.75, and DoDI 5105.84 to reflect use of updated methods and remove earned value management (EVM) for software programs.	A&S	Q1 FY20

SWAP working group inputs related to this recommendation

Acq	Revise DFARS Subpart 234.201, DoDI 5000.02 Table 8, and OMB Circular A-11 to remove EVM requirement.
Con	Allow for documentation and reporting substitutions to improve agility (agile reporting vs. EVM) (Cultural and EVM Policy).
Con	Establish a clear definition of done targets for software metrics for defense systems of different types (code coverage, defect rate, user acceptance).
D&M	Congress could establish, via an NDAA provision, new data-driven methods for governance of software development, maintenance, and performance. The new approach should require on-demand access to standard (and perhaps real-time) data with reviews occurring on a standard

	calendar, rather than the current approach of manually developed, periodic reports.
D&M	DoD must establish the data sources, methods, and metrics required for better analysis, insight, and subsequent management of software development activities. This action does not require Congressional action but will likely stall without external intervention and may require explicit and specific Congressional requirements to strategically collect, access, and share data for analysis and decision making.
T&E	Establish requirements for government-owned software to be instrumented such that critical monitoring functions (e.g., performance, security) can be automated as much as possible, persistently available, and such that authoritative data can be captured, stored, and reused in subsequent testing or other analytic efforts.

Related recommendations from previous studies

DSB87	Rec 19: DoD should develop metrics and measuring techniques for software quality and completeness and incorporate these routinely in contracts.
DSB87	Rec 20: DoD should develop metrics to measure implementation progress.
Sec809	Rec 19: Eliminate the Earned Value Management (EVM) mandate for software programs using agile methods.
MITRE18	Elevate Security as a Primary Metric in DoD Acquisition and Sustainment.

Additional Recommendation D5
Iterative Development

Line of Effort	Change the practice of how software is procured and developed.
Recommendation	**Shift the approach for acquisition and development of software (and software-intensive components of larger programs) to an iterative approach: start small, be iterative, and build on success or be terminated quickly.**
Stakeholders	USD(A&S), CAPE, USD(C), USD(P&R), SAE, Service HR
Background	Current-language DoD acquisition guidance is largely based around a hardware-centric paradigm, with a well-defined start and end and sequential life cycle activities.
Desired State	Software acquisition in DoD follows an iterative approach, with frequent deployment of working software, supported by a DevSecOps infrastructure that enables speed through continuous integration/continuous delivery. Software projects are continuously evaluated by the quality of their deployed capability and are terminated early if they are found to be non-performant. Software is never “complete.” Programs are viewed as an ongoing service rather than a discrete project.
Role of Congress	Authorize and track software programs that utilize iterative methods of development rather than milestone-based progress. Recognizing that the distinction between RTD&E, procurement, and sustainment is not appropriate for many types of software, identify new ways of providing oversight while enabling much more flexibility for programs.

Draft Implementation Plan		**Lead Stakeholders**	**Target Date**
D4.1	Issue guidance immediately changing the default for acquisition programs to use iterative software development methodologies (e.g., DevSecOps, agile development).	USD(A&S)	Q3 FY19
D4.2	Issue guidance changing the default for acquisition programs to be iterative software development methodologies.	SAE	Q3 FY19
D4.6	Select three software programs widely perceived to be in dire straits and go through a program termination exercise to identify new potential solutions and the blockers to more effectively terminating non-performing programs.	USD A&S	Q1 FY20
D4.3	Modify DoDI 5000.02 and 5000.75 (or DoDI 5000.SW) to reflect more iterative approaches for software development.	USD(A&S)	Q2 FY20
D4.4	Modify Service acquisition policy to reflect more iterative approaches for software development.	SAE	Q2 FY20
D4.5	Build a Congressional Reporting Dashboard that would be available to the four Defense Committees to show the progress of DoD and Services DevSecOps programs, including speed and cycle time, code quality, security, and user satisfaction.	USD(A&S)	Q4 FY20

SWAP concept paper recommendations related to this recommendation

10C	Adopt a DevOps culture for software systems.
10C	All software procurement programs should start small, be iterative, and build on success—or be terminated quickly.
D&D	Accept 70% solutions in a short time (months) and add functionality in rapid iterations (weeks).
D&D	Take advantage of the fact that software is essentially free to duplicate, distribute, and modify.
D&D	Treat software development as a continuous activity, adding functionality continuously across its life cycle.
Visits	Spend time upfront getting the architecture right: modular, automated, secure.

SWAP working group inputs (reflected in Appendix F) related to this recommendation

Con	Treat procurements as investments; “What would you pay for a possible initial capability?”
Con	Leverage incentives to make smaller purchases to take advantage of simplified acquisition procedures.
Con	Use modular contracting to allow for regular investment decisions based on perceived value.
Con	Streamline acquisition processes to allow for replacing poorly performing contractors.
T&E	Develop the enterprise knowledge management and data analytics capability for rapid analysis/ presentation of technical RDT&E data to support deployment decisions at each iterative cycle.

Related recommendations from previous studies

OSD06	Change DoD’s preferred acquisition strategy for developmental programs from delivering 100 percent performance to delivering useful military capability within a constrained period of time, no more than 6 years from Milestone A. This makes time a Key Performance Parameter.
OSD06	Direct changes to the DoD 5000 series to establish Time Certain Development as the preferred acquisition strategy for major weapons systems development programs.
GAO17	Follow an evolutionary path toward meeting mission needs rather than attempting to satisfy all needs in a single step.
GAO17	Ensure that critical technologies are proven to work as intended before programs begin. Assign more ambitious technology development efforts to research departments until they are ready to be added to future generations (or increments) of a product.
NDU17	Prioritize technical performance and project schedules over cost. Maintain aggressive focus on risk identification and management across all elements of the open system, and resolve technical problems as rapidly as possible.
DSB18	Rec 2a: [DoD programs should] develop a series of viable products (starting with MVP) followed by successive next viable products (NVPs).
DSB18	Rec 2b: [DoD programs should] establish MVP and the equivalent of a product manager for each program in its formal acquisition strategy and arrange for the warfighter to adopt the initial operational capability (IOC) as an MVP for evaluation and feedback.
DSB18	Rec 3a: The MDA (with the DAE, the SAE, the PEO, and the PM) should allow multiple vendors to begin work. A down-selection should happen after at least one vendor has proven they can do the work, and should retain several vendors through development to reduce risk, as feasible.

Additional Recommendation D6
Software Research Portfolio

Line of Effort	Change the practice of how software is procured and developed.
Recommendation	**Maintain an active research portfolio into next-generation software methodologies and tools, including the integration of ML and AI into software development, cost estimation, security vulnerabilities, and related areas.**
Stakeholders	USD(R&E), USD(A&S)
Background	Software is essential to national security, and DoD needs to stay ahead of adversaries on emerging SW development practices.
Desired State	DoD benefits from a feedback loop between research and practice, in areas important to retaining the ability to be able to field innovations in software-enabled technologies. Mission needs and a practical understanding of the acquisition ecosystem inform research programs in emerging technologies. Results emerging from research impact the department's warfighting and other systems thanks to high-quality and modular software systems, a DevSecOps infrastructure capable of moving fast, and other enablers. Model-based engineering of software (including "digital twin" approaches) is routinely used to speed development and increase security.
Role of Congress	N/A

Draft Implementation Plan		**Lead Stakeholders**	**Target Date**
D6.1	Designate a responsible person or organization to coordinate software research activities.	USD(R&E)	Q4 FY19
D6.2	Stand up a Chief Engineer for Software to direct the implementation of next-generation software methodologies and tools.	SAEs	Q4 FY19
D6.4	Direct the Principal Civilian Deputy to the SAE to implement the acquisition infrastructure for DevSecOps, allowing quick incorporation of new technologies into DoD systems, implemented by someone with software development experience.	SAEs	Q4 FY19
D6.6	Create a documented DoD Software strategy, perhaps patterned on the DoD cyber strategy,[4] with ties to other existing national and DoD research strategies, and with involvement of A&S and the Services.	USD(R&E)	Q4 FY19
D6.5	Make acquisition data collected continuously from DevSecOps infrastructure and tools available to researchers with appropriate clearances, as a testbed for AI, ML, or other technologies. (See Recs A3, D4)	USD(A&S)	Q4 FY20

Related recommendations from previous studies

DSB18	Rec 7a: Under the leadership and immediate direction of the USD(R&E), the Defense Advanced

[4] https://media.defense.gov/2018/Sep/18/2002041658/-1/-1/1/CYBER_STRATEGY_SUMMARY_FINAL.PDF

	Research Projects Agency (DARPA), the SEI FFRDC, and the DoD laboratories should establish research and experimentation programs around the practical use of machine learning in defense systems with efficient testing, independent verification and validation (IVV), and cybersecurity resiliency and hardening as the primary focus points.

Additional Recommendation D7
Transition Emerging Tools and Methods

Line of Effort	Change the practice of how software is procured and developed.
Recommendation	**Invest in transition of emerging tools and methods from academia and industry for creating, analyzing, verifying, and testing of software into DoD practice (via pilots, field tests, and other mechanisms).**
Stakeholders	USD(A&S), USD(R&E), Service Digital PEOs
Background	Software is essential to national security, and DoD needs to stay ahead of adversaries in implementing emerging SW development practices. Research work at universities and in the private sector, along with best practice implementation from the private sector, can provide valuable tools and methods to be deployed across DoD.
Desired State	Development and test technology, tools, and methods that are being created and used in the private sector and academia and are known and visible to the PEOs Digital who enable transition into Service programs. DoD labs are investing internally and externally to mature software development and analysis tools.
Role of Congress	N/A

Draft Implementation Plan		Lead Stakeholders	Target Date
D7.1	Create a community of practice, code repositories, and other mechanisms to keep all practitioners knowledgeable about the latest trends and capabilities in software development, testing, and deployment.	USD(A&S)	Q4 FY19
D7.2	Invest in and engage with academic and private sector efforts to transition tools to do software engineering: creating, analyzing, verifying, testing, and maintaining software.	Service Digital PEOs, USD(R&E)	FY20

SWAP working group inputs (reflected in Appendix F) related to this recommendation

Req	OSD should consider identifying automated software generation areas that can apply to specific domains.

Related recommendations from previous studies

OSD06	Direct the Deputy Director for Research and Engineering to coordinate service science and technology transition plans with the appropriate military service.
OSD06	Direct the Deputy Director for Research and Engineering to actively participate in the Joint Capabilities Acquisition and Divestment process to reemphasize technology push initiatives.

Additional Recommendation D8
Collect Data

Line of Effort	Change the practice of how software is procured and developed.
Recommendation	**Automatically collect all data from DoD national security systems, networks, and sensor systems, and make the data available for machine learning (via federated, secured enclaves, not a centralized repository).**
Stakeholders	USD(A&S), USD(P&R), SAE, CMO, CAPE, DOT&E, DDR&E(AC)
Background	DoD discards or does not have access to significant amounts of data for its systems and has not established an infrastructure for storing data, mining data, or making data available for machine learning. Current analytical efforts are siloed and under-resourced in many cases.
Desired State	DoD has a modern architecture to collect, share, and analyze data that can be mined for patterns that humans cannot perceive. Data is being used to enable better decision-making in all facets of the Department, providing significant advantages that adversaries cannot anticipate. Data collection and analysis is done without compromising security, and DoD, with minimum exceptions, should have complete data rights for all systems (developed with industry).
Role of Congress	N/A

Draft Implementation Plan		**Lead stakeholders**	**Target Date**
D8.1	Develop comprehensive data strategy for DoD, taking into account future AI/ML requirements,	CDO with USD(A&S), SAE	Q1 FY20
D8.2	Implement a minimum viable product (MVP) that collects and analyzes the most critical data element for one or more programs.	CDO with USD(A&S), SAE	Q3 FY20
D8.3	Create digital data infrastructure to support collection, storage, and processing.	CDO with USD(A&S), SAE	Q1 FY21
D8.4	Require that all new major systems should specify a data collection and delivery plan.	A&S	Q2 FY21
D8.5	Implement data collection requirements for new sensor and weapon system acquisition.	A&S	FY21

SWAP concept paper recommendations related to this recommendation

10C	All data generated by DoD systems—in development and deployment—should be stored, mined, and made available for machine learning.

Related recommendations from previous studies

DSB18	Rec 7b: [USD(R&E)] should establish a machine learning and autonomy data repository and exchange along the lines of the U.S. Computer Emergency Readiness Team (US-CERT) to collect and share necessary data from and for the deployment of machine learning and autonomy.
DSB18	Rec 7c: [USD(R&E)] should create and promulgate a methodology and best practices for the construction, validation, and deployment of machine learning systems, including architectures and test harnesses.

Appendix B: Legislative Opportunities in Response to 2016 NDAA Section 805

(Template Language for Recommendations A1 and A2)

This appendix provides a template for the type of legislative language that could represent a new category/pathway to procure, develop, deploy and continuously improve software for DoD applications, aligned with Recommendations A1 and A2 in Chapter 5. This template is designed to serve as an example of how the types of changes we envision might be implemented and has not been reviewed or endorsed by the Department. It is written to be consistent with 2016 NDAA Section 805 (Use of alternative acquisition paths to acquire critical national security capabilities).

SEC. [???]. SPECIAL PATHWAYS FOR RAPID ACQUISITION OF SOFTWARE APPLICATIONS AND UPGRADES.

(a) GUIDANCE REQUIRED.—Not later than [90, <u>180</u>, 270] days after the date of the enactment of this Act, the Secretary of Defense shall establish guidance authorizing the use of special pathways for the rapid acquisition of software applications and upgrades that are intended to be fielded within one year.

(b) SOFTWARE ACQUISITION PATHWAYS.—

(1) The guidance required by subsection (a) shall provide for the use of proven technologies and solutions to continuously engineer and deliver capabilities in software. The objective of an acquisition under this authority shall be to begin the engineering of new capabilities quickly, to demonstrate viability and effectiveness of those capabilities in operation, and continue updating and delivering new capabilities iteratively afterwards. An acquisition under this authority shall not be treated as an acquisition program for the purpose of section 2430 of title 10, United States Code or Department of Defense Directive 5000.01.

(2) Such guidance shall provide for two rapid acquisition pathways:

(A) APPLICATIONS.—The applications software acquisition pathway shall provide for the use of rapid development and implementation of applications and other software and software improvements running on commercial commodity hardware (including modified or ruggedized hardware) operated by the Department; and

(B) EMBEDDED SYSTEMS.—The embedded systems software acquisition pathway shall provide for the rapid development and insertion of upgrades and improvements for software embedded in weapon systems and other military-unique hardware systems.

(c) EXPEDITED PROCESS.--

(1) IN GENERAL.—The guidance required by subsection (a) shall provide for a streamlined and coordinated requirements, budget, and acquisition process that results in the rapid fielding of software applications and software upgrades to embedded systems in a period of

not more than [one year] from the time that the process is initiated. It shall also require the collection of data on the version fielded and continuous engagement with the users of that software, so as to enable engineering and delivery of additional versions in periods of not more than one year each.

(2) EXPEDITED SOFTWARE REQUIREMENTS PROCESS.—

(A) Software acquisitions conducted under the authority of this provision shall not be subject to the Joint Capabilities Integration and Development System Manual and Department of Defense Directive 5000.01, except to the extent specifically provided in the guidance required by subsection (a).

(B) The guidance required by subsection (a) shall provide that—

(1) Requirements for covered acquisitions are developed on an iterative basis through engagement with the user community, and utilization of user feedback in order to regularly define and prioritize the software requirements, as well as to evaluate the software capabilities acquired;

(2) The requirements process begins with the identification of 1) the warfighter or user need, 2) the rationale for how these software capabilities will support increased lethality and/or efficiency, and 3) the identification of a relevant user community;

(3) Initial contract requirements are stated in the form of a summary-level list of problems and shortcomings in existing software systems and desired features or capabilities of new or upgraded software systems;

(4) Contract requirements are continuously refined and prioritized in an evolutionary process through discussions with users that may continue throughout the development and implementation period;

(5) Issues related to life-cycle costs and systems interoperability are considered; and

(6) Issues of logistics support in cases where the software developer may stop supporting the software system are addressed.

(3) RAPID CONTRACTING MECHANISM.— The guidance required by subsection (a) shall authorize the use of a rapid contracting mechanism, pursuant to which—

(A) Aa contract may be awarded within a [90-day] period after proposals are solicited on the basis of statements of qualifications and past performance data submitted by contractors, supplemented by discussions with two or more contractors determined to be the most highly-qualified, without regard to price;

(B) a contract may be entered for a period of not more than one-year and a ceiling price of not more than [$50 million] and shall be treated as a contract for the acquisition of commercial services covered by the preference in section 2377 of title 10, United States Code;

(C) a contract shall identify the contractor team to be engaged for the work, and substitutions shall not be made during the base contract period without the advance written consent of the contracting officer;

(D) the contractor may be paid during the base contract period on a time and materials basis up to the ceiling price of the contract to review existing software in consultation with the user community and utilize user feedback to define and prioritize software requirements, and to design and implement new software and software upgrades, as appropriate;

(E) a contract may provide for a single one-year option to complete the implementation of one or more specified software upgrades or improvements identified during the period of the initial contract, with a price of not more than [$100 million] to be negotiated at the time that the option is awarded; and

(F) an option under the authority of this section may be entered on a time and materials basis and treated as an acquisition of commercial services or entered on a fixed price basis and treated as an acquisition of commercial products, as appropriate.

(4) EXECUTION OF RAPID ACQUISITIONS.--The Secretary shall ensure that —

(A) software acquisitions conducted under the authority of this provision are supported by an entity capable of regular automated testing of the code, which is authorized to buy storage, bandwidth, and computing capability as a service or utility if required for implementation;

(B) processes are in place to provide for collection of testing data automatically from [entity specified in (A)] and using those data to drive acquisition decisions and oversight reporting;

(C) the Director of Operational Test and Evaluation and the director of developmental test and evaluation participate with the acquisition team to design acceptance test cases that can be automated using the entity specified in (A) and regularly used to test the acceptability of the software as it is incrementally being engineered;

(D) acquisition progress is monitored through close and regular interaction between government and contractor personnel, sufficient to allow the government to understand progress and quality of the software with greater fidelity than provided by formal but infrequent milestone reviews;

(E) an independent, non-advocate cost estimate is developed in parallel with engineering of the software, and is based on an investment-focused alternative to current estimation models, which is not based on source lines of code;

(F) the performance of fielded versions of the software capabilities are demonstrated and evaluated in an operational environment; and

(G) software performance metrics addressing issues such as deployment rate and speed of delivery, response rate such as the speed of recovery from outages and cybersecurity vulnerabilities, and assessment and estimation of the size and complexity of software development effort are established that can be automatically generated on a [monthly, weekly, continuous] basis and made available throughout the Department of Defense and the congressional defense committees.

(5) ADMINISTRATION OF ACQUISITION PATHWAY.—The guidance for the acquisitions conducted under the authority of this section may provide for the use of any of the following streamlined procedures in appropriate circumstances:

(A) The service acquisition executive of the military department concerned shall appoint a project manager for such acquisition from among candidates from among civilian employees or members of the Armed Forces who have significant and relevant experience in modern software methods.

(B) The project manager for each large software acquisition as designated by the service acquisition executive shall report with respect to such acquisition directly, and without intervening review or approval, to the service acquisition executive of the military department concerned.

(C) The service acquisition executive of the military department concerned shall evaluate the job performance of such manager on an annual basis. In conducting an evaluation under this paragraph, a service acquisition executive shall consider the extent to which the manager has achieved the objectives of the acquisition for which the manager is responsible, including quality, timeliness, and cost objectives.

(D) The project manager shall be authorized staff positions for a technical staff, including experts in software engineering to enable the manager to manage the acquisition without the technical assistance of another organizational unit of an agency to the maximum extent practicable.

(E) The project manager shall be authorized, in coordination with the users of the equipment and capability to be acquired and the test community, to make trade-offs among life-cycle costs, requirements, and schedules to meet the goals of the acquisition.

(F) The service acquisition executive or the defense acquisition executive in cases of defense wide efforts, shall serve as the decision authority for the acquisition.

(G) The project manager of a defense streamlined acquisition shall be provided a process to expeditiously seek a waiver from Congress from any statutory or regulatory requirement that the project manager determines adds little or no value to the management of the acquisition.

(6) OTHER FLEXIBLE ACQUISITION METHODS.—The flexibilities provided for software acquisition pathways under this section do not preclude the use of acquisition flexibilities otherwise available for the acquisition of software. The Department may use other transactions authority, broad agency announcements, general solicitation competitive procedures authority under section 879 of the National Defense Authorization Act for Fiscal Year 2017, the challenge program authorized by section 2359b of title 10, United States Code, and other authorized procedures for the acquisition of software, as appropriate. Such authorities may be used either in lieu of or in conjunction with the authorities provided in this section.

(d) FUNDING MECHANISMS.—

(1) SOFTWARE FUND.—

(A) IN GENERAL.—The Secretary of Defense shall establish a fund to be known as the [‘‘Department of Defense Rapid Development of Effective Software Fund’’] to provide funds, in addition to other funds that may be available for acquisition under the rapid software development pathways established pursuant to this section. The Fund shall be managed by a senior official of the Department of Defense designated by the [Under Secretary of Defense for Acquisition and Sustainment]. The Fund shall consist of amounts appropriated to the Fund and amounts credited to the Fund pursuant to section [???] of this Act.

(B) TRANSFER AUTHORITY.—Amounts available in the Fund may be transferred to a military department for the purpose of starting an acquisition under the software acquisition pathway established pursuant to this section. These funds will be used to fund the first year of the software acquisition and provide the Department an opportunity to field software capabilities that address newly discovered needs. A decision to continue the acquisition on other funds will be made based upon the progress demonstrated after the first year. Any amount so transferred shall be credited to the account to which it is transferred. The transfer authority provided in this subsection is in addition to any other transfer authority available to the Department of Defense.

(C) CONGRESSIONAL NOTICE.—The senior official designated to manage the Fund shall notify the congressional defense committees of all transfers under paragraph (2). Each notification shall specify the amount transferred, the purpose of the transfer, and the total projected cost and funding based on the effort required each year to sustain the capability to which the funds were transferred. The senior official will also notify the congressional defense committees at the end of the one-year timeframe and report on the fielded capabilities that were achieved. A notice under this paragraph shall be sufficient to fulfill any requirement to provide notification to Congress for a new start.

(2) PILOT PROGRAM. The Secretary may conduct a pilot program under which funding is appropriated in a single two-year appropriation for life-cycle management of software-intensive and infrastructure technology capabilities conducted under the authority of this section. The objective of the appropriation software pilot program would be to provide 1) greater focus on managed services versus disaggregated development efforts, 2) additional accountability and transparency for information centric and enabling technology capabilities, and 3) flexibility to pursue the most effective solution available at the time of acquisition; 4) much greater insight into the nature of software expenditures across the DOD enterprise; 5) an improved ability to measure costs and program performance;

Appendix C: An Alternative to P-Forms and R-Forms: How to Track Software Programs

Background. DoD's Planning, Programming, and Budgeting System (PPBS) establishes the basis for the budget submission to Congress. Multiple statutes, instructions, and directives must be addressed in order to change the way the budget is put together, adjudicated, enacted and managed. Exhibits are prepared by OSD and DoD Components to support requests for appropriations from Congress and help justify the President's budget. These include a number of forms that are aligned with the existing appropriations process:

- P-Form: Procurement
- R-Form: Research, Development, Test, and Evaluation (RDT&E)
- O-Form: Operations and Maintenance
- M-Form: Military Personnel
- C-Form: Military Construction

As described by the Section 809 panel, the competing objectives of the acquisition system make it very difficult for Congress and the Department to effectively budget and manage defense projects, as illustrated in the following diagram (from the Section 809 panel, Volume 3):

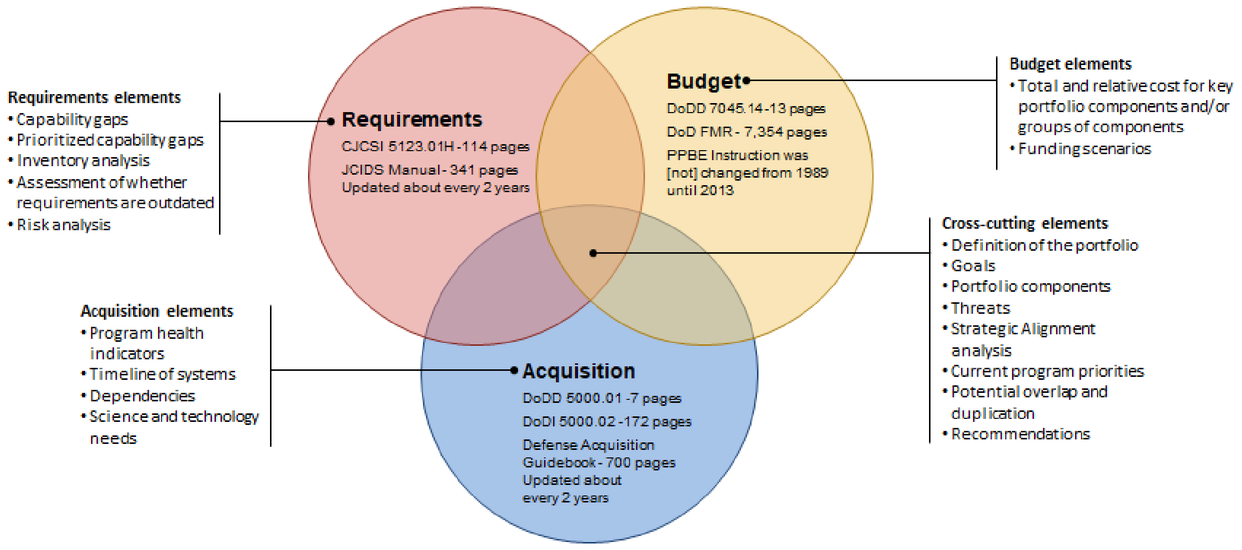

Requirements elements
• Capability gaps
• Prioritized capability gaps
• Inventory analysis
• Assessment of whether requirements are outdated
• Risk analysis
Requirements
CJCSI 5123.01H - 114 pages
JCIDS Manual - 341 pages
Updated about every 2 years
Budget
DoDD 7045.14-13 pages
DoD FMR - 7,354 pages
PPBE Instruction was [not] changed from 1989 until 2013
Budget elements
• Total and relative cost for key portfolio components and/or groups of components
• Funding scenarios
Cross-cutting elements
• Definition of the portfolio
• Goals
• Portfolio components
• Threats
• Strategic Alignment analysis
• Current program priorities
• Potential overlap and duplication
• Recommendations
Acquisition elements
• Program health indicators
• Timeline of systems
• Dependencies
• Science and technology needs
Acquisition
DoDD 5000.01 - 7 pages
DoDI 5000.02 - 172 pages
Defense Acquisition Guidebook - 700 pages
Updated about every 2 years

Figure C.1. Multi-layered DoD budget environment.

In this appendix, we describe a different type of mechanism for budget management for software programs, one that is tuned to the nature of software development. We envision this design to reflect and be interweaved with our primary recommendations—in particular A1 (new acquisition pathway for software) and A2 (new appropriations category for software). It could be also be used for software programs that are making use of other pathways (e.g., traditional DoD 5000.02, mid-tier [Sec 804] acquisition, other transaction authority [OTA] based pathways, or operations and maintenance [O&M]).

Key Characteristics. It is useful to list some of the properties that the new process should satisfy before presenting a specific approach for new methods of managing the budget for software programs. The characteristics that we believe are most important are that the process be:

- *Iterative*: In proposing a new approach for approval and oversight of software programs, we envision a process very similar to the way that software itself is developed: Congress and DoD should articulate what their needs are for oversight and approval of software programs, then try out different ways to gain transparency in proposing and monitoring of software programs. Oversight processes can evolve iteratively, ultimately achieving better oversight

- *Efficient:* The current budget process requires the separate creation of standalone forms and documents that are not a part of the regular information that is maintained and tracked as part of the planning and execution of the software program. Instead, we emphasize the use of automated and machine-readable budget information that is interoperable with financial management tools (with translation to human-readable form when useful).

- *Insightful.* The process should provide insights to both DoD and Congress about the planned and current capabilities of the program and opportunities for portfolio optimization. This includes making use of metrics that are appropriate for software (cycle time, rollback time, automated test coverage, etc.), extracting those metrics in an automated fashion wherever possible, and treating software as an enduring capability.

- *Electronic.* Consistent with the nature of software and software development, the budget artifacts used by Congress and DoD should be largely electronic in nature. By "electronic" we do not mean electronic forms that are "printable" (e.g., PDF and Word files), but rather information that is available in electronic form and requires no further processing to be ingested into analysis systems.

Budget Information for Ongoing Software Programs. Since software is never done, the most important budget artifacts will be those for ongoing programs. The information that is required depends on the type of software, so we briefly describe here our advice for what information should be most relevant in evaluating and renewing the budget of an ongoing program.

- *Type A (commercial off-the-shelf apps):* By its nature, ongoing expenses for COTS apps will be based on the commercial price of the software or service. Existing mechanisms for budgeting materials, supplies, and consumables for DoD functions should be used: usage, spend rate, attainment of (volume) price discounts, etc. It is also important to track resources (money and people) needed to perform upgrades made mandatory by vendor version updates and obsolescence.

- *Type B (customized software)* and *Type C (COTS hardware/operating system):* These classes of software represents custom software that is developed, assured, deployed, and maintained by either organic developers or a contractor/vendor for DoD-specific purposes. Type B software will require primarily configuration management and customization, whereas Type C software will involve customized coding. These types of software are perhaps the least well-suited to the traditional spiral development/hardware-focused acquisition and budgeting process, since they often represent an enduring capability in which new features are continuously added.

The diagram below shows the expected cost profile of a software program of this type, in which the annual cost starts small (and may terminate, if not successful), rises as the software is scaled to its full extent, and then falls as it is optimized and continuously improved.

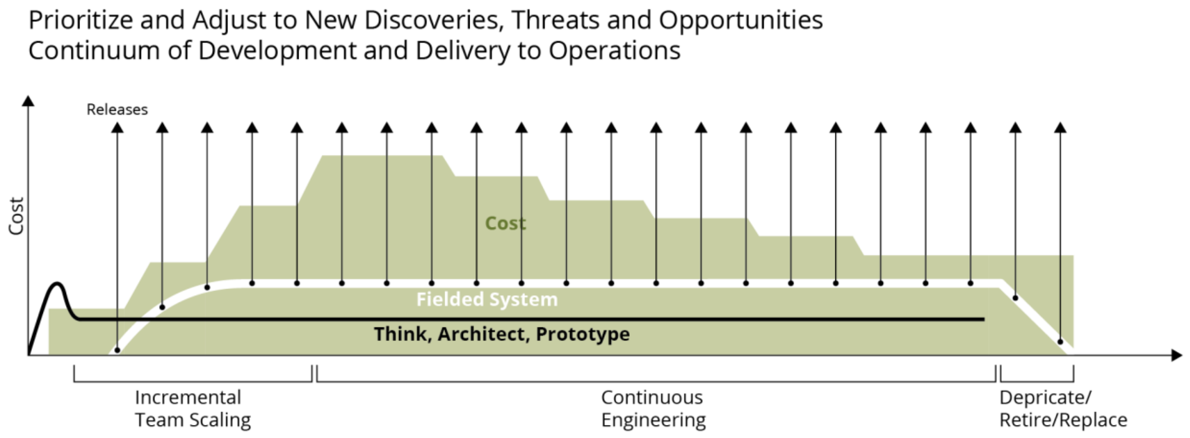


Figure C.2. DevSecOps life cycle cost profile.

The information available as part of the budget process should reflect the following data on the current and desired state of the program:

- List of features implemented and those planned for future releases
- Number of active users and level of satisfaction of the user base
- Time required to field high priority functions (specifications → operations) or fix newly found security holes (discovery → operations)
- Time from code committed to code in use
- Time required for full regression test and cybersecurity audit/penetration testing (and the percentage of such testing that is automated)
- Time required to restore service after outage
- Percentage test coverage of specs/code, including percentage of tests that are automated
- Number of bugs caught in testing versus in field use
- Change failure rate (rollback deployed code)
- Percentage code available to DoD for inspection/rebuild

The cost data associated with the program should include the following information:

- The size and annual cost of the development team, along with the percentage of programmers, designers, user interface engineers, system architects and other key development categories.
- The size and annual cost of the program management team, including both government and contractor program management (if applicable).
- Software licensing fees
- Computing costs (including cloud services)
- Other costs associated with the program

These metrics should be tracked over time, with reports of the past three years of data as well as targets for the coming two years. Annual budget submissions should compare the projected metrics and costs of the program from the past fiscal year with the actual metrics and costs for that period, as well as rolling updating the time horizons to drop the oldest year of tracking data and add the newest year of projected data.

- *Type D (custom hardware and software, including embedded systems):* Embedded systems associated with custom hardware that is still in the development phase is most likely to be reported as part of the hardware development program (using traditional budget items). However, once the software/hardware platform and form factor has been designed then the continued development of the software should be reported in a manner similar to Type C (COTS hardware/operating systems).

Budget Information for New Software Programs. Creating new software programs involves estimation of the cost of the software over at least the initial procurement and deployment phases. Such programs should start small, be iterative, and build on success—or be terminated quickly. Whenever possible, new software programs should have small budgets, require early demonstration of results, and then be turned into ongoing programs (with budget justification as described above). We remark briefly on specific considerations based on the type of software.

- *Type A (commercial off-the-shelf apps)* and *Type B (customized software).* For commercial software of these two types, the most relevant information is the features to be provided by the software, the number of instances of the software expected over time, and the cost of that software (either as purchase cost or licensing costs). For Type B software, additional information should be provided regarding the staffing needs for software configuration, in a manner that is similar to customized software (Type C), though with less intensive development costs.

- *Type C (COTS hardware/operating system).* For custom software running on commodity hardware and operating systems, there are two primary questions that must be addressed: (a) is the software functionality available in commercial products that meets the (primary) needs of the Department and, if not, (b) how large should the initial development effort be in order to create a minimum viable product (MVP) and then begin to scale the initial deployment if successful.

  For comparing customized software to commercially available software, the following information should be provided:

  - A list of features that are desired and an indication of which of those features are available in commercial packages versus those that are DoD-specific.
  - A list of commercial software packages providing similar functionality and the cost of purchasing or licensing that software for initial and full-scale deployment.
  - A justification for why DoD processes cannot be adopted to the development and operations practices of standard commercial approaches and/or why a smaller software development program focused on interfacing DoD specific cases to commercial packages cannot be accomplished.

The goal of providing this information is to ensure that commercial processes/software can be adopted and implemented as standard business practices within DoD. If a DoD-customized software is needed, this information also serves as a good comparison point on the rough costs that are available for related commercial software (when it exists).

- *Type D (custom hardware and software, including embedded systems):* The initial phases of development for custom hardware and software are likely to track hardware development, although in some cases it may be possible to begin software development using emulation and simulation. Care should be taken that embedded software truly requires custom solutions: the trend in commercial software is to establish a layer between hardware and software that allows software to be hardware agnostic (converting Type D into Type C). This approach is quite prevalent in consumer electronics (smart phones and other mobile devices) and transportation systems (automobiles, aircraft).

Software Program Budget Exhibits. Since software programs will be integrated into larger programs and elements of larger programs will have software component, it will be necessary to provide budget exhibits that are compatible with other budget processes used by Congress and DoD. As described above, we believe that the primary information used for tracking ongoing programs should be electronic in nature, and that it should be pulled from existing databases and systems rather than compiled specifically for the budget process.

Following the format used by R-docs, we believe that software programs budget exhibit can be broken down into 5 levels, as shown in the diagram to the right. Each of the exhibits should reflect the information described above (depending on the type of software program) and should exist primarily as electronic databases whose information can be presented in a form consistent with the information that Congress desires.

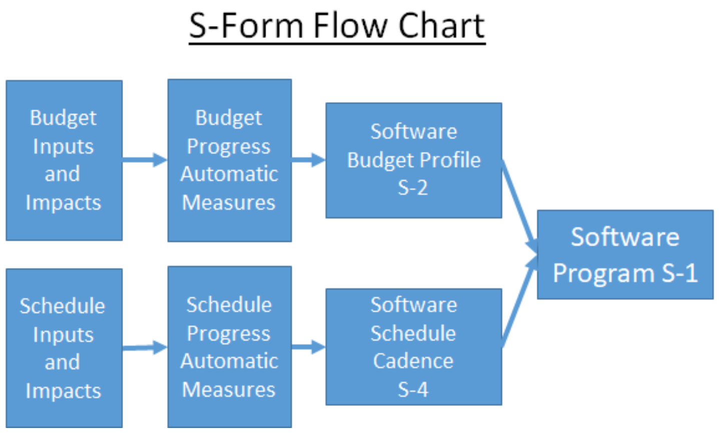

S-Form Flow Chart
Budget Inputs and Impacts
Budget Progress Automatic Measures
Software Budget Profile S-2
Schedule Inputs and Impacts
Schedule Progress Automatic Measures
Software Schedule Cadence S-4
Software Program S-1

Figure C.3. S-Form inputs.

The individual exhibits are as follows:

- S-1 Exhibits: the basic document for presenting DoD's software program information. The S-1 is prepared at the OSD-level, with one exhibit for each separate software appropriation account/portfolio. Because the S-1 is a summary document, all other software exhibits submitted for a program element must reconcile to the numbers shown on the S-1. The S-1 form should be automatically generated from information maintained by the Component headquarters based on information provided (electronically) be individual software program elements.

- S-2 Exhibits: feeds into the S-1 and are automatically populated to provide summary funding information, program description, metrics, and budget justification for each software program element.

- S-4 Exhibits: generate a display of major program releases. This exhibit is required for each project. If a program element consists of only one project, then the S-4 is prepared for the entire program element.

Multi-Element Program Budgets. For the purpose of establishing a new funding authority that will address the continuous improvement nature of software, a coordinated set of budget exhibits must be put in place. Capability elements that are solely software are relatively rare. The hardware platform that the software must run on will either be provided by a different program under a platform-as-a-service (PaaS), or involve computing hardware that is necessarily coincident to a military vehicle (carried in a ship, aircraft, ground or space). When physical space, power, weight and cooling needs for the computer services have to be managed at the vehicle level, a coordination of the design and implementation of the hardware/software environment must be established and managed over a long period—several epochs of lifespans for computer equipment on which the continuously changing software must run. This is a fundamentally different environment than hardware and must be accommodated in a new software budget exhibit, at the right time of development, while managing within the appropriate form-factor.

Fortunately, the PPBS environment has a mechanism for managing this—the multi-Program Element Project. The coordination of research (R-Form), Procurement (P-Form) and Operations (O-Form) with software program information (electronically generated S-Form) can be accommodated in a single project or set of projects in the PPBS. The most limiting case is the one that requires the greatest level of coordination in software-intensive and embedded products. The figure below shows a parallel timeline for the ideation, creation, scaling and implementation phases of software with the spiral nature of hardware for research, engineering/manufacturing development, procurement, operations, sustainment and disposal.

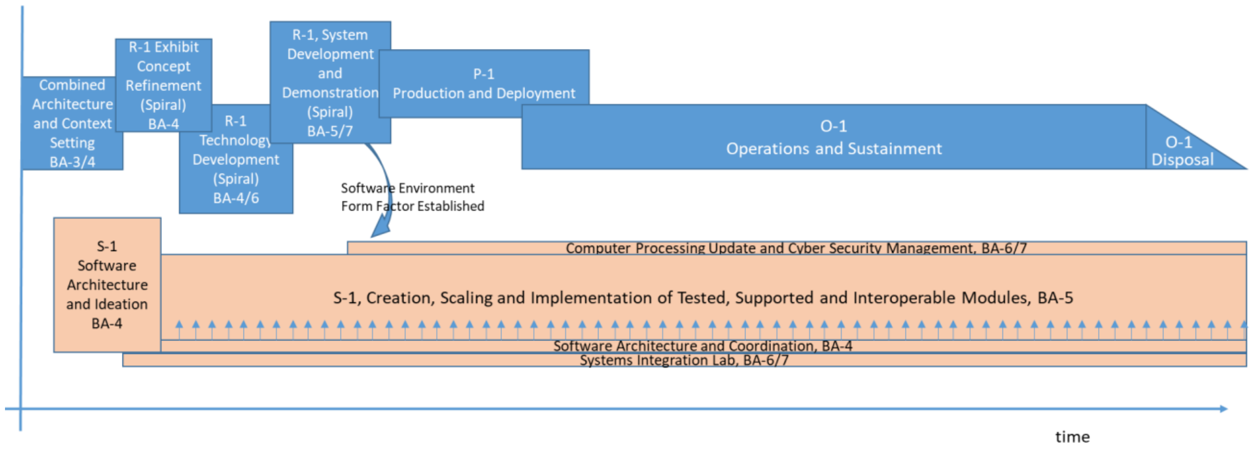

Combined Architecture and Context Setting BA-3/4
R-1 Exhibit Concept Refinement (Spiral) BA-4
R-1 Technology Development (Spiral) BA-4/6
R-1, System Development and Demonstration (Spiral) BA-5/7
P-1 Production and Deployment
O-1 Operations and Sustainment
O-1 Disposal
Software Environment Form Factor Established
S-1 Software Architecture and Ideation BA-4
Computer Processing Update and Cyber Security Management, BA-6/7
S-1, Creation, Scaling and Implementation of Tested, Supported and Interoperable Modules, BA-5
Software Architecture and Coordination, BA-4
Systems Integration Lab, BA-6/7
time

Figure C.4. Budget exhibits by program phase.

Sample Budget Exhibits. To illustrate the type of information that could be presented to Congress as part of the budgeting process, we provide below a sample of some “S-Forms” that might be used to describe a hypothetical software program. For the purposes of illustration, we focus here on a Type C (custom software on commercial hardware/operating system). Other types of software could make use of similar exhibits. We again emphasize that the desired state is that these documents are automatically populated based on electronic databases used within program offices and maintained as part of ongoing development activities.

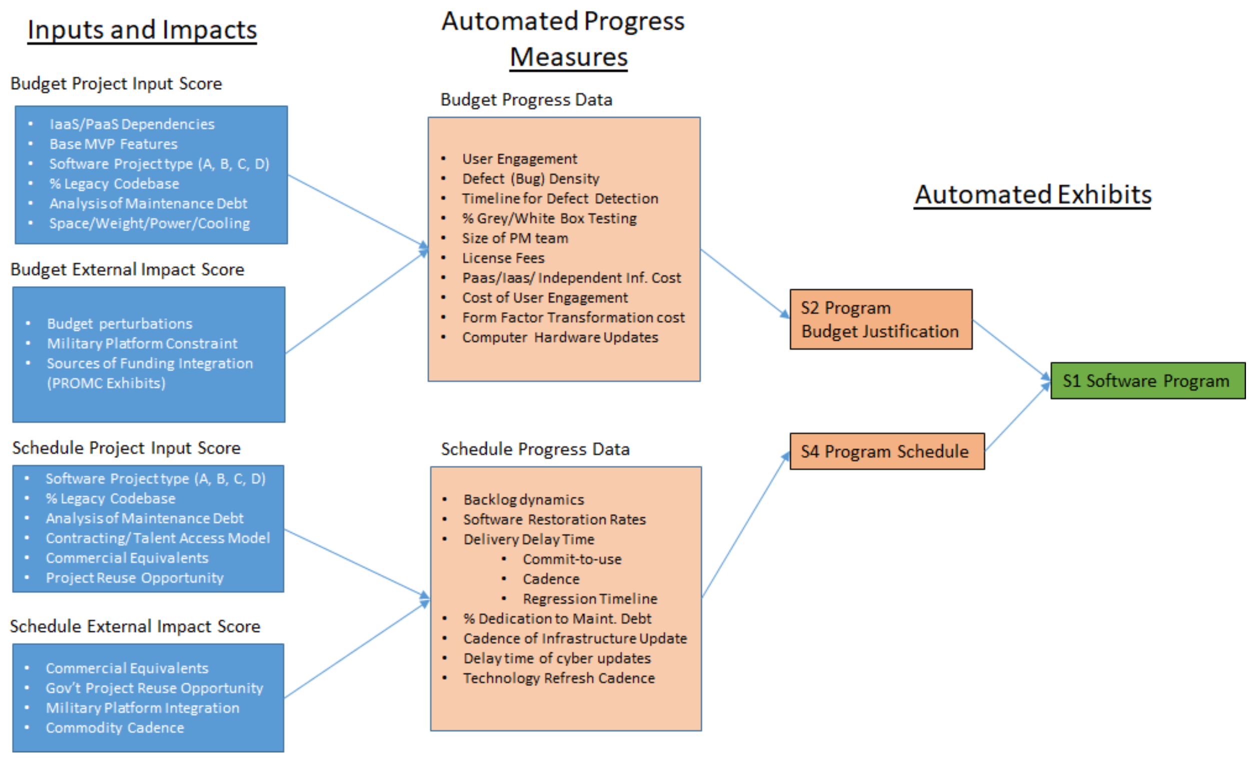

Inputs and Impacts
Budget Project Input Score
IaaS/PaaS Dependencies
Base MVP Features
Software Project type (A, B, C, D)
% Legacy Codebase
Analysis of Maintenance Debt
Space/Weight/Power/Cooling
Budget External Impact Score
Budget perturbations
Military Platform Constraint
Sources of Funding Integration (PROMC Exhibits)
Schedule Project Input Score
Software Project type (A, B, C, D)
% Legacy Codebase
Analysis of Maintenance Debt
Contracting/Talent Access Model
Commercial Equivalents
Project Reuse Opportunity
Schedule External Impact Score
Commercial Equivalents
Gov't Project Reuse Opportunity
Military Platform Integration
Commodity Cadence
Automated Progress Measures
Budget Progress Data
User Engagement
Defect (Bug) Density
Timeline for Defect Detection
% Grey/White Box Testing
Size of PM team
License Fees
Paas/Iaas/ Independent Inf. Cost
Cost of User Engagement
Form Factor Transformation cost
Computer Hardware Updates
Schedule Progress Data
Backlog dynamics
Software Restoration Rates
Delivery Delay Time
Commit-to-use
Cadence
Regression Timeline
% Dedication to Maint. Debt
Cadence of Infrastructure Update
Delay time of cyber updates
Technology Refresh Cadence
Automated Exhibits
S2 Program Budget Justification
S4 Program Schedule
S1 Software Program

Figure C.5. Software progress metrics and budget exhibit crosswalk

Appendix D: Frequently Asked Questions (FAQ)

This document captures some of the common questions and comments that we have received as we discussed the report with various groups.

1. **Haven't all of these ideas already been recommended in previous studies? Why is this study/report any different?**

   Yes, the vision for how to do software right has existed for decades and most of the best practices that we and others have recommended are common practice in industry today. Chapter 3 (Been There, ~~Done~~ Said That) summarizes previous work and provides our assessment of why things haven't changed. Here are the parts we think are new and different:

   - The recommendations in this report serve primarily as documentation of a sequence of iterative conversations and the real work of the report is the engagements before and after the report is released.
   - Our engagements in the process, and the iterative ways we have worked on this study (just like good software!) have created a willing group of advocates (inside the Department) ready to move forward. If we permit them, we believe change will occur.
   - We focus on speed and cycle time as the key drivers for what needs to change and recommend optimizing statutes, regulations, and processes to allow management and oversight of speed at scale. This won't fix everything, but if you optimize for speed then many other things will improve as well (including oversight).
   - This report is shorter and pithier than previous reports, so we hope people will read it.

2. **Shouldn't Congress just get out of the way and let DoD run things the way they want?**

   This is not the way that the Constitution works. The Legislative branch is an equal branch of government and has a responsibility to see that the Executive branch performs its duties well and properly uses taxpayer resources. This makes implementation of many of the ideas in this report a challenge, but we believe that oversight of software is actually *easier* than oversight of hardware, and Congress can and should take advantage of the insights provided by optimizing speed and cycle time to perform oversight of defense software.

3. **Military software is different than commercial software since lives and national security are at stake, so we can't just do things like they do in industry.**

   Not all (defense) software is the same. Some software requires different consideration in DoD compared with industry, but some software is very much equivalent. Foreign governments perform espionage against U.S. companies and those companies should be protecting themselves in the same way as the U.S. government should (and in many cases, companies are doing better at protecting their code than the government, in our experience).

   And even for those types of software that are very different from what we would find in the commercial world, the broad themes of modern software development are the same: software is never done, speed and cycle time are critical measures, software is by people and for people, and software is different from hardware. In all cases we believe that the acquisition of software must recognize these broad themes to take advantage of the opportunities provided by modern software development practices.

While certainly agreeing that the role of military is different, there are many areas of the private sector in which health, economic well-being, and life safety are critically dependent on software - aircraft, hospitals, traffic management, etc.

4. **Embedded software (in weapons systems) is different than commercial software since it is closely tied to hardware, so we can't just do things like they do in industry.**

   Not all software is the same, and embedded systems have different requirements for testing and verification that may not be present in other types of systems. The broad themes of modern software development also hold for embedded systems: software is never done, speed and cycle time are critical measures, software is by people and for people, and software is different from hardware. The issue of cycle time is the one that usually raises the most concern, but we note that embedded software can also have bugs and vulnerabilities and figuring out how to deploy patches and updates quickly is a valuable feature (think about hardware-coupled features in a mobile device or a Tesla as examples of where this is already being done in industry).

5. **For military systems, training is an essential element and we can't change the software quickly because we can't retrain people to use the new version.**

   Not all software is the same and many types of software have functions that are not directly evident to the user. Indeed, there are some types of software where you might want to update things more slowly to avoid creating confusion for a human operating under stress and having to rely on their training to avoid doing something wrong. For those systems, it will be important to figure out how to couple software updates with training so that warfighters have access to the latest version of the software that provides the functionality and security required to carry out their mission. It is also important to continuously evolve our training regimes to take advantage of what may be increased flexibility and adaptability of "digital natives."

6. **Providing source code to the government is a non-starter for industry. How will they make money if they have to give the government their code?**

   It is critical that DoD have access to source code for purpose-build software: it is required in order to do security scans to identify and fix vulnerabilities, and only with access to the source code and build environment can the government maintain code over time. However, providing source code is different than handing over the rights to do anything they want with that code. Modern intellectual property (IP) language should be used to ensure that the government can use, scan, rebuild, and extend purpose-built code, but contractors should be able to use licensing agreements that protect any IP that they have developed with their own resources.

8. **Won't Congress simply reject modern continuous, incremental software programs believing that "software is never done" is just an open invitation to make programs last forever?**

   "Software is never done" specifically highlights that certain capabilities will be enduring, e.g., DoD will always need the capability to ingest data from overhead assets, process that data, and disseminate it and the information it contains. In this situation sensors will change, new analyses will be developed and new products will be required by decision makers. In the traditional DoD software world, a highly defined requirement would be defined, a program would be launched and years later a (likely) out-of-date capability would be delivered, followed immediately by a new, large scale, highly definable requirement, blah, blah, blah. In a world where this need will endure, a continuously funded, incrementally managed software program works better. We must be comfortable that we will spend a certain amount of money each year, we let the program use modern tools for delivering value to real end users incrementally, and we measure success by real-time metrics delivered by the development infrastructure and through direct feedback from the user community. This is the best way to provide Congress with the oversight it deserves.

9. **Have you read a P-Form and an R-Form?**

   We have! To us, these do not seem to be able to provide the type of insight into a software (or software-intensive) program that would be required to make a sound judgement about whether a program is in trouble. In addition, they appear to require substantial manual effort to generate and that effort has relatively little added value, they are missing key metrics that are important to understand whether a software program is on track (speed, cycle time, bugs found in test versus in the field, etc.), and the information they contain is updated to infrequently.

   In Appendix C of our report we describe a different type of mechanism for budget submissions for software programs, one that is tuned to the nature of software development. We believe that it is possible to implement a mechanism for managing software program that makes use of digitally generated information that is part of the ongoing data that are used in the software development process and that provides improved insight into how well that program is delivering value to the end user.

10. **Government will never hire software developers that are as good as industry.**

    While it is certainly true that the vast majority of the highest capability software developers are in the private sector, it is also true that we found extremely capable and dedicated people in the Department—just not nearly enough of them. Actions as consistently detailed in our study can help to address this gap. First, the government should continue to partner with industry and to make use of contractors as a mechanism for obtaining the talent that it needs to develop software that meets its needs. For those cases where it makes sense to use organic (government) software development, the government should make use of existing or new hiring authorities to offer salaries that are as competitive as possible. It is highly unlikely that these will match commercial salaries, but it will show that DoD values software development expertise and that it recognizes that this expertise is in high demand and short supply. On top of this, DoD should anticipate that they will not be able to attract software developers for their entire career. Instead, DoD should have a plan and a set of mechanisms that allow it to hire people for shorter periods of time (e.g., 2-4 years), a period which we believe individuals who are interested in serving their country will be willing to devote. Recommendation C4 (Recruiting (Transient) Digital Talent) provides some ideas for how this might be implemented.

11. **What is the purpose of the use of commercial services guidance in the new acquisition pathway that you propose (Recommendation A1 and Appendix B)?**

    Commercial item procurement was established in 1994 by Congress as a way of encouraging new entrants into the industrial base. While the law was directed at Silicon Valley it also included the vast majority of other types of commercial products at the time — eventually to expand into a greater number of services. Procedures were established (under FAR Part 12) to exempt these types of fixed price contracts from a significant portion of defense-unique acquisition requirements. A preference was also established for the government to buy commercial products and solutions where they existed over defense unique solutions.

    The rapid contracting mechanism in Appendix B would essentially treat all purchases through this mechanism as a commercial item covered under FAR Part 12 to limit DoD from applying unique accounting and oversight procedures applicable to traditional defense contracts. Defining these purchases as commercial item purchases triggers two things: (1) a purchasing preference and (2) relief from regulatory burdens, including government-unique contract clauses and data requirements. The purpose of this language is to ensure this favorable treatment for the alternative acquisition pathway without requiring the contractor to make any proof that is a “commercial” vendor.

12. **Would the use of the proposed acquisition pathway (Recommendation A1) and/or proposed appropriation category (Recommendation A2) be required for all software programs?**

    No. We envision this as becoming the *preferred* pathway for software because it is optimized for software. However, traditional acquisition pathways would still be available.

Appendix E: DIB Guides for Software

As a mechanism for obtaining feedback as it carried out its work, the SWAP study wrote a sequence of short "concept papers" that provided a view on what software acquisition and practice should look like. These documents were released on the DIB website (http://innovation.defense.gov/software) and discussed in DIB public meetings. Feedback from the DIB and other stakeholders was used to iterate on the concept papers. The current snapshot of these papers is provided in this appendix.

List of concept papers:

1. Ten Commandments of Software
2. Metrics for Software Development
3. Do's and Don'ts for Software
4. Detecting Agile BS
5. Is Your Development Environment Holding You Back?
6. Is Your Compute Environment Holding You Back?
7. Site Visit Observations and Recommendations
8. How To Justify Your Agile Budget

The copies of the concept papers in this appendix reflect versions in place as of the approval of this report. We anticipate updating and augmenting these reports as the study continues into the implementation phase. The most up-to-date versions of the concept papers can be found at http://innovation.defense.gove/software.

Defense Innovation Board
Ten Commandments of Software

Executive Summary

The Department of Defense (DoD) must be able to develop and deploy software as fast or faster than its adversaries are able to change tactics, building on commercially available tools and technologies. Recognizing that "software" can range from off-the-shelf, non-customized software to highly-specialized, embedded software running on custom hardware, it is critical that the right tools and methods be applied for each type. In this context we offer the following ten "commandments" of software acquisition for the DoD:

1. Make computing, storage, and bandwidth abundant to DoD developers and users.
2. All software procurement programs should start small, be iterative, and build on success – or be terminated quickly.
3. The acquisition process for software must support the full, iterative life cycle of software.
4. Adopt a DevSecOps culture for software systems.
5. Automate testing of software to enable critical updates to be deployed in days to weeks, not months or years.
6. Every purpose-built DoD software system should include source code as a deliverable.
7. Every DoD system that includes software should have a local team of DoD software experts who are capable of modifying or extending the software through source code or API access.
8. Only run operating systems that are receiving (and utilizing) regular security updates for newly discovered security vulnerabilities.
9. Security should be a first-order consideration in design and deployment of software, and data should always be encrypted unless it is part of an active computation.
10. All data generated by DoD systems - in development and deployment - should be stored, mined, and made available for machine learning.

Motivation and Scope

The latest industry best practices for developing, fielding, and sustaining software applications and information technology (IT) systems are substantially outpacing the US government's industrial-era planning, programming, budgeting, and execution system (PPBES) methods. In the commercial software industry, there is no clear delineation between development, procurement, and sustainment; rather it is a continuous cycle that changes rapidly. New functionality is made available and deployed to users in months to weeks (and even days, for truly critical updates). Existing government appropriation structures make it difficult to implement this approach in the DoD.

Currently available commercial technology for rapidly deploying new advances in software, electronics, networking, and other areas means that our adversaries can rapidly develop new tactics that will be used against us. The only defense is to get inside our adversaries' observe, orient, decide, and act (OODA) loop, which requires the ability to rapidly develop and deploy software into operational environments. For software that is used as part of operations, whether it is run in the Pentagon or in the field, this will require new methods for (automated) testing and rapid deployment of updates, patches, and new functionality.

In this document, we provide ten "commandments" (principles) for DoD software that provide an approach to development that builds on the lessons learned in the software industry and enables rapid deployment of software into military operations. These principles are not universal and may not apply in all situations, but they provide a framework for improving the use of software in DoD operations going forward that we believe will provide substantial improvements compared to the current state of practice.

Software Types

Not all software is alike and different types of software require different approaches for procurement and sustainment. It is important to avoid a "one size fits all" approach to weapons systems. Acquisition practices for hardware are almost never right for software: they are too slow, too expensive, and too focused on enterprise-wide uniformity instead of local customization. Similarly, the process for obtaining software to manage travel is different than what is required to manage the software on an F-35. We suggest a taxonomy with four types of software requiring four different approaches:

- A: commercial ("off-the-shelf") software with no DoD-specific customization required;
- B: commercial software with DoD-specific customization needed;
- C: custom software running on commodity hardware (in data centers or in the field);
- D: custom software running on custom hardware (e.g., embedded software).

While many of the principles below apply to all DoD software, some are relevant only for specific types, as we indicate at the end of each description.

To amplify at the extremes of this continuum of software types, we note especially the tendency of large organizations to believe their needs are unique when it comes to software of Type A. Business processes, financial, human resources, accounting and other “enterprise” applications in DoD are generally not more complicated nor significantly larger in scale than those in the private sector. Commercial software, unmodified, can be deployed in nearly all circumstances. At the opposite end of the spectrum we recognize the highly coupled nature of real-time, mission-critical, embedded software with its customized hardware, denoted in Type D. Examples here include primary avionics or engine control, or target tracking in shipboard radar systems, where requirements such as safety, target discrimination and fundamental timing considerations demand that extensive formal analysis, test, validation and verification activities be carried out.

The DIB’s Ten Commandments of Software

Commandment #1. Make computing, storage, and bandwidth abundant to DoD developers and users, especially in operational systems. Effective use of software requires that sufficient resources are available for computing, storage, and communications. The DoD should adopt a strategy for rapidly transitioning DoD IT to current industry standards such as cloud computing, distributed databases, ubiquitous access to modernized wireless systems (leveraging commercial standards), abundant computing power and bandwidth that is made available as a platform, integration of mobile technologies, and the development of a DoD platform for downloading applications. Unit cost of IT infrastructure and services should be used as a metric in track improvements. An important metric of abundance must be the ability to actually deliver code, and DoD must be able to count the number of programmers within an organization and make sure that the balance of coders to managers is correct [All types]

Commandment #2. All software procurement programs should start small, be iterative, and build on success – or be terminated quickly. Good software development provides value to the customer quickly, based on working with users starting on day one and defining success based on customer value, not creation of code. Large software programs are doomed to fail because of the rigidity, process, competition protests, and bureaucracy that accompany them (typically starting with the Joint Capabilities Integration Development System (JCIDS) process). The separation of software development into research, development, test and evaluation (RDT&E), procurement, and operations & maintenance (O&M) appropriations (colors of money) – and the use of cost-based triggers within each acquisition category (ACAT) – causes delays and places artificial limitations on the program management office’s (PMO’s) ability to quickly meet the changing needs, resulting in increased lifetime cost of software and slower deployment. Modern (“agile”) approaches used in commercial software development will result in faster deployment *and* significant cost savings. [All types, especially B and C]

Commandment #3. The acquisition process for software must support the full, iterative life cycle of software. Software does not age well. It must be constantly maintained and updated, ideally in an automated fashion. The PPBES process is nominally a two (2) year timeline to request and receive funding, with initial planning occurring five (5) years prior to actual receipt, and funding must be requested by intent of use (RDT&E, procurement, and O&M). But this fiscal separation does not match the process of software development, where all creation of code is “development,” whether it falls within the fiscal law definition or not. As an alternative, the DoD should make use of “level of effort” (or capacity) constructs to allow continuous development and testing. Assume that low criticality software that is routinely used will require 10% of the development cost to maintain (per year) and more critical software will likely require more resources. This funding must be planned for at the time of initial development, not as an annual allocation that could be interrupted. Enhanced software capability should never be considered “ahead of need.” [All types]

Commandment #4. Adopt a DevSecOps culture for software systems. “DevOps” represents the integration of software development and software operations, along with the tools and culture that support rapid prototyping and deployment, early engagement with the end user, automation and monitoring of software, and psychological safety (e.g., blameless reviews). “DevSecOps” adds the integration of security at all stages of development and deployment, which is essential for DoD applications. These techniques should be adopted by the DoD, with appropriate tuning of approaches used by the community for mission-critical, national security applications. Open source software should be used when possible to speed development and deployment, and leverage the work of others. Waterfall development approaches (e.g., DoD-STD-2167A) should be banned and replaced with true, commercial agile processes. Thinking of software “procurement” and “sustainment” separately is also a problem: software is never “finished” but must be constantly updated to maintain capability, address ongoing security issues and potentially add or increase performance (see Commandment #3). [Type C; Type D when tied to iterative hardware development and deployment methodologies]

Commandment #5. Automate configuration, testing, and deployment of software to enable critical updates to be deployed in days to weeks, not months or years. While operational test and evaluation (OT&E) is useful, it must not be the pacing item for deployment of software, especially upgrades to existing software. Automated configuration management, unit testing, software/hardware-in-the-loop (SIL/HIL) testing, continuous integration, A/B testing, usage and issues tracking, and other modern tools of software development should be used to provide high confidence in software correctness and enable rapid, push deployment of patches, upgrades, and apps. Make use of modern software development tool sets that support these processes (and other types of development stack automation and software instrumentation) to enable code optimization and refactoring. [All types]

Commandment #6. Every purpose-built DoD software system should include source code as a deliverable. DoD should have the rights to and be able to modify (DoD-specific) code when new conditions and features arise. Providing source code will also allow the DoD to perform detailed (and automated) evaluation of software correctness, security, and performance, enabling more rapid deployment of both initial software releases and (most importantly) upgrades (patches and enhancements). [Types C, D]

Commandment #7. Every DoD system that includes software should have a local team of DoD software experts who are able to modify or extend the software through source code or API access. Modern weapons systems are software-driven and utilization of those systems in a rapidly changing environment will require that the system (software) be customizable by the user. In order to do this, all fielded DoD systems that include software must also have a local team (responsible to the operational unit) that has the skills and permission to modify and extend the software through either source code (Commandment #6) or application programming interface (API) access. Local experts should have “reachback” capabilities to larger team and the ability to pull new features into their code (and generate pull requests for features that they add which should go back into the main codebase [repository]). [Types B, C, sometimes D]

Commandment #8. Only run modern operating systems that are receiving (and utilizing) regular security updates for newly discovered security vulnerabilities. Outdated operating systems are a major vulnerability and the DoD should assume that any computer running such a system will eventually be compromised. Standard practice in industry is that security patches should be applied within 48 hours of release, though this is probably too big a window for defense systems. Treat software vulnerabilities like perimeter defense vulnerabilities: if there is a hole in your perimeter and people are getting in, you need to patch the hole quickly and effectively. [Types A, B, C]

Commandment #9. Security should be a first-order consideration in design and deployment of software, and data should always be encrypted unless it is part of an active computation. All data should be encrypted, whether in motion (across a network) or at rest (memory, disk, cloud, etc). A possible exception is real-time data that is part of an embedded control system and is being sent across an internal bus/network that is not accessible from outside that network. [Types A, B, C and D when possible]

Commandment #10. All data generated by DoD systems – in development and operations – should be stored, mined, and made available for machine learning. Create a new architecture to collect, share, and analyze data that can be mined for patterns that humans cannot perceive. Utilize data to enable better decision-making in all facets of the Department, providing significant advantages that adversaries cannot anticipate. Forge culture of data collection/analysis to meet the demands of a software-centric combat environment. Such data collection and analysis can be done without compromising security: in fact, a comprehensive understanding of the data the DoD collects can improve security. [All types]

Supporting Thoughts and Recommendations

In addition to the ten principles given above, we offer the following thoughts and recommendations for how the DoD can best take advantage of software as a force multiplier. While not directly related to software, they are enablers for adopting the principles required for rapid development and deployment of software.

Establish Computer Science as a DoD core competency. Do not rely solely on contractors as the only source of coding capability for DoD systems. Instead, the DoD should recruit, train, and retain internal capability for software development, including by service members, and maintain this as a separate career track (like DoD doctors, lawyers, and musicians). This should complement work done by civilians and contractors. Create new and expand existing programs to attract promising civilian and military science, technology, engineering and math (STEM) talent. Reach into new demographic pools of people who are interested in the work DoD does but otherwise would not be aware of DoD opportunities. Be able to count the number of programmers within an organization and make sure that the balance of developers to managers is correct

Use commercial process and software to adopt and implement standard business practices within the Services. Modern enterprise-scale software has been optimized to allow business to operate efficiently. The DoD should take advantage of these systems by adopting its internal (non-warfighter specific) business processes to match industry standards, which are implemented in cost-efficient, user-friendly software and software as a service [SaaS] tools. Rather than adopt a single approach across the entire DoD, the individual Services should be allowed to implement complementary approaches (with appropriate interoperability).

Move to a model of continuous hardware refresh in which computers are treated as a consumable with a 2-3 year lifetime. The current approach — in which technology refreshes take 8-10 years from planning to implementation — means that most of the time our systems are running on obsolete hardware that limits our ability to implement the algorithms required to provide the level of performance required to stay ahead of our adversaries. Moving to the cloud provides a solution to this issue for enterprise and other software systems that do not operate on local or specialized hardware. However for weapons systems, a continuous hardware refresh mentality would enable software upgrades, crypto updates, and connectivity upgrades to be rapidly deployed across a fleet, rather than maintaining legacy code for different variants that have hardware capabilities ranging from 2 to 12 years old (an almost impossibly large spread of capability in computing, storage, and communications). From a contracting perspective, this change would require DoD to provide a stable annual budget that paid for new hardware and software capability (see Commandment #3), but this would very likely save money over the longer term.

Defense Innovation Board
Metrics for Software Development

Software is increasingly critical to the mission of the Department of Defense (DoD), but DoD software is plagued by poor quality and slow delivery. The current state of practice within DoD is that software complexity is often estimated based on number of source lines of code (SLOC), and rate of progress is measured in terms of programmer productivity. While both of these quantities are easily measured, they are not necessarily predictive of cost, schedule, or performance. They are especially suspect as measurements of program success, defined broadly as delivering needed functionality and value to users. Measuring the health of software development activities within DoD programs using these obsolete metrics is irrelevant at best and, at worst, could be misleading. As an alternative, we believe the following measures are useful for DoD to track performance for software programs and drive improvement in cost, schedule, and performance.

#	Metric	Target value (by software type)[5]				Typical DoD values for SW
		COTS apps	**Custom -ized SW**	**COTS HW/OS**	**Real-time HW/SW**	
1	Time from program launch to deployment of simplest useful functionality	<1 mo	<3 mo	<6 mo	<1 yr	3-5 yrs
2	Time to field high priority fcn (spec → ops) or fix newly found security hole (find → ops)	N/A <1 wk	<1 mo <1 wk	<3 mo <1 wk	<3 mo <1 wk	1-5 yrs 1-18 m
3	Time from code committed to code in use	<1 wk	<1 hr	<1 da	<1 mo	1-18 m
4	Time req'd for full regression test (automat'd) and cybersecurity audit/penetration testing	N/A <1 mo	<1 da <1 mo	<1 da <1 mo	<1 wk <3 mo	2 yrs 2 yrs
5	Time required to restore service after outage	<1 hr	<6 hr	<1 day	N/A	?
6	Automated test coverage of specs/code	N/A	>90%	>90%	100%	?
7	Number of bugs caught in testing vs field use	N/A	>75%	>75%	>90%	?
8	Change failure rate (rollback deployed code)	<1%	<5%	<10%	<1%	?
9	% code avail to DoD for inspection/rebuild	N/A	100%	100%	100%	?
10	Number/percentage of functions implemented	80%	90%	70%	95%	100%
11	Usage and user satisfaction	TBD	TBD	TBD	TBD	?

[5] Target values are notional; different types of software (SW) as defined in DIB Ten Commandments.

Acronyms defined: Commercial off-the-shelf (COTS), apps is short for applications, specs is short for specifications, hardware/operating system (HW/OS), hardware/software (HW/SW)

12	Complexity metrics	#/type of specs structure of code #/type of platforms		# programmers #/skill level of teams #/type deployments		Partial/ manual tracking
13	Development plan/environment metrics					
14	“Nunn-McCurdy” threshold (for any metric)	1.1X	1.25X	1.5X	1.5X each effort	1.25X Total $

Supporting Information

The information below provides additional details and rationale for the proposed metrics. The different types of software considered in the document are described here in greater depth, followed by comments on the proposed metrics, grouped into four categories: (a) deployment rate metrics, (b) response rate metrics, (c) code quality metrics, and (d) program management, assessment and estimation metrics.

Software Types (from DIB Ten Commandments)

Not all software is alike, and different types of software require different approaches for development, deployment, and life-cycle management. It is important to avoid a “one size fits all” approach to weapons systems. Acquisition practices for hardware are almost never right for software: they are too slow, too expensive, and too focused on enterprise-wide uniformity instead of local customization. Similarly, the process for obtaining software to manage travel is different than what is required to manage the software on an F-35. We suggest a taxonomy with four types of software requiring four different approaches:

- A: commercial (“off-the-shelf”) software with no DoD-specific customization required;
- B: commercial software with DoD-specific customization needed;
- C: custom software running on commodity hardware (in data centers or in the field);
- D: custom software running on custom hardware (e.g., embedded software).

Type A (COTS apps): The first class of software consists of applications that are available from commercial suppliers. Business processes, financial management, human resources, accounting and other “enterprise” applications in DoD are generally not more complicated nor significantly larger in scale than those in the private sector. Unmodified commercial software should be deployed in nearly all circumstances. Where DoD processes are not amenable to this approach, those processes should be modified, not the software.

Type B (Customized SW): The second class of software constitutes those applications that consist of commercially available software that is customized for DoD-specific usage. Customizations can include the use of configuration files, parameter values, or scripted functions that are tailored for DoD missions. These applications will generally require configuration by DoD personnel, contractors, or vendors.

Type C (COTS HW/OS): The third class of software applications is those that are highly specialized for DoD operations but can run on commercial hardware and standard operating systems (e.g., Linux or Windows). These applications will generally be able to take advantage of commercial processes for software development and deployment, including the use of open source code and tools. This class of software includes applications that are written by DoD personnel as well as those that are developed by contractors.

Type D (Custom SW/HW): This class of software focuses on applications involving real-time, mission-critical, embedded software whose design is highly coupled to its customized hardware. Examples include primary avionics or engine control, or target tracking in shipboard radar systems. Requirements such as safety, target discrimination, and fundamental timing considerations demand that extensive formal analysis, test, validation, and verification activities be carried out in virtual and "iron bird" environments before deployment to active systems. These considerations also warrant care in the way application programming interfaces (APIs) are potentially presented to third parties.

Types of Software Metrics

Deployment Rate Metrics

Overview: Consistent with previous Defense Innovation Board (DIB) commentary, and software industry best practices, an organizational mentality that prioritizes speed is the ultimate determinant of success in delivering value to end users. An approach to software development that privileges speed over other factors drives efficient decision-making processes; forces the use of increased automation of development and deployment processes; encourages the use of code that is machine-generated as well as code that is correct-by-construction; relies heavily on automated unit and system level testing; and enables the iterative, deliver-value-now mentality of a modern software environment. Thus we list these metrics first.

#	Metric	Target value (by software type)				Typical DoD values for SW
		COTS apps	Customized SW	COTS HW/OS	Real-time HW/SW	
1	Time from program launch to deployment of simplest useful functionality	<1 mo	<3 mo	<6 mo	<1 yr	3-5 yrs
2	Time to field high priority fcn (spec → ops) or fix newly found security hole (find → ops)	N/A <1 wk	<1 mo <1 wk	<3 mo <1 wk	<3 mo <1 wk	1-5 yrs 1-18 m
3	Time from code committed to code in use	<1 wk	<1 hr	<1 da	<1 mo	1-18 m

Background: These measures capture the rate at which new functions and changes to a software application can be put into operation (in the field):

1. The time from program launch to deployment of the "simplest useful functionality" is an important metric because it determines the first point at which the code can start doing useful work and also at which feedback can be gathered that supports refinement of the features. There is a tendency in DoD to deliver code only once it has met all of the specifications, but this can lead to significant delays in providing useful code to the user. We instead advocate

getting code in the hands of the user quickly, even if it only solves a subset of the full functionality. Something is better than nothing, and user feedback often reveals omissions in the specifications and can refine the initial requirements. As code becomes more customized, this interval of time might extend due to the need to run more complex tests to ensure that all configurations operate as expected, and that complex timing and other safety/mission-critical specifications are satisfied. It is important to note that this metric is not just about coding time. It also measures the time required to process and adjudicate the changes (including release approval), often the most time-consuming part of providing new or upgraded functionality.

2. Once the code is deployed, it is possible to measure the amount of time that it takes to make incremental changes that either implement new functions or fix issues that have been identified. The importance of the functionality or severity of the error will determine how quickly these changes should be made, but it should be possible to deploy high priority code updates much more quickly and in much smaller increments than typical DoD "block" upgrades. A similar measure to the time it takes to deploy code to the field is deployment frequency. Deployment frequency can be on-demand (multiple per day), once per hour, once per day, once per week, etc. Faster deployment frequency often correlates with smaller batch sizes.

3. The time from which code is committed to a repository until it is available for use in the field is referred to as "lead time," and good performance on this metric is a necessary condition for rapid evolution of delivered software functionality. Shorter product delivery times demand faster feedback, which enables tighter coupling to user needs. For commercially available applications, the lead time will be based on vendor deployment processes and may be slower than what is needed for customized code, be it for commercial hardware/operating systems or custom hardware. However, we believe that in the selection of commercial software, emphasis should be given to the vendor's iteration cycles and lead time performance. Embedded code will often require much more extensive testing before it is deployed, and therefore its lead time may be longer.

Response Rate Metrics

Overview: Our philosophy is that delivering a partial solution to the user quickly is almost always better than delivering a complete or perfect solution at the end of a contract, on the first attempt. Consistent with that, mistakes will occur. No software is bug-free, and so it is unrealistic and unnecessary to insist on that, except where certain safety matters are concerned.[6] Code that does most things right will still be useful while a patch is being identified and fielded. How gracefully software fails, how many errors are caught and resolved in testing, and how rapidly developers patch bugs are excellent measures of software development prowess.

[6] The Department and its suppliers (due to the requirements of the contracts to which they are bound) often resort to blanket pronouncements about safety and security, which often lead to applying the most extreme measures even when not needed; this risk-averse approach to treating everything as a grave risk to cyber security or safety has been labeled by the DIB as a "self-denial of service attack." While cybersecurity is clearly critical for software systems, the Department needs to rely on product managers who use judgment to make subtle, nuanced, and risk-based judgments about trade-offs during the software development process.

#	Metric	Target value (by software type)				Typical DoD values for SW
		COTS apps	Customized SW	COTS HW/OS	Real-time HW/SW	
4	Time required for full regression test (automated) and cybersecurity audit/penetration testing[7]	N/A <1 mo	<1 da <1 mo	<1 da <1 mo	<1 wk <3 mo	2 yrs 2 yrs
5	Time required to restore service after outage	<1 hr	<6 hr	<1 day	N/A[8]	?

Background: These two metrics are intended for "generic" software programs with moderate complexity and criticality. Their purpose is to:

4. Measure the ability to conduct more complete functional tests of the full software suite (e.g., regression tests) in a timely fashion, to identify problems in deployed software that can be quickly corrected, and to restore service after an incident such as an unplanned outage or service impairment, occurs (also called "mean time to repair," (MTR)).

5. Track the time required to resolve an interruption to service, including a bad deployment.

Code Quality Metrics

Overview: These metrics are intended to be used as a measure of the quality of the code and to focus on identifying errors in the code, either at the time of development (e.g., via unit tests) or in the field.

#	Metric	Target value (by software type)				Typical DoD values for SW
		COTS apps	Customized SW	COTS HW/OS	Real-time HW/SW	
6	Automated test coverage of specs/code	N/A	>90%	>90%	100%	?
7	Number of bugs caught in testing vs field use	N/A	>75%	>75%	>90%	?
8	Change failure rate (rollback deployed code)	<1%	<5%	<10%	<1%	?
9	% code avail to DoD for inspection/rebuild	N/A	100%	100%	100%	0%

[7] The two different response rate metrics for different types of software reflect the level of complexity of the software, the likely resources available to identify and fix problems, and the level of integration of the hardware and software.

[8] We note that for embedded systems, which must be running at all times and which are updated much less frequently, the notion of "restoring" service is not directly applicable.

Background:

6. Automated developmental tests provide a means of ensuring that updates to the code do not break previous functionality and that new functionality works as expected. Ideally, for each function that is implemented, a set of automated tests will be constructed that cover both the specification for what the performance should achieve as well as the code that is used to implement that function.

7. The percentage of specifications tested by the automated test suite provides rapid confidence that a software change has not caused some specification to fail, as well as confidence that the software does what it is supposed to do. Test coverage of the code is a common metric for software test quality and one that most software development environments can compute automatically (e.g., in a continuous integration (CI) workflow, each commit and/or pull request to a repository would run all the automated developmental tests and compute the percentage covered). For customized software and applications that run on commercial hardware and operating systems, 90% unit test coverage is a good target. Embedded code should strive for 100% coverage (i.e., no "dark" code) since it is often safety- or mission-critical.[9] The focus of these metrics is on developmental tests, as operational testing is important, but expensive, so it is far less expensive to find and fix defects through developmental testing.

8. Developmental tests do not cover every conceivable situation in which an application might be used, so errors will be discovered in the field. The percent of bugs caught in testing (via unit tests or regression tests) versus those caught in the field provide a measure of the both the quality of the code and the thoroughness of the testing environment. Bugs discovered late in the development cycle or after deployment can "cost" an order of magnitude more than early bugs (in terms of time to fix and impact to a program), and a software system that is mature finds many more bugs during testing and few in the field. Late bugs are particularly expensive when fixing those bugs can require hardware changes, and so code running on custom hardware should be tested more strenuously. Bugs should be prioritized by severity and the trends over time for serious bugs should be monitored and used to drive changes in the test environment and software development process.

9. When bugs do occur, it may be necessary to roll back the deployed code and return to an earlier version. Change fail percentage is the percentage of changes to production that fail, including software releases and infrastructure changes. This should include changes that result in degraded service or subsequently require remediation, such as those that lead to service impairment or outage, or require a hotfix, rollback, fix-forward, or patch. For COTS applications, this should occur rarely because the amount of testing done by the vendor, including test deployments to beta users, will typically be very high. There may be a higher change failure rate as the application becomes more customized—because it can be difficult to test for issues where there is a variety of hardware configurations operating in the field, for example—but for embedded code, the change failure rate should be small, due to the more safety-critical nature of that code leading to more emphasis on up-front testing.

Functionality metrics

Overview: These metrics are intended to capture how useful the software program is in terms of delivering value to the field. We envision that a software program will have a number of desired

[9] Safety- or mission-critical software often strives for more rigorous test coverage metrics, such as high branch coverage or in some cases high modified condition/decision coverage (MC/DC).

features that define its functionality. Software should be instrumented so that the use of those features is measured and, when appropriate, users of the software should be monitored or surveyed to determine their use of/satisfaction with the software.

#	Metric	Target value (by software type)				Typical DoD values for SW
		COTS apps	Custom -ized SW	COTS HW/OS	Real-time HW/SW	
10	Number/percentage of functions implemented	80%	90%	70%	95%	100%
11	Usage and user satisfaction	TBD	TBD	TBD	TBD	?

Background:

10. An ongoing software program will have some number of functions that it performs and a list of additional functions that are to be added over time. These new functions could be feature requires from users or desired features generated by the program office that are on the list for consideration to be implemented next. Keeping track of these features and the rate at which they are implemented provides a measure of the delivery of functionality to the user. This specific way in which functionality is measured will be dependent on the type of software being developed.

11. For software that is used by a person, the ultimate metric is whether the software is helping that person get useful work done. Keeping track of the usage of the software (and different parts of the software) can be done by instrumenting the code and keeping track of the data it generates. To determine whether or not the software is providing good value to the person who is using it, surveying the user may be the most direct mechanism (similar to rating software that you use on a computer or smart phone).

Program Management, Assessment, and Estimation Metrics

Overview: The final set of metrics are intended for management of software programs, including cost assessment and performance estimation. These metrics describe a list of "features" (performance metrics, contract terms, project plans, activity descriptions) that should be required as part of future software projects to provide better tools for monitoring and predicting time, cost, and quality. In its public deliberations regarding software acquisition and practices, the DIB has described how metrics of this type might be used to estimate the cost, schedule, and performance of software programs.

#	Metric	Target value (by software type)				Typical DoD values for SW
		COTS apps	Custom ized SW	COTS HW/OS	Real-time HW/SW	
12	Complexity metrics	#/type of specs # programmers structure of code #/skill level of teams #/type of platforms #/type deployments				Partial/ manual tracking
13	Development plan/environment metrics					

14	“Nunn-McCurdy” threshold (for any metric)	1.1X	1.25X	1.5X	1.5X each effort	1.25X Total $

Background:

12. Structure of specifications, code, and development and execution platforms**.**

    To measure the complexity of a software program, and therefore assess the cost, schedule and performance of that program, a number of features must be measured that capture the underlying “structure” of the application. The use of the term “structure” is intentionally flexible, but generally includes properties such as size, type, and layering. Examples of features that can be captured that related to underlying complexity include:

    - *Structure of specifications:* Modern specification environments (e.g., application life-cycle management [ALM] tools) provide structured ways of representing specifications, from program level requirements to derived specifications for sub-systems, or individual teams. The structure represented in these tools can be used as a measure of the difficulty of the application that is being designed.

    - *Level and type of user engagement during application development:* How much time do developers spend with users, especially early in the program? How many developers are “on site” (in the same organization and/or geographic location as the end user)?

    - *Structure of the code base (software architecture):* Modern software development environments allow structured partitioning of the code into functions, libraries/frameworks, and services. The structure of this partitioning (number of modules, number of layers, and amount of coupling between modules and layers) can provide a measure of the complexity of the underlying code.

    - *The amount of reuse of existing code, including open source code:* In many situations there are well-maintained code bases that can be used to quickly create and scale applications without rewriting software from scratch. These libraries and code frameworks are particularly useful when using commodity hardware and operating systems, since the packages will often be maintained and expanded by others, leveraging external effort.

    - *Structure of the development platform/environment:* This includes the software development environments that are being used, the types of programming methodologies (e.g., XP, agile, waterfall, spiral) that are employed, and the level of maturity of the programming organization (ISO, CMMI, SPICE).

    - *Structure of the execution platform/environment:* The execution environment can have an impact on the ability to emulate the execution environment within the development environment, as well as the portability of applications between different execution environments. Possible platforms include various cloud computing environments as well as platform-as-a-service (PaaS) environments that support multiple cloud computing vendors.

13. Structure and type of development and operational environment.

To predict and monitor the level of effort required to implement and run a software application, measurement of the development and operation environments is critical. These measurements include the structure of those environments (e.g., waterfall versus spiral versus agile, use of continuous integration tools, integrated tools for issue tracking/resolution, code review mechanisms), the tempo of development and delivery, and use of the functionality provided by the application. Example of features that can be captured that relate to the structure and type of development and operation use:

- Number and skill level of programmers on the development team
- Number of development platforms used across the project
- Number of subcontractors or outside vendors used for application components
- Number and type of user operating environments (execution platforms) supported
- Rate at which major functions (included in specifications) are delivered and updated
- Rate at which the operational environment must be updated (e.g., hardware refresh rate)
- Rate at which the mission environment changes (driving changes to the code)
- Number (seats or sites) and skill level of the users of the software

14. Tracking software program progress

To properly manage the continuous development and deployment of software, DoD should be able to track the metrics above with minimal additional effort from the programmers because this information should be gathered and transmitted automatically through the development, deployment, and execution environments, using automated tools. Some examples of the types of metrics that are readily available include commit activity data (number and rate of commits), team size, number of commenters (on pull requests), number of pull request mergers, average and standard deviation of the months in which there was development activity, average and standard deviation of the number of commits per month.

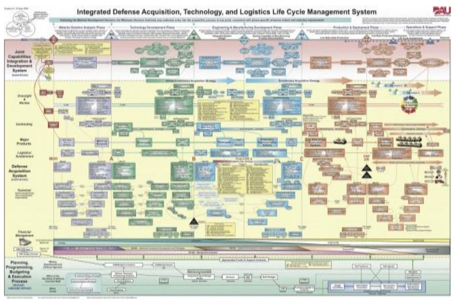
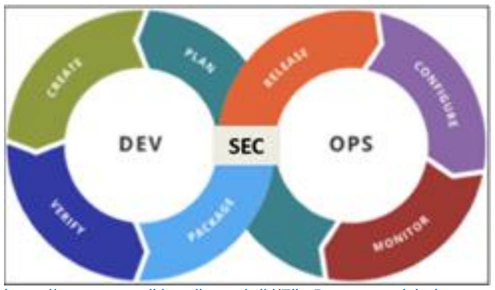
Thresholds should be established to determine when management attention is required, but also when a program is so far off its initial plan that it should be re-evaluated. Today’s “Nunn-McCurdys” or “Critical Changes” refer to breaches in cost or schedule thresholds. The current 25% unit cost growth and 50% total program cost growth thresholds often will not make sense for continuously developed software programs.

An alternative to cost-based thresholds is to establish thresholds based on the metrics listed above, with different thresholds for different types of software. A notion of a “Nunn-McCurdy type breach” for software programs based on some of the above performance metrics recorded at lower levels of effort or on specific applications could serve as better means of identifying major issues earlier in a program. Commercially available software, with or without customization, should be the easiest type for which to establish accurate metrics, since it already exists and should be straightforward to purchase and deploy. Metrics for customized software running on either commercial or DoD-specific hardware is likely to be more difficult to predict, so a higher threshold can be used in those circumstances.

Defense Innovation Board Do's and Don'ts for Software

This document provides a summary of the Defense Innovation Board's (DIB's) observations on software practices in the DoD and a set of recommendations for a more modern set of acquisition and development principles. These recommendations build on the DIB's "Ten Commandments of Software." In addition, we indicate some of the specific statutory, regulatory, and policy obstacles to implementing modern software practices that need to be changed.

Executive Summary

Observed practice (Don'ts)	**Desired state (Do's)**	**Obstacles**
 Defense Acquisition University, June 2010	 CREATE PLAN RELEASE CONFIGURE DEV SEC OPS VERIFY PACKAGE MONITOR https://commons.wikimedia.org/wiki/File:Devops-toolchain.svg (modifications licensed CC-BY-SA)	10 U.S.C. §2334 10 U.S.C. §2399 10 U.S.C. §2430 10 U.S.C §2433a 10 U.S.C. §2460 10 U.S.C. §2464 DODI 5000.02, par 5.c.(2) and 5.c.(3)(c)-(d)
Spend 2 years on excessively detailed requirements development	Require developers to meet with end users, then start small and iterate to quickly deliver useful code	DODI 5000.02, par 5.c.(2) CJCSI 3170.01I App A.1.b
Define success as 100% compliance with requirements	Accept 70% solutions[10] in a short time (months) and add functionality in rapid iterations (weeks)	10 U.S.C. §2399 OMB Cir A-11 pp 42-43
Require OT&E to certify compliance after development and before approval to deploy	Create automated test environments to enable continuous (and secure) integration and deployment to shift testing left	10 U.S.C. §139b/d 10 U.S.C. §2399 Cultural
Apply hardware life-cycle management processes to software	Take advantage of the fact that software is essentially free to duplicate, distribute, and modify	10 U.S.C. §2334 10 U.S.C. §2399 10 U.S.C. §2430 48 CFR 207.106 DODI 5000.02
Require customized software solutions to match DoD practices	For common functions, purchase existing software and change DoD processes to use existing apps	Culture

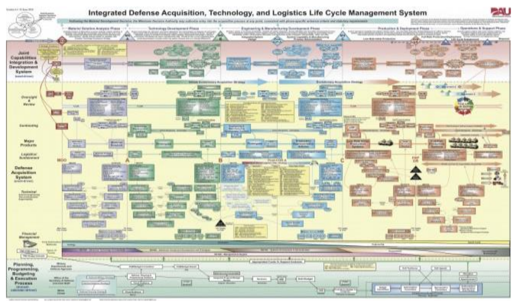
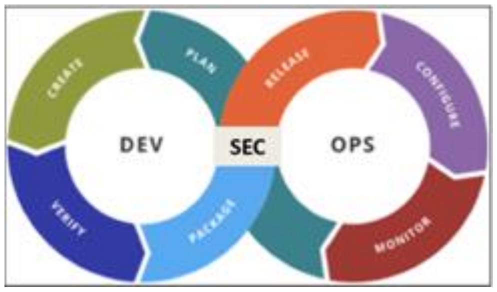
[10] 70% is notional. The point is to deliver the simplest, most useful functionality to the warfighter quickly.

Use legacy languages and operating systems that are hard to support and insecure	Use modern software languages and operating systems (with all patches up-to-date)	10 U.S.C. §2334 DoDI 5000.02, Enclosure 11 Culture
Evaluate cyber security after the systems have been completed, separately from OT&E	Use validated software development platforms that permit continuous integration & evaluation (DevSecOps)	DOT&E Memos Culture
Consider development and sustainment of software as entirely separate phases of acquisition	Treat software development as a continuous activity, adding functionality across its life cycle	10 U.S.C. §2399 10 U.S.C. §2430 10 U.S.C. §2460 10 U.S.C. §2464 DODI 5000.02, par 5.c.(2) and 5.c.(3)(c)-(d)
Depend almost entirely on outside vendors for all product development and sustainment	Require source code as a deliverable on all purpose-built DoD software contracts. Continuous development and integration, rather than sustainment, should be a part of all contracts. DoD personnel should be trained to extend the software through source code or API access[11]	Culture (no apparent statutory obstacle) FAR/DFARS technical data rights
Turn documents like this into a process and enforce compliance	Hire competent people with appropriate expertise in software to implement the desired state and give them the freedom to do so ("competence trumps process")	Culture

[11] As noted in the DIB's 10 Commandments of Software

Supporting Information

The information below, broken out by entry in the executive summary table (see table E.8 above), provides additional information and a rationale for each desired state.

Don't	Do
 Defense Acquisition University, June 2010	 CREATE PLAN RELEASE CONFIGURE DEV SEC OPS VERIFY PACKAGE MONITOR https://commons.wikimedia.org/wiki/File:Devops-toolchain.svg

The DoD 5000 process, depicted on the left in figure E.1, provides a detailed DoD process for setting requirements for complex systems and ensuring that delivered systems are compliant with those requirements. The DoD's "one size fits all" approach to acquisition has attempted to apply this model to software systems, where it is wholly inappropriate. Software is different than hardware. Modern software methods make use of a much more iterative process, often referred to as "DevOps," in which development and deployment (operations) are a continuous process, as depicted on the right. A key aspect of DevOps is continuous delivery of improved functionality through interaction with the end user.

Why this is hard to do, but also worth doing:[12]

- DoD 5000 is designed to give OSD, the Services, and Congress some level of visibility and oversight into the development, acquisition, and sustainment of large weapons systems. While this directive may be useful for weapons systems with multi-billion dollar unit costs, it does not make sense for most software systems.
- While having one consistent procurement process is desirable in many cases, the cost of using that same process on software is that software is delivered late to need, costs substantially more than the proposed estimates, and cannot easily be continuously updated and optimized.
- Moving to a software development approach will enable the DoD to move from a *specify, develop, acquire, sustain* mentality to a more modern (and more useful) *create, scale, optimize* (DevOps/DevSecOps) mentality. Enabling rapid iteration will create a system in which the United States can update software at least as fast as our adversaries can change tactics, allowing us to get inside their OODA loop.

[12] These comments and the similar ones that follow for other area were obtained by soliciting feedback on this document from people familiar with government acquisition processes and modern software development environments.

Acronyms defined: Office of the Secretary of Defense (OSD), OODA is short for the decision cycle of Observe, Orient, Decide, and Act.

Don’t	Do
Spend 2 years on excessively detailed requirements development	Require developers to meet with end users, then start small and iterate to quickly deliver useful code
Define success as 100% compliance to requirements	Accept 70% solutions in a short time (months) and add functionality in rapid iterations (weeks)

Developing major weapons systems is costly and time consuming, so it is important that the delivered system meets the needs of the user. The DoD attempts to meet these needs with a lengthy process in which a series of requirements are established, and a successful program is one that meets those requirements (ideally close to the program’s cost and schedule estimates). Software, however, is different. When done right, it is easy to quickly deploy new software that improves functionality and, when necessary, rapidly rollback deployed code. It is more useful to get something simple working quickly (time-constrained execution) and then exploit the ability to iterate rapidly in order to get the remaining desired functionality (which will often change in any case, either in response to user needs or adversarial tactics).

Why this is hard to do, but also why it is worth doing:

- Global deployment of software on systems which are not always network-connected (e.g., an aircraft carrier or submarine underway) introduces very real problems around version management, training, and wisely managing changes to mission-critical systems.
- In the world of non-military, consumer Internet applications, it is easy to glibly talk about continuous deployment and delivery. In these environments, it is easy to execute and the consequences for messing up (such as making something incredibly confusing or hard to find) are minor. The same is not always true for DoD systems—and DoD software projects rarely offer scalable and applicable solutions to address the need for continuous development.
- Creating an approach (and the supporting platforms) that enables the DoD to achieve continuous deployment is a non-trivial task and will have different challenges than the process for a consumer Internet application. The DoD must lay out strategies for mitigating these challenges. Fortunately, there are tools that can be build upon: many solutions have already been developed in consumer industries that require failsafe applications with security complexities.
- Continuous deployment depends on the entire ecosystem, not just the front-end software development.
- Make sure to focus on product design and *product* management, which prioritizes delivery of capability to meet the changing needs of users, rather than program/project management, which focus on execution against a pre-approved plan. This shift is key to user engagement, research, and design.

Don’t	Do
Require OT&E to certify compliance after development and before approval to deploy	Create automated test environments to enable continuous (and secure) integration and deployment to shift testing left
Evaluate cyber security after the system has been completed, separately from OT&E	Use validated software development platforms that permit continuous integration and evaluation

Why this is hard to do, but also worth doing:

- The DoD typically performs a cyber evaluation on software only after delivery of the initial product. Modern software approaches have not always explicitly addressed cyber security (though this is changing with “DevSecOps”). This omission has given DoD decision-makers an easy “out” for dismissing recommendations (or setting up roadblocks) for DevOps strategies like continuous deployment. Cyber security concerns must be addressed head on, and in a manner that demonstrates better security in realistic circumstances. Until then, change is unlikely.

- More dynamic approaches to address the cyber security concerns must be developed and implemented through some amount of logic and a fair bit of data. Case studies of red teaming also help: *Hack the Pentagon* should be able to provide some true examples that generate concern. It may be necessary to obtain access to some additional good data that goes beyond what corporations are willing to share publicly.

- To succeed, it will be important not to assume that it will be clear how these recommendations solve for all cyber security concerns. Recommendations should make explicit statements about what can be accomplished, taking away the reasons to say “no.”

Don’t	Do
Apply hardware life-cycle management processes to software	Take advantage of the fact that software is essentially free to duplicate, distribute, and modify
Consider development and sustainment of software as entirely separate phases of acquisition	Treat software development as a continuous activity, adding functionality across its life cycle

Why this is hard to do, but also worth doing:

- Program of record funding is specifically broken out into development and sustainment. These distinct categories of appropriations lead program managers and acquisition professionals to the conclusion that new functionality can only be added within development contracts and that money allocated for sustainment cannot be used to add new features. Vendor evaluation for development and sustainment contracts are different; vendors on sustainment contracts often do not have the same development competencies and frequently are not the people who built the original system. To create an environment that will support a DevOps/DevSecOps approach, DoD Commands and Services should jointly own the development and maintenance of software with contractors who provide more specialized capabilities. Contracts for software should focus on developing and deploying software (to operations) over the long term, rather than the typical, sequential approach - "acquiring" software followed by "sustaining" that software.

Don't	**Do**
Require customized software solutions to match DoD practices	For common functions, purchase existing software and change DoD processes to use existing apps

Business processes, financial, human resources, accounting and other "enterprise" applications in the DoD are generally not more complicated nor significantly larger in scale than those in the private sector. Commercial software, unmodified, should be deployed in nearly all circumstances. Where DoD processes are not amenable to this approach, those processes should be modified, not the software. Doing so allows the DoD to take advantage of the much larger commercial base for common functions (e.g., Concur has 25M active users for its travel software).

Don't	**Do**
Use legacy languages and operating systems that are hard to support and insecure	Use modern software languages and operating systems (with all patches up-to-date)

Modern programming languages and software development environments have been optimized to help eliminate bugs and security vulnerabilities that were often left to programmers to avoid (an almost impossible endeavor). Additionally, outdated operating systems are a major security vulnerability and the DoD should assume that any computer running such a system will eventually be compromised.[13] Standard practice in industry is to apply security patches within 48 hours of release, though even this is probably too big a window for defense systems. Treat software vulnerabilities like perimeter defense vulnerabilities: if there is a hole in your perimeter and people are getting in, you need to patch the hole quickly and effectively.

Why this is hard to do, but also worth doing:

- DoD looks at the cost of upgrading hardware as a major cost that is tied to "modernization." But hardware should be thought of as a consumable like any other, such as fuel and parts, that must be continually replaced for a weapon system to maintain operational capability. This change would require DoD to provide a stable annual budget that paid for new hardware and software capability.
- The advantage of using modern hardware and operating systems on DoD systems are manifold: better security, better functionality, reduced (unit) costs, and lower overall maintenance costs.

[13] See the DIB 10 Commandments of Software supporting thoughts and recommendations. "*Move to a model of continuous hardware refresh in which computers are treated as a consumable with a 2-3 year lifetime.*"

Don’t	Do
Turn documents like this into a process and enforce compliance	Hire competent people with appropriate expertise in software to implement the desire state and give them the freedom to do so (“competence trumps process”)

Why this is hard to do, but also why it is worth considering doing it:

- Good engineers want to build things, not just write and evaluate contracts. If their jobs are mainly contracting or monitoring, their software skills will quickly become outdated. This can be solved in the short term by a rotational program: do not allow programmers to stay in contracting for more than 4 years, so their technical capabilities are current.
- The government must team with commercial companies to ensure that it has access to the collection of talent required to develop modern software systems, as well as develop internal talent. The DoD should increase its use of contractors whose aim is not just to provide software, but to increase the software development capabilities and competency of the department. By making use of enlisted personnel, reservists, contractors, and other resources, it is possible to create and maintain highly effective teams who contribute to national security through software development.

Additional Obstacles

In addition to the specific obstacles listed above, we capture here a collection of statutes, regulations, processes and cultural norms that are impediments to implementing a modern set of software acquisition and development principles.

Statutes

The statutes below provide examples of impediments to the implementation of modern software development practices in DoD systems.

Acquisition strategy (10 U.S.C §2431a): 2431a(d) establishes the review process for major defense acquisition programs and is written around the framework of waterfall development for long timescale, hardware-centric programs. In particular, this statute establishes decision-gates at Milestone A (entry into technology maturation and risk reduction), Milestone B (entry into system development and demonstration), and entry into full-rate production. For many software programs this set of terms and approach does not make sense and is incompatible with the ability to deliver capability to the field in a rapid fashion.

Critical cost growth in major defense acquisition programs (10 U.S.C. §2433a [Nunn-McCurdy]): 2433 establishes the conditions under which Congress reviews a major program that has undergone critical cost growth and determines with it should continue. By the time a software program hits a Nunn-McCurdy breach it has already gone well past the point where the program should have been terminated and restarted using a different approach. All software procurement programs should start small, be iterative, and build on success – or be terminated quickly.

Independent cost estimation and cost analysis (10 U.S.C. §2334)

Working capital funds (10 U.S.C. §2208(r)):
- 2+ year lead times from plan to budget does not allow for continuous engineering
- Differentiating software development workload as Research, Development, Test and Engineering (RDT&E), Procurement, or Operations and Maintenance (O&M) is meaningless as there should be no final fielding or sustainment element to continuous engineering.
- System-defined program elements hinder the ability to deliver holistic capabilities and enable real-time resource, requirements, performance and schedule trades across systems without significant work.

Operational Test and Evaluation (10 U.S.C. §139b/d, 10 U.S.C. §2399): 139 establishes the position of the Director of Operational Test and Evaluation (DOT&E) and requires that person to carry out field tests, under realistic combat conditions, of weapon systems for the purpose of determining the effectiveness and suitability of those systems in combat by typical military users. 2399(a) states that a major defense acquisition program "may not proceed beyond low-rate initial production until initial operational test and evaluation of the program, subprogram, or element is completed." 2399(b)(4) further states that the program many not proceed "until the Director [of Operational Test and Evaluation] has submitted to the Secretary of Defense the report with respect to that program under paragraph (2) and the congressional defense committees have received that report." These are obstacles for DevSecOps implementation of software, where changes should be deployed to the field quickly as part of the (continuous) development process. They are an example of a "tailgate" process for OT&E that impedes our ability to deploy software quickly and drives a set of processes in which OT&E impedes rather than enhances the software development process. Instead of this process, Congress should allow independent OT&E of software to occur in parallel with deployment and also require that OT&E cycles for software match development cycles through the use of automated workflows and test harnesses wherever possible.

Additional issues:
- Testing and evaluation (T&E) must be integrated into the development life cycle to facilitate DevSecOps, and reduce operations and sustainment (O&S) costs. T&E should be present from requirements setting to O&S
- Programs need persistent and realistic environments that permit continuous, agile testing of all systems (embedded, networked, etc.) in a representative SoS environment
- Software environments should be part of the contract deliverables and accessible to T&E, including source code, build tools, test scripts, data

Definition of a major acquisition program (10 U.S.C. §2430): The designation of a program as a major acquisition program triggers a set of procedures that are designed for acquisition of hardware. This includes triggering of the DoD Instruction 5000.02, which is currently tuned for hardware systems. An alternative instruction, DoD Instruction 5000.75, is better tuned for software, but can only used for defense *business* systems; it is not valid for "weapons systems."

Depot level maintenance and repair; core logistics (10 U.S.C. §2460, 10 U.S.C. §2464): The definitions of maintenance, repair, and logistics are based on an acquisition model that is appropriate for hardware but not well aligned with the operation of modern software. For example, §2464 says that Services will "maintain and repair the weapon systems." But software is not maintained, it is *optimized* (with better performance and new functionality) on a *continuous* basis. §2460(b)(1) further states that depot level maintenance and repair "does not include the procurement of major modifications or upgrades of weapon systems that are designed to improve program performance."

Additional issues:

- DoD's challenge in shifting from applying a Hardware (HW) maintenance mindset to Software (SW) hinders DoD's ability to better leverage DoD's organic SW engineering infrastructure to deliver greater capability to the warfighter.
- DoD's acquisition process is not emphasizing an upfront focus on design for software sustainment and a seamless transition to organic software engineering sustainment to reduce the life-cycle cost of software and to speed delivery of capability over the life cycle. Such upfront emphasis is critical given the scope, complexity, and mix of the growing software sustainment demand, in the face of persistent affordability concerns.
- DoD's organic software engineering capabilities and infrastructure are critical to national security, but there is limited enterprise visibility of this infrastructure, its capabilities, workload, and resources to leverage it at the enterprise level to deliver greater capability more affordably to the warfighter.

Regulations

The regulations are the mechanism by which the DoD implements the statutes that govern its operations. They provide additional examples of impediments to the implementation of modern software development practices in DoD systems.

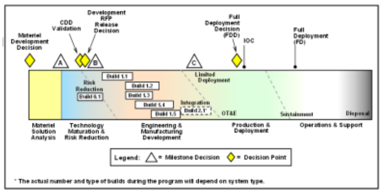
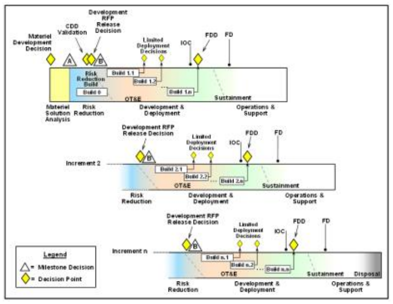
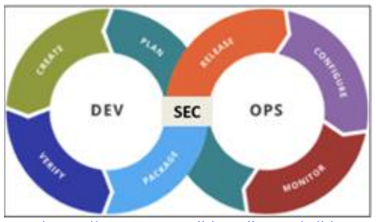
Cost estimating system requirements (48 CFR 252.215-7002) : These regulations set out the expectations for estimation of costs of a program against a set of system requirements. While perhaps appropriate for a hardware-oriented system, they do not take into account the type of continuous development cycle that is required to implement modern software.

Additional requirements for major systems (48 CFR 207.106): These regulations set out procedures for competition of contracts and are written in a manner that separates out the initial deployment of a system with the operation and sustainment of that system. This doesn't make sense for software.

Processes (Instructions)

The detailed processes used to implement the regulations are laid out in Department of Defense Instructions. We illustrate here some of the specific instructions that are obstacles to implementation of modern software development practices.

Major acquisition program development process (DODI 5000.02, par 5.c.(2) and 5.c.(3)(c)-(d)): These portions of the DoD Instructions apply to “Defense Unique Software Intensive” programs and “Incrementally Deployed Software Intensive” programs. While well-intentioned, they are still waterfall processes with years between the cycles of deployments (instead of weeks). These processes may be appropriate for some embedded systems, but are not the right approach for DoD-specific software running on commercial hardware and operating systems, as the diagrams below illustrate:

Definitely not this:	**Better, but still not right:**	**What we need:**
Specify, design, deploy, sustain  DODI 5000.02, Figure 4. Model 2: Defense Unique Software Intensive Program	 DODI 5000.02, Figure 5. Model 3: Incrementally Deployed Software Intensive Program	Implement, scale, optimize  https://commons.wikimedia.org/wiki/File:Devops-toolchain.svg (modifications licensed CC-BY-SA)
Waterfall development	Waterfall development with overlapping builds	Continuous integration and deployment (DevSecOps)

Requirements for programs containing information technology (DoDI 5000.02, Enclosure 11): This enclosure attempts to define the requirements for ensuring information security. It is written under the assumption that the standard waterfall process is being used.

Preparation, Submission, and Execution of the Budget - Acceptance (OMB Cir A-11, II.10): This document is the primary document that instructs agencies how to prepare and submit budget requests for OMB review and approval. Section II.10 describes the conditions for acceptance of an acquired item by the government, and requires that the asset meets the requirements of the contract. The impact of this procedure is that it establishes a “100% compliance” mentality in order for the government to accept a software “asset.”

Culture

In this final section we catalog a list of culture items that do not necessarily require changes in statutes, regulations, or instructions, but rather a change in the way that DoD personnel interpret implement their processes. Changing the culture of DoD is a complex process, depending in large part on incentivizing the behaviors that will lead to the desired state.

Data and metrics

- Multiple, competing, and sometimes conflicting types of data and metrics used, or not used, for assessing software in DOD
- Inability to collect meaningful data about software development and performance in a low cost manner, at scale
- Inability to turn data into meaningful analysis and inability to implement decisions or changes to software activities (L/R/C)

Contracts

- Individual contracts are subject to review processes designed for large programs (of which they are likely enabling). This limits the agility of individual contract actions, even when modular contracting approaches are applied. In addition, the acquisition process is rigid and revolves around templates, boards, and checklists thus limiting the ability for innovation and streamlining execution.
- Contracts focus on technical requirements instead of contractual process requirements. The contract should address overall scope, PoP, and price. The technical execution requirements should be separate and managed by the product owner or other technical lead.
- Intellectual Property (IP) rights are often generically incorporated without considering the layers of technology often applied to a solution. A single solution might include open source, proprietary SW, and government custom code. The IP clauses should reflect all of the technology that is used.

Security Accreditation

- Although developing and operating software securely is a primary concern, the means to achieve and demonstrate security is overly complex and hampered by inconsistent and outdated/misapplied policy and implementation practices (e.g., overlaying historical DoD Information Assurance Certification and Accreditation Process (DIACAP) over risk management framework (RMF) controls for individual pieces of software versus system accreditation). The sense is that the certification and accreditation process is primarily a “check- the-box” documentary process, adds little value to the overall security of the system, and is likely to overlook flaws in the design, implementation, and the environment in which the software operates.
- The DoD needs to be able to calculate the true and component costs for implementing the RMF and certification and accreditation (C&A) in order to identify inefficiencies, duplicative capabilities, and redundant or overlapping security products and services that are being acquired or developed. Absent a set of metrics it is difficult to prioritize risk areas, investments, and evaluating risk reduction and return on investment.
- The DoD needs to ensure that each Joint Capability Area (JCA) flow-down its strategy, best practices, and implementation requirements/guidance for security and accreditation to allow the Component responsible for implementing the software to appropriately tailor RMF and plan the development, accreditation, and operation of the software.
- The DoD needs to provide automated tools and services needed to integrate continuous monitoring with the development life cycle, enable continuous assessment and accreditation, and delegate decision making at the lowest level possible. The DoD should embrace DevSecOps (not just DevOps) and provide policy supported processes, certified libraries, tools, and a toolchain reference implementation to produce “born secure” software

Testing and Evaluation

- The DoD lacks the realistic test environments needed to support test at the pace of modern software methods.
- The DoD lacks the modern software intellectual property (IP) regime needed to support test and evaluation at the pace of modern software methods
- The DoD lack the enterprise knowledge management/data analytics capability needed to support evaluation of test data at the pace of modern software methods

Workforce

- No defined requirements for software developers
- Antiquated policies (talent management, software development)
- Culture and knowledge (DoD, societal, defense contractors)

Appropriations/Funding

- 2+ year lead times from plan to budget does not allow for continuous engineering
- Differentiating software development workload as Research, Development, Test and Engineering (RDT&E), Procurement, or Operations and Maintenance (O&M) is meaningless as there should be no final fielding or sustainment element to continuous engineering.
- System defined program elements hinder the ability to deliver holistic capabilities and enable real-time resource, requirements, performance and schedule trades across systems without significant work.

Infrastructure

- Creating software: The DoD lacks availability of vetted, secure, reusable components, either as source code, or other digital artifacts (think hardened Docker containers or virtual machines (VMs) here). A repository of discoverable, well indexed, vetted, secure, and reusable components could go a long way. This also emphasizes the point that an awful lot of software now-a-days is software by construction with minimal “glue” code applied.
- Building/managing/testing software: There is a general lack of available tools to build software, especially automated tools (testing/scanning/fuzzing etc.) integrated into a secure pipeline supporting rapid agile development. There is also a significant need to have a common, government owned and managed code repository that all programs could/should/must use (e.g., government-furnished GitHub).
- Running/hosting software: The DoD needs to continually push the level of abstraction up as much as possible for programs. Traditionally programs, even cloud-based solutions, tend to start at Infrastructure as a Service (IaaS) and build their own rest of the stack. We need secure and available Platform as a Service (PaaS) and Function as a Service (FaaS) so that programs only need to focus on core business logic and not on securing a database or message bus over and over again.
- Operating/updating securely: Once developed and instantiated on a secure and available platform, we need to continually monitor, red team (automated?), and evolve the software. This requires proper instrumentation, logging, and monitoring of the platform, supporting libraries/components, and the core program code. A standard/common way to provide instrumentation and monitoring of the running services built into the infrastructure would be very helpful.

Requirements

- A byproduct of top-level requirement flow down is rigidity and over specificity at the derived requirements level that greatly hinders agile s/w design.
- Too often exquisite requirements are levied on a system that in turn drive extensive complex software requirements and design, affecting development, integration, and system test.
- Data sets are siloed within programs: a common “law of requirements” is that programs of record try to avoid dependencies with other programs of record. This is problematic for software-based capabilities because data is often siloed within single programs of record. We have network programs to “pass” data, but the promise of artificial intelligence (AI), including machine learning (ML), is that software algorithms can leverage pools of data from disparate sources (data lakes).
- By tying software to a program of record, it becomes harder to transfer that code across systems and data environments. As a result, DoD limits code reuse within and across Services.

Modernization and sustainment

- DoD’s challenge in shifting from applying a hardware maintenance mindset to software hinders DoD’s ability to better leverage DoD’s organic software engineering infrastructure to deliver greater capability to the warfighter.
- DoD’s acquisition process is not emphasizing an upfront focus on design for software sustainment and a seamless transition to organic software engineering sustainment to reduce the life-cycle cost of software and to speed delivery of capability over the life cycle. Such upfront emphasis is critical given the scope, complexity, and mix of the growing software sustainment demand, in the face of persistent affordability concerns.
- DoD’s organic software engineering capabilities and infrastructure are critical to national security, but there is limited enterprise visibility of this infrastructure, its capabilities, workload, and resources to leverage it at the enterprise level to deliver greater capability more affordably to the warfighter.

Acquisition Strategy

- Acquisition policy framework: Create a cohesive acquisition policy architecture within which effective, efficient software acquisition policy has a home.
- Acquisition management and governance: Flip the concept of an oversight model on its head.

DIB Guide: Detecting Agile BS

Agile is a buzzword of software development, and so all DoD software development projects are, almost by default, now declared to be "agile." The purpose of this document is to provide guidance to DoD program executives and acquisition professionals on how to detect software projects that are really using agile development versus those that are simply waterfall or spiral development in agile clothing ("agile-scrum-fall").

Principles, Values, and Tools

Experts and devotees profess certain key "values" to characterize the culture and approach of agile development. In its work, the DIB has developed its own guiding maxims that roughly map to these true agile values:

Agile value	**DIB maxim**
Individuals and interactions over processes and tools	"Competence trumps process"
Working software over comprehensive documentation	"Minimize time from program launch to deployment of simplest useful functionality"
Customer collaboration over contract negotiation	"Adopt a DevSecOps culture for software systems"
Responding to change over following a plan	"Software programs should start small, be iterative, and build on success – or be terminated quickly"

Key flags that a project is not really agile:

- Nobody on the software development team is talking with and observing the users of the software in action; we mean the *actual* users of the *actual* code.[14] (The PEO does not count as an actual user, nor does the commanding officer, unless she uses the code.)
- Continuous feedback from *users* to the development team (bug reports, users assessments) is not available. Talking once at the beginning of a program to verify requirements doesn't count!
- Meeting requirements is treated as more important than getting something useful into the field as quickly as possible.
- Stakeholders (dev, test, ops, security, contracting, contractors, end-users, etc.)[15] are acting more-or-less autonomously (e.g., 'it's not my job.')
- End users of the software are missing-in-action throughout development; at a minimum they should be present during Release Planning and User Acceptance Testing.

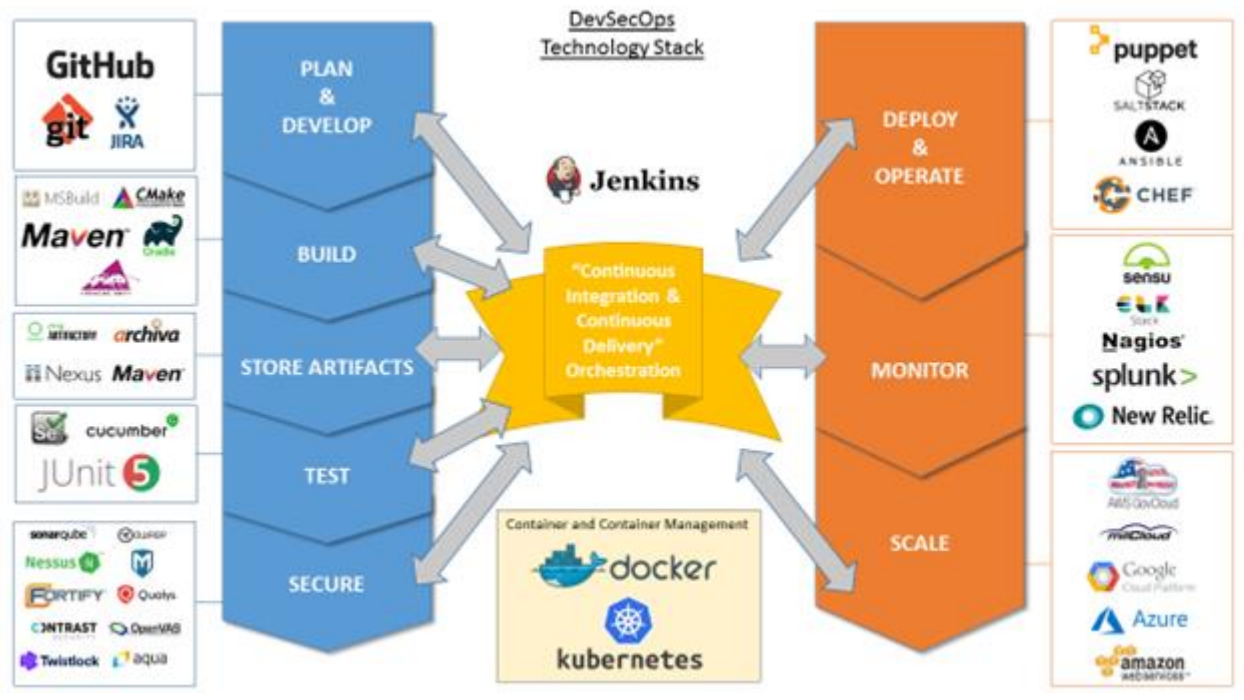
[14] Acceptable substitutes for talking to users: Observing users working, putting prototypes in front of them for feedback, and other aspects of user research that involve less talking.
[15] Dev is short for development, ops is short for operations

- DevSecOps culture is lacking if manual processes are tolerated when such processes can and should be automated (e.g., automated testing, continuous integration, continuous delivery).

Some current, common tools in use by teams using agile development (these will change as better tools become available):[16]

- Git, ClearCase, or Subversion - version control system for tracking changes to source code. Git is the *de facto* open source standard for modern software development.
- BitBucket or GitHub - Repository hosting sites. Also provide issues tracking, continuous integration "apps" and other productivity tools. Widely used by the open source community.
- Jenkins, Circle CI or Travis CI - continuous integration service used to build and test BitBucket and GitHub software projects
- Chef, Ansible, or Puppet - software for writing system configuration "recipes" and streamlining the task of configuring and maintaining a collection of servers
- Docker - computer program that performs operating-system-level virtualization, also known as "containerization"
- Kubernetes or Docker Swarm for Container orchestration
- Jira or Pivotal Tracker - issues reporting, tracking, and management

Graphical version:



Questions to Ask Programming Teams

[16] Tools listed/shown here are for illustration only: no endorsement implied.

- How do you test your code? (Wrong answers: “we have a testing organization,” “OT&E is responsible for testing”)
    - Advanced version: what tool suite are you using for unit tests, regression testing, functional tests, security scans, and deployment certification?
- How automated are your development, testing, security, and deployment pipelines?
    - Advanced version: what tool suite are you using for continuous integration (CI), continuous deployment (CD), regression testing, program documentation; is your infrastructure defined by code?
- Who are your users and how are you interacting with them?
    - Advanced version: what mechanisms are you using to get direct feedback from your users? What tool suite are you using for issue reporting and tracking? How do you allocate issues to programming teams? How to you inform users that their issues are being addressed and/or have been resolved?
- What is your (current and future) cycle time for releases to your users?
    - Advanced version: what software platforms to you support? Are you using containers? What configuration management tools do you use?

Questions for Program Management

- How many programmers are part of the organizations that owns the budget and milestones for the program? (Wrong answers: “we don’t know,” “zero,” “it depends on how you define a programmer”)
- What are your management metrics for development and operations; how are they used to inform priorities, detect problems; how often are they accessed and used by leadership?
- What have you learned in your past three sprint cycles and what did you do about it? (Wrong answers: “what’s a sprint cycle?,” “we are waiting to get approval from management”)
- Who are the users that you deliver value to each sprint cycle? Can we talk to them? (Wrong answers: “we don’t directly deploy our code to users”)

Questions for Customers and Users

- How do you communicate with the developers? Did they observe your relevant teams working and ask questions that indicated a deep understanding of your needs? When is the last time they sat with you and talked about features you would like to see implemented?
- How do you send in suggestions for new features or report issues or bugs in the code? What type of feedback do you get to your requests/reports? Are you ever asked to try prototypes of new software features and observed using them?
- What is the time it takes for a requested feature to show up in the application?

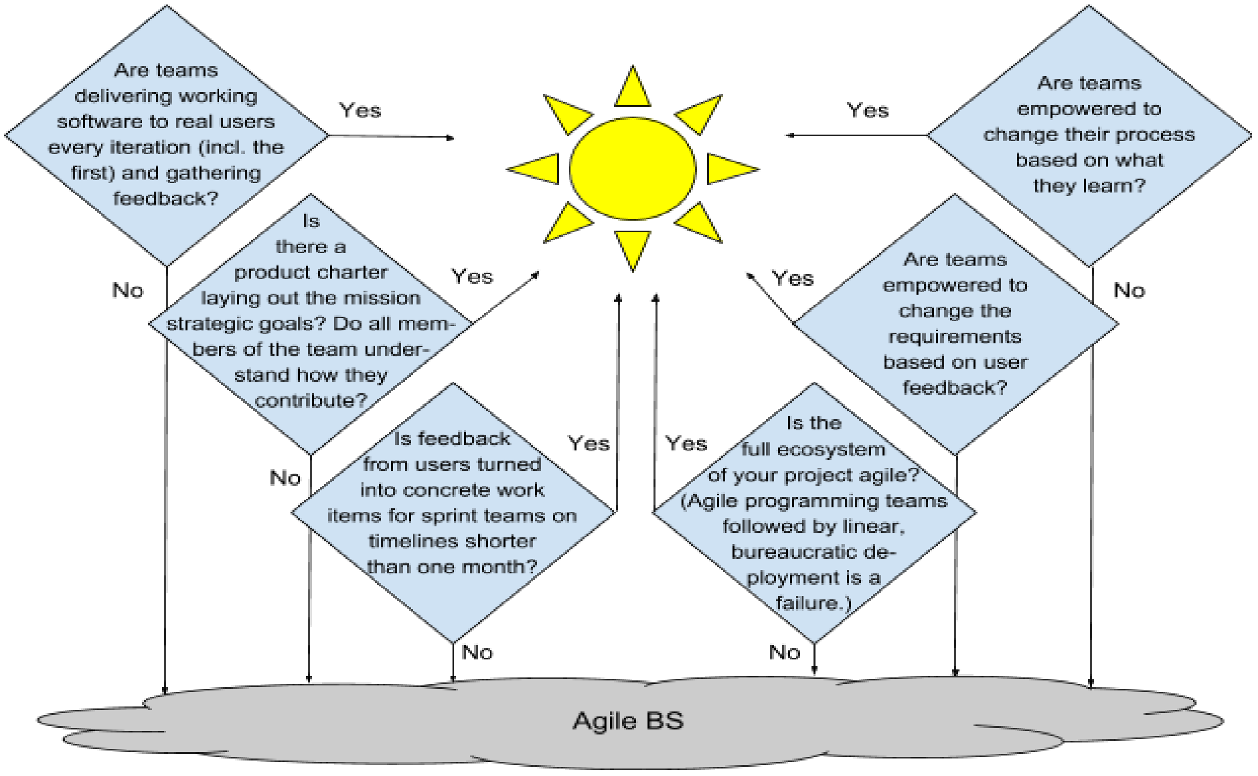
Questions for Program Leadership

- Are teams delivering working software to at least some subset of real users every iteration (including the first) and gathering feedback? (alt: every two weeks)

- Is there a product charter that lays out the mission and strategic goals? Do all members of the team understand both, and are they able to see how their work contributes to both?
- Is feedback from users turned into concrete work items for sprint teams on timelines shorter than one month?
- Are teams empowered to change the requirements based on user feedback?
- Are teams empowered to change their process based on what they learn?
- Is the full ecosystem of your project agile? (Agile programming teams followed by linear, bureaucratic deployment is a failure.)

For a team working on agile, the answer to all of these questions should be “yes.”

Graphical version:


Are teams delivering working software to real users every iteration (incl. the first) and gathering feedback?
Yes
No
Is there a product charter laying out the mission strategic goals? Do all members of the team understand how they contribute?
Yes
No
Is feedback from users turned into concrete work items for sprint teams on timelines shorter than one month?
Yes
No
Is the full ecosystem of your project agile? (Agile programming teams followed by linear, bureaucratic deployment is a failure.)
Yes
No
Are teams empowered to change the requirements based on user feedback?
Yes
Are teams empowered to change their process based on what they learn?
Yes
No
Agile BS

More information on some of the features of DoD software programs are included in the DIB’s “Ten Commandments of Software,” the DIB’s “Metrics for Software Development,” and the DIB’s “Do’s and Don’ts of Software.”

Is Your Development Environment Holding You Back?
A DIB Guide for the Acquisition Community

A strong software development team is marked by some common attributes, including the use of practices, processes, and various tools.

An effective team starts with clear goals. The entire software team should have a clear sense of the project's goals and the value they seek to provide "the client." The goals should be translated into specific objectives, which may be measured in terms of agreed-upon key performance indicators (KPIs) or other frameworks. An effective development environment is one designed to deliver value toward those goals. (This KPI-driven paradigm should *not* be seen as an invitation to reprise an extended debate about requirements.)

Technical practices and processes that enable a development environment to deliver value toward those goals include:

- Organization of activities through discrete "user stories" that can be broken down into smaller components and continually prioritized by the product owner
- Relatively short "sprints" (often two weeks), each ending in a retrospective, that enable measurement and learning throughout the process
- Blameless postmortems that allow for maximum learning and speedy recovery from failures
- Automated testing, security, and deployment
- Testing (including user testing) and security should be shifted to the left and be part of the day-to-day operations within the development teams
- Continuous integration, in which developers integrate code into a shared repository several times a day, and check-ins are then verified by an automated build for early problem detection
- Continuous delivery or continuous deployment, in which the software is seamlessly deployed into staging and production environments
- Trunk-based development, in which team members work in small batches and develop off of trunk or master, rather than long-lived feature branches
- Version control for all production artifacts including open source and third party libraries
- Infrastructure as code: version control for all configuration, networking requirements, container orchestration files, continuous integration/continuous delivery (CI/CD) pipeline files
- Ability to execute A/B testing and canary deployments
- Ability to get rapid and continuous user feedback and to test new features with users throughout the development process

Effective teams will practice continuous delivery, in which teams deploy software in short cycles, ensuring that the software can be reliably released at any time. Continuous deployment can be measured by a team's ability to achieve the following outcomes:

- Teams can deploy on-demand to production or to end users throughout the software delivery life cycle.

- Fast feedback on the quality and deployability of the system is available to everyone on the team, and people make acting on this feedback their highest priority.

Specific measures that will help you gauge if your development environment is working as it should include development frequency; lead time for changes; time to restore service after outage; and change failure rate (rollback deployed code). These questions and data, borrowed from the 2017 State of DevOps Report from DORA, can help assess where your teams stand:

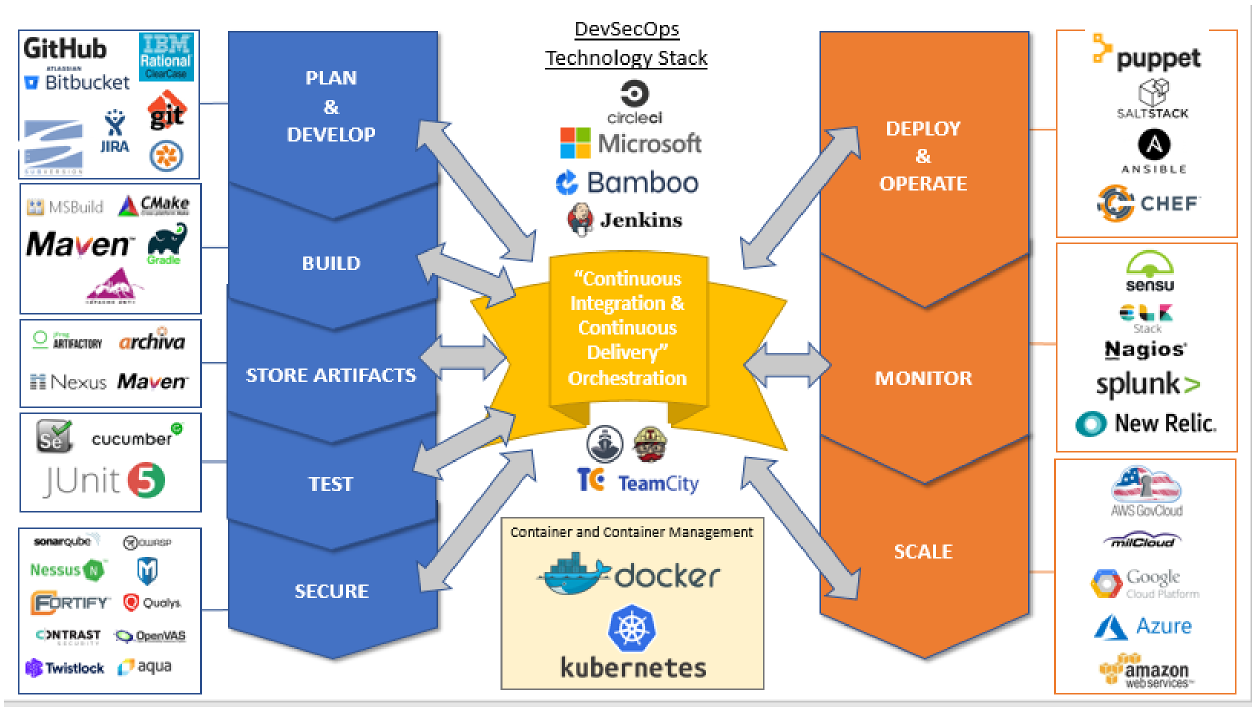
	High performance	**Medium performance**	**Low performance**
Deployment frequency How often does your organization deploy code?	On demand (multiple deploys per day)	Between once per week and once per month	Between once per week and once per month
Lead time for changes What is your lead time for changes (i.e., how long does it take to go from code-commit to code successfully running in production)?	Less than one hour	Between one week and one month	Between one week and one month*
Mean time to recover (MTTR) How long does it generally take to restore service when a service incident occurs (e.g., unplanned outage, service impairment)?	Less than one hour	Less than one day	Between one week and one day
Change failure rate What percentage of changes results either in degraded service or subsequently requires remediation (e.g., leads to service impairment, service outage, requires a hotfix, rollback, fix forward, patch)?	0-15%	0-15%	31-45%

* Low performers were lower on average (at a statistically significant level), but had the same median as the medium performers (2017 DevOps Report)

There is ***no exact set of tools*** that indicate that your development environment is working as it should, but the use of some tools will often indicate that the practices and processes above are in place. You commonly see effective software teams using:

- An issue tracker, like Jira or Pivotal Tracker
- Continuous integration and/or continuous integration/continuous delivery (CI/CD) tools, like Jenkins, Circle CI, or Travis CI
- Automated build tools, like Maven, Grable, Cmake, and Apache Ant
- Automated testing tools, like Selenium, Cucumber, J-Unit

- A centralized artifacts repository, like Nexus, Artifactory, or Maven
- Automated security tools for static and dynamic code analysis and container security, like Sonarqube, OWASP ZAP, Fortify, Nessus, Twistlock, Aqua, and more.
- Automation tools, like Chef, Ansible, or Puppet
- Automated code review tools, like Code Climate
- Automated monitoring tools, like Nagios, Splunk, New Relic, and ELK
- Container and container orchestration tools like Docker, Docker Swarm, Kubernetes, and more



Warning signs that you may have screwed up your development environment include:

- If teams cannot effectively track progress toward defined goals and objectives roughly every two weeks
- If teams cannot rapidly deploy various environments that mirror production to test their code such as in development, QA, and staging
- If teams cannot have real-time feedback regarding their code building, passing tests, and passing security scans
- If it takes months for end users to be able to see changes and provide feedback
- If teams cannot rapidly roll-back to previous versions or perform rolling-update to new versions without downtime
- If recovering from incidents results in significant drama or the assignment of blame
- If having code ready to deploy is a big event (it should happen routinely and without drama)
- If changes to the software frequently result in breaking it

If developers are not empowered to change the code or build new functionality based on user feedback, or to change their process based on what they learn.

Is Your Compute Environment Holding You Back?
A DIB Guide for the Acquisition Community

To enable software to provide a competitive advantage to the warfighter, DoD must adopt a strategy for rapidly transitioning DoD IT to current industry standards. This modernization agenda should include providing distributed databases and abundant computing power; making bandwidth available as a platform; integrating mobile technologies; and developing DoD platforms for downloading applications. This document outlines compute and infrastructure capabilities that should be available to DoD programmers (and contractors) who are developing software for national defense. The capabilities include:

1. **Scalable compute.** Access to computing resources should never be a limiting factor when developing code. Modern cloud environments provide mechanisms to provide any developer with a powerful computing environment that can easily scale with the needs of an individual programmer, a product development team, or an entire enterprise.

2. **Containerization.** Container technology provides sandbox environments in which to test new software without exposing the larger system to the new code. It “packages up” an application with all of the operating system services required for executing the application and allowing that application to run in a virtualized environment. Containers allow isolation of components (that communicate with each other through well-defined channels) and provide a way to “freeze” a software configuration of an application without freezing the underlying physical hardware and operating system.

3. **Continuous integration/continuous delivery (CI/CD) pipeline (DevSecOps platform).** A platform that provides the CI/CD pipeline is used for automated testing, security, and deployment. This includes license access for security tools and a centralized artifacts repository with tools, databases, and a base operating system (OS) with an existing authorization to operate (ATO).

4. **Infrastructure as code: automated configuration, updating, distribution, and recovery management.** Manual configuration management of operating systems and middleware platforms leads to inconsistencies in fielded systems and drives up the operating costs due to the labor hours required for systems administration. Modern software processes avoid this by implementing “infrastructure as code,” which replaces manual processes for provisioning infrastructure with automated processes that use machine-readable definition files to manage and provision containers, virtual machines, networking, and other components. Adopting infrastructure as code and software distribution tools in a standardized way streamlines uniformity of deployment and testing of changes, which are both vital to realizing the benefits of agile development processes.

5. **Federated identity management and authentication backend with common log file management and analysis.** Common identity management across military, government, and contractors greatly simplifies the assignment of permissions for accessing information across multiple systems and allows rapid and accurate auditing of

code. The ability to audit access to information across multiple systems enables the detection of inappropriate access to information, and can be used to develop the patterns of life that are essential for proper threat analysis. Common identity management can ease the integration of multi-factor authentication across servers, desktops, and mobile devices. Along with public key infrastructure (PKI) integration, it allows verification of both the service being accessed by the user and the user accessing information from the service.

6. **Firewall configuration and network access control lists.** Having a common set of OS and application configurations allows network access control not just through network equipment, but at the server itself. Pruning unnecessary services and forcing information transfer only through intentional interfaces reduce the attack surface and make servers more resilient against penetration. Server-to-server communication can be encrypted to protect from network interception and authenticated so that software services can only communicate with authorized software elements.

7. **Client software.** Remote login through remote desktop access is common throughout DoD. This greatly increases the difficulty of integrating mobile platforms and of permitting embedded devices to access vital information, especially from the field. It also complicates uniform identity management and multi-factor verification, which is key to securing information. By moving to web client access mobile integration - and development - is greatly eased. It also becomes possible to leverage industry innovation, as this is where the commercial sector is heading for all interactions.

8. **Common information assurance (IA) profiles.** Information assurance (IA) for DoD systems is complex, difficult, and not yet well-architected. Test, certification, and IA are almost always linear “tailgate” processes instead of being integrated into a continuous delivery cycle. Common IA profiles integrated into the development environment and part of the development system architecture are less likely to have bugs than customized and add-on solutions.

Desired State with Examples

Effective use of software requires sufficient resources for computing, storage, and communications. Software development teams must be provided with abundant compute, storage, and bandwidth to enable rapid creation, scaling, and optimization of software products.

Modern cloud computing services provide such environments and are widely available for government use. In its visits to DoD programs, the DIB Software Acquisition and Practices (SWAP) team has observed many programs that are regenerating computing infrastructure on their own—often in a highly non-optimal way—and typically due to constraints (or perceived constraints) created by government statutes, regulations, and culture. This approach results in situations where compute capability does not scale with needs; operating systems cannot be upgraded without upgrading applications; applications cannot be upgraded without updating the operating systems; and any change requires a complete information assurance recertification.

Compute platforms are thus “frozen” at a point established early in the program life cycle, and development teams are unable to take advantage of new tools and new approaches as they become available. The DIB SWAP team has noted a general lack of good tools for profiling code, maintaining access and change logs, and providing uniform identity management, even though the DoD has system-wide credentials through Common Access Control (CAC) cards.

It would be highly beneficial to create common frameworks and/or a common set of platforms that provide developers with a streamlined or pre-approved Authority to Operate (ATO). Use of these pre-approved platforms should not be mandated, but they create cost and time incentives by enabling more consolidated platforms. DoD could make use of emerging government cloud computing platforms or achieve similar consolidation within a DoD-owned data center (hybrid cloud). DoD should move swiftly from a legacy data center approach to a cloud-based model, while taking into account the lessons learned and tools and services available from commercial industry, with assumed hardware and operating system updates every 3-5 years.

Warning signs

Some indicators that you may have screwed up your compute environment include:

- Your programmers are using tools that are less effective than what they used in school
- The headcount needed to support the system grows linearly with the number of servers or instances
- You need system managers deployed with hardware at field locations because it is impossible to configure new instances without high skill local support
- You have older than current versions of operating systems or vendor software because it is too hard to test or validate changes
- Unit costs for compute, network transport and storage are not declining, or are not measurable to be determined
- Logging in via remote desktop is the normal way to access an information service
- You depend on network firewalls to secure your compute resource from unauthorized access
- You depend on hardware encryptors to keep your data safe from interception
- You have to purge data on a regular basis to avoid running out of storage
- Compute tasks are taking the same or longer time to run than they did when the system was first fielded
- Equipment or software is in use that has been “end of lifed” by the vendor and no longer has mainstream support
- It takes significant work to find out who accessed a given set of files or resources over a reasonable period of time
- No one knows what part of the system is consuming the most resources or what code should be refactored for optimization
- Multifactor authentication is not being used
- You cannot execute a disaster recovery exercise where a current backup up of a system cannot be brought online on different hardware in less than a day

Getting It Right

These capabilities should be available to all DoD programmers and contractors developing software for national defense:

Scalable compute

- Modern compute architectures
- Environments that make transitions across cloud and local services easy
- Graphics Processing Unit (GPU)- and ML-optimized compute nodes available for specialized tasks
- Standardized storage elements and ability to expand volumes and distribute them based on performance needs
- Standardized network switching options with centralized image control
- Property management tagging—no equipment can be placed in a data center without being tagged for inventory and tracked for End of Life support from vendors
- Supply chain tracking for all compute elements

Containerization

- Software deployment against standard profile OS image
- Containers can be moved from physical to cloud-based infrastructure and vice versa
- Applications and services run in containers and expand or contract as needed
- OS updates separated from application container updates
- Centralized OS patch validation and testing
- Containers can be scaled massively horizontally
- Containers are stateless and can be restarted without impact
- Configuration management for deployment and audit

Continuous integration/continuous delivery (CI/CD) pipeline (DevSecOps platform)

- Select, certify, and package best of breed development tools and services
- Can be leveraged across DoD Services as a turnkey solution
- Develop standard suite of configurable and interoperable cybersecurity capabilities
- Provide onboarding and support for adoption of Agile and DevSecOps
- Develop best-practices, training, and support for pathfinding and related activities
- Build capability to deliver a Software Platform to the Defense Enterprise Cloud Environment
- Self-service portal to selectively configure and deliver software toolkit with pre-configured cybersecurity capabilities

Infrastructure as code: automated configuration, updating, distribution and recovery management

- Ability to test changes against dev environments
- Standardized profiling tools for performance measurement
- Centralized push of patches and updates with ability for rapid rollback
- Auditing and revision control framework to ensure proper code is deployed and running
- Ability to inject faults and test for failover in standardized ways

- Disaster recovery testing and failover evaluation
- Utilization tracking and performance management utilities to predict resource crunches
- Standardized OS patch and distribution repositories
- Validation tools to detect manual changes to OS or application containers with alerting and reporting

Federated identity management and authentication backend with common log file management and analysis

- Common identity management across all DoD and contractors
- Common multifactor backends for authentication of all users along with integration of LDAP/Radius/DNS or active directory services
- Integrated PKI services and tools for automated certificate installation and updating
- Common DRM modules that span domains between DoD/contractors and vendor facilities that can protect, audit and control documents, files, and key information. All encrypted at rest, even for plain text files.
- Useful for debugging and postmortem analysis
- Develop patterns of life to flag unusual activity by users or processes
- Automated escalation to defensive cyber teams

Firewall configuration and network access control lists

- Default configuration for containers is no access
- Profiles for minimal amounts of ports and services being open/run
- All network communications are encrypted and authenticated, even on the same server/container

Client software

- Web-based access the norm, from desktops/laptops as well as mobile devices
- Remote login used as a last resort - not as the default
- Security technical implementation guides (STIGs) for browsers and plugins, as well as common identity management at the browser interface (browsers authenticate to servers as well as servers authenticating to browsers)
- Minimal state kept on local hardware - purged at end of session

Common information assurance (IA) profiles

- Enforces data encrypted in flight and at rest
- Software versions across DoD with automated testing
- Application lockdowns at the system level so only authorized applications can run on configured systems
- “Makefile” to build configurations from scratch from base images in standardized approved configurations
- Use of audit tools to detect spillage and aid in remediation (assisted via DRM

SWAP Program Visits: Questions and Observations

Programs Reviewed

Reviewed 6 programs to date:

- Next Generation fighter jet
- Next Generation ground system
- Kessel Run—AOC Pathfinder
- Space tracking system
- Naval radar system
- Cross-service business system

What we hope to understand:

- Why is the software the way it is?
- How have you gone about developing and deploying it?
- What constraints/obligations have you been under and what would be your recommendations to change those?

Standard Questions

- What is the coding environment and what languages/SW tools do you use?
- What do the software and system architectures look like?
- What is the computational environment (processing, comms, storage)?
- How is software deployed and how often are updates delivered to the field?
- What determines the cycle time for updates?
- How does software development incorporate user feedback? What is the developer-user interface? How quickly are user issues addressed and fixed?
- How long does it take to compile the code from scratch?
- How much access does the DoD have to the source code?
- How is testing done? What tool suites are used? How much is automated? How long does it take to do a full regression test?
- How is cybersecurity testing done? How are programs/updates certified?
- What does the workforce look like (headcounts, skill sets)? How many programmers? How much software expertise is there in the program office?
- What is the structure of the contract with the government? How are changes, new features, and new ideas integrated into the development process?

Preliminary Observations

- Software is being delivered to the field 2-10X slower than it could be due to outdated requirements, test requirements, and lack of trust in SW
- Many systems are using legacy hardware and outdated architectures that make it much harder to exploit advances in computing and communications

- Program requirements were often formulated 5+ years ago (when the threat environment + available technologies were very different => wasted effort)
- New capabilities and features are added in multi-year (multi-decade?) development "blocks" instead of continuously and iteratively
- Most program offices don't have enough expertise in modern SW methods
- Most SW teams are attempting to implement DevOps and "agile" approaches, but in most cases the capabilities are still nascent (and hence fragile)
- Transition to DevOps is often hindered by a gov't support structure focused on technical performance in a waterfall setting ("waterfall with sprints")
- Information assurance (IA) is complex, difficult, and not yet well architected
- Test, certification and IA are almost always linear "tailgate" processes instead of being integrated into a continuous delivery cycle.

What should be done differently in future programs?

- Spend time upfront getting the architecture right: modular, automated, secure
- Make use of platforms (hardware and software) that continuously evolve at the timescales of the commercial sector (3-5 years between HW/OS updates)
- Start small, be iterative, and build on success – or terminate quickly
- Construct budget to support the full, iterative life cycle of the software
- Adopt a DevOps culture: design, implement, test, deploy, evaluate, repeat
- Automate testing of software to enable critical updates to be deployed in days to weeks, not months or years (also requires changes in testing organization)
- Have a local team of DoD software experts who are capable of modifying or extending the software through source code or API access
- Separate development of mission level software from development of IA-accredited platforms

How to Justify Your Budget When Doing DevSecOps

As we transition software development from big spiral programs into DevSecOps, program managers will have to wrestle with using new practices of budget estimation and justification, while potentially being held to old standards that should no longer apply. In addition to all of the regular challenges of retaining a budget allocation (budget reviews, audits, potential reductions and realignment actions, all many times a year), defending a budget for a DevSecOps acquisition requires additional explanation and justification because those charged with oversight—whether inside the Department or in Congress—have come to expect specific information on a tempo that doesn't make sense for DevSecOps projects. Program managers leading DevSecOps projects therefore must not only do the hard work of leading agile teams toward successful outcomes, but also create the conditions that allow those teams to succeed by convincing cost assessors and performance evaluators to evaluate the work differently. Fortunately, commercial industry already has best practices for budget estimation and justification for DevSecOps and that DoD should follow industry approaches rather than create new ones

This DIB Guide is intended to help with this challenge. It seeks to provide guidelines and approaches to help program managers of DevSecOps projects[17] interact with those cost assessors and performance evaluators through the many layers of review and approval authorities while carrying out their vital oversight role. This guide should help with projects where the development processes is optimized for software rather than hardware and where most key stakeholders are aligned around the goal of providing needed capability to the warfighter without undue delay.

Questions that we attempt to answer in this concept paper:

1. What does a well-managed software program look like and how much should it cost?
2. What are the types of metrics that should be provided for assessing the cost of a proposed software program and the performance of an ongoing software program?
3. How can a program defend its budget if the requirements aren't fixed or are changing?
4. How do we estimate costs for "sustainment" when we are adding new features?
5. Why is ESLOC (effective source lines of code) a bad metric to use for cost assessment (besides the obvious answer that it is not very accurate)?

What does a well-managed DevSecOps program look like and how much should it cost?

The primary focus for DevSecOps programs is about regular and repeatable, sustainable delivery of innovative results on a time-box pattern, not on specifications and requirements without bounding time (Figure 1). The fixed-requirements spiral-development spending model has created program budgets that approach infinity. DevSecOps projects, on the other hand will be focused on different activities at different stages of maturity. In a DevSecOps project, management should be tracking services and measuring the results of working software as the product evolves, rather than inspecting end items when the effort is done, as would be expected

[17] Not all software is the same; we focus here only on software programs using or transitioning to DevSecOps.

in a legacy model. Software is never done and not all software is the same, but generally the work should look like a steady and sustainable continuum of useful capability delivery.

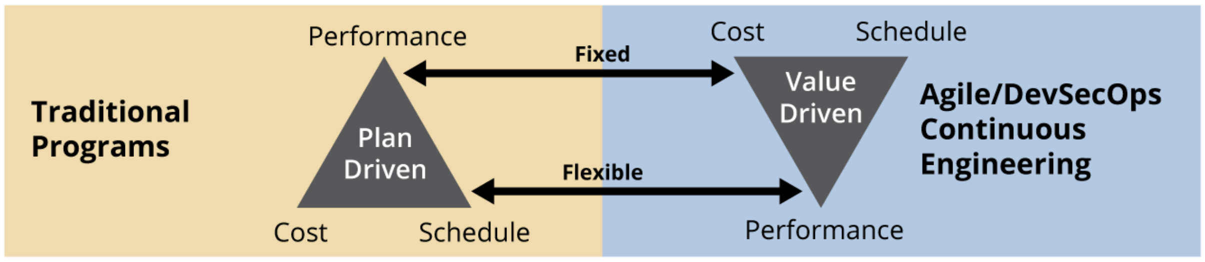


Figure 1. Value Driven Iron Triangle (Carnegie Mellon University, Software Engineering Institute).

- During the creation phase, program managers will most likely decide to adopt Agile based on criteria that fits their design challenge (e.g., software dependent). They would also be motivated to build their products on top of widely used software platforms that are appropriate for the technical domain at hand (e.g., embedded vs. web applications). During this phase team also establishes base capability and what they consider a minimum viable product (MVP).[18] This is where all programs start and many should end. Starting small and incrementing is not only the right way to do software, but it is also a great way to limit financial exposure. A key tenet of agile development is learning early and being ready to shift focus to increase the likelihood for success.
- During the scaling phase, the entire team (industry and government) commit and learn how to transition to appropriate agile activities that are optimizing for implementing DevSecOps for the project. This should focus the team on transitioning to a larger user base with improved mechanisms for automated testing (including penetration testing), red team attacks, and continuous user feedback. A key management practice in agile development is to keep software projects to a manageable size. If the project requires more scope, divide the effort into modular, easily connected chunks that can be managed using agile methods and weave the pieces together in implementation.
- Once into implementation, a well-managed program should have a regular release cadence (e.g., for IT projects every 2-3 weeks, while safety-critical products could run a bit longer, 3-4 weeks). Each of these releases delivers small increments of software that are as intuitive to use as possible and directly deployable to actual users. DevSecOps programs move from small successes into larger impacts.

With allowances made for different sizes of project, DevSecOps should share certain characteristics, including:

- An observer should easily find an engaged program office, as well as development teams that are small (5-11 people), and well connected to one another through structured meetings and events (a.k.a. “ceremonies”).

[18] The MVP should not be overspecified since the main goal is getting the MVP into the hands of users for feedback.

- A set of agile teams work on cross-functional capabilities of the system and include a planning team and a system architecture team.
- The teams should have frequent interaction with subject matter experts and users from the field or empowered product owners. Active user engagement is a vital element of an Agile approach, but getting actual users (*not* just user representatives) to participate also needs to be a managed cost that the program needs to plan for.
- The project should have a development environment that supports transparency of the activities of the development teams to the customer. Maximal automation of reporting is the norm for commercial development and should be for DoD programs as well.
- The program should include engaged test and certification communities who are deeply involved in the early stages (i.e., who have “shifted left”) and throughout the development process. Not just checkers at the end of that process. They would help design and validate the use of automation and computer-assisted testing/validation tools whenever possible as well.
- Capability should also be delivered in small pieces on a continuing basis—as frequently as every two weeks for many types of software (see the DIB’s Guide to Agile BS).

The cost of a program always depends on the scale of the solution being pursued, but in an agile DevSecOps project, the cost should track to units of 5–11-person cross-functional team (team leader, developers, testers, product owners, etc.) with approximately 6–11 teams making up a project. If the problem is bigger than that, the overall project could be divided up into related groups of teams. A reliance on direct interaction between people is another central element of Agile and DevSecOps; the communication overhead means that this approach loses effectiveness with too many people in a team (typically 5–11 cross-functional members). Also, groups of teams have difficulty scaling interactions when the number of teams gets too large (less than twelve). A team-of-teams approach will allow scaling to fit the overall scope. Organizing the teams is also a valuable strategy where higher level development strategies and system architectures get worked out and the lower level teams are organized around cross-domain capabilities to be delivered. Cost incentives for utilizing enterprise software platform assets should be so attractive, and the quality of that environment so valuable, that no program manager would reasonably decide to have his/her contractor build their own.

Here are some general guidelines for project costs when pursuing a DevSecOps approach:

- Create: deliver initial useful capability *to the field* within 3-6 months (the use of commodity hardware and rapid delivery to deployment). If this cannot be achieved, it should be made clear that the project is at risk of not delivering and is subject to being canceled. Outcomes and indicators need to be examined for systematic issues and opportunities to correct problems. Initial investment should be limited in two ways: 1) in size to limit financial exposure and 2) in time to no more than 1 year.

- Scale: deliver increased functionality across an expanding user base at decreasing unit cost with increased speed. Investment should be based on the rate limiting factors of time and talent, not cost. Given a delivery cycle and the available talent, the program should project only spending to the staffing level within a cycle.

- Good agile management is not about money, it is about regular and repeated deliver. That is to say, it is about time boxing everything. Releases, staffing, budget, etc. Nick, strongly recommend that you rework this to reflect time boxing as the most important aspect of “defending your agile budget.

- Optimize: deliver increased functionality fixed or decreasing unit cost (for a roughly constant user base). Investment limit should be less than 3 project team sets[19].

What are the types of metrics that should be provided for assessing the cost of a proposed software program and the performance of an ongoing software program?

Assessing the cost of a proposed software program has always been difficult, but can be accomplished by starting one or more set of project teams at a modest budget (1-6 sets of teams) and then adjusting the scaling of additional teams (and therefore the budget) based on the value those teams provide to the end user. It may be necessary to identify the size of the initial team required to deliver the desired functions at a reasonable pace and then price the program as the number of teams scales up. The DIB recommends that program managers start small, iterate quickly, and terminate early. The supervisors of program managers (e.g., PEOs) should also reward aggressive early action to shift away from efforts that are not panning out into new initiatives that are likely to deliver higher value. Justifying a small budget and getting something delivered quickly is the best way to provide value (and the easiest way to get and stay funded).

The primary metric for an *ongoing* program should be user satisfaction and operational impact. This can be different for every program and heavily depends on the context. The challenge, and therefore the responsibility of the PM then is to define mission relevant metrics to determine achieved and delivered value. Examples could include, personnel hours saved, number of objects tracked or targeted, accuracy of the targeting solution, time to first viable targeting solution, number of sorties generated per time increment, number of ISR sensors integrated, etc. Other key metrics that are often advocated by agile programs (inside and outside of DoD) include:

- *deployment frequency* (Is the program getting increments of functionality out into operations?),
- *lead time* (how quickly can the program get code into operation?),
- *mean time to recover* (how quickly can the program roll back to a working version, if problems are found in operation?), and
- *change fail rate* (rate of failures in delivered code).

These four break down into two process metrics (release cadence and time from code-commit to release candidate, and two are quality metrics (change fail rate and time to roll back). In addition, each project should also have 3-5 key value metrics that are topical to the solution space being addressed. Metrics must be available both to the teams and the customer so they can see how their progress compares to the projected completion rate for delivering useful functionality. A key reason for Government access to those metrics is for supporting the real-time tracking of progress and prediction of new activities in the future. The biggest difference between a DevSecOps

[19] Average of 8 people per team with an average of 8 teams per project.

program and the classic spiral approach is that the cadence of information transparency between the developers and the customer is, at slowest, weekly, but if properly automated, should be instantly and continuously available.. Quality metrics and discovery timelines (such as defects identified early in development versus bugs identified in the field) can also be used to evaluate the maturity of a program. This kind of oversight enables fast and effective feedback before the teams end up in extremis, or set up unrealistic expectations.

Software projects should be thought of as a fixed cadence of useful capability delivery where the “backlog” of activities are managed to fit the “velocity” of development teams as they respond to evolving user needs. Data collected on developers inside of the software development infrastructure can be provided continuously, instead of packaged into deliverables that cannot be directly analyzed for concerns and risks.

The DIB’s “Metrics for Software Development” provide a set of metrics for monitoring performance:

1. Time from program launch to deployment of simplest useful functionality.
2. Time to field high priority functions (spec → ops) or fix newly found security holes
3. Time from code committed to code in use
4. Time required for regression tests (automated) and cybersecurity audit/penetration tests
5. Time required to restore service after outage
6. Automated test coverage of specs/code
7. Number of bugs caught in testing vs field use
8. Change failure rate (rollback deployed code)
9. Percentage of code available to DOD for inspection/rebuild
10. Complexity metrics
11. Development plan/environment metrics

These data provide management flexibility since data about implementation of capability can be made *during* development—instead of at a major milestone review or after “final” delivery, when changing direction comes at a much higher cost and schedule impact. So data collection and delivery must be continuous as well. Another note, these metrics are recommendations and not intended to be prescriptive. Use what fits your program. Not all of these may be required.

An additional pair of overarching key metrics are headcount and expert talent available. If the project headcount is growing, but delays are increasing,, aggressive management attention is called for. The lack of expert talent also increases risks of failure.

How can a program defend its budget if the requirements are not fixed years in advance, or are constantly changing?
It is relatively easy to defend changing capability by making changes to the software of existing systems, as compared to starting up a new acquisition. Software must evolve with the evolving needs of the customers. This is often the most cost effective and rapid way to respond to new requirements and a changing threat landscape. A new approach to funding the natural activities of continuous engineering and DevSecOps requires a system that can prioritize new features and manage these activities as dependent and tightly aligned in time

(see Figure 1). A continuous deployment approach is needed for delivering on the evolving needs culled from user involvement combining R&D, O&M, Procurement, and Sustainment actions within weeks of each other, not years (see Figure 2). Great software development is an iterative process between developers and users that see the results of the interaction in new capability that is rapidly put in their hands for operational use.

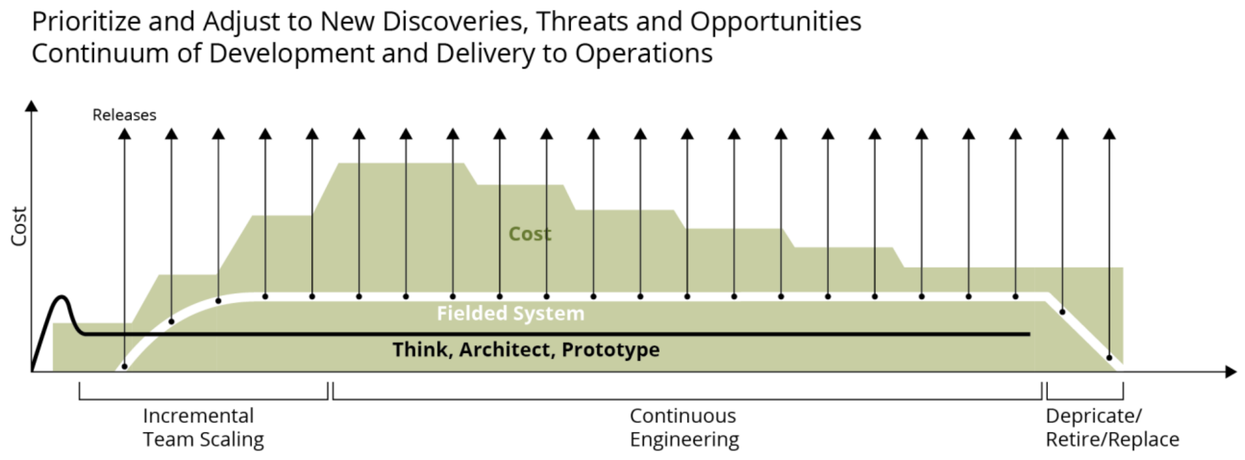

Prioritize and Adjust to New Discoveries, Threats and Opportunities
Continuum of Development and Delivery to Operations
Releases
Cost
Cost
Fielded System
Think, Architect, Prototype
Incremental Team Scaling
Continuous Engineering
Depricate/ Retire/Replace

Figure 2. Continuous Delivery of Modular Changes to Working Software (Carnegie Mellon University, Software Engineering Institute).

Elements to address include in budget justification and management materials:

- DevSecOps programs have to be at least as valuable and urgent to fund as a classic DoD spiral program in the hyper-competitive budget environment. Over time, DoD will realize that the DevSecOps approach is inherently more valuable. However, time is of the essence. It must be acknowledged that the current waterfall approach is no longer serving us well in the area of software. The mainstream software industry has already made the move to agile ten years ago and the methods are rigorously practices and proven valuable.
- The classic approach of doing cost estimates of designs based on fixed requirements has always been wrong, even when accounting for intended capability growth because the smart adversaries get a continuous vote on the threat environment. Accurate prediction of a rapidly changing technology environment and solution methods only exacerbate the unknowns of product development outcomes.
- DevSecOps programs have requirements, but start out at a higher level and use a disciplined approach to continuously change and deliver greater value.
- DIB's "Ten Commandments of Software" calls for the use of shared infrastructure and continuous delivery, which will reduce the cost of infrastructure and overhead, thus freeing up capital to advance unique military capability.
- Data available above the program manager's level has been insufficient for cost and program evaluation communities to assess software projects. However, the reporting of metrics that are a natural consequence of using DevSecOps approaches should be automated to provide transparency and rapid feedback.

The benefits of this approach are manifold. It allows for thoughtful rigor up front and early and the rapid abandonment of marginal or failure-prone approaches early in the design cycle before large investments are sunk. Details are allowed to evolve. More stable chunks of capability are defined at the “epic” level and a stable cadence of engineering and design pervades the life cycle. Under this operational concept, testing is performed early, during the architecture definition stage and continuously as new small deployments of functionality are delivered to the user. The identification of budget is redistributed as value is provided and validated for warfighting impact. A closer alignment of flexible requirements and budget allocation/ appropriation will be necessary in order to ensure that the national defense needs and financial constraints are continuously managed.

Continuous access to design and delivery metrics will illuminate developer effectiveness, user delight, and the pace of delivery for working code to include analytical data for in-stride oversight and user/programmatic involvement This will replace the standard practice of document-based deliverables and time-late data packages that take months to develop and are not current when provided.

The way that DoD has classically managed these activities is to break them up into different “colors of money” associated with hardware-centric phases (see Figure 3). This places an artificial burden on excellence in software. Rapid and continuous delivery of working code requires addressing these different types of requirements within shorter time-horizons than is natural for the existing federal budgeting process.

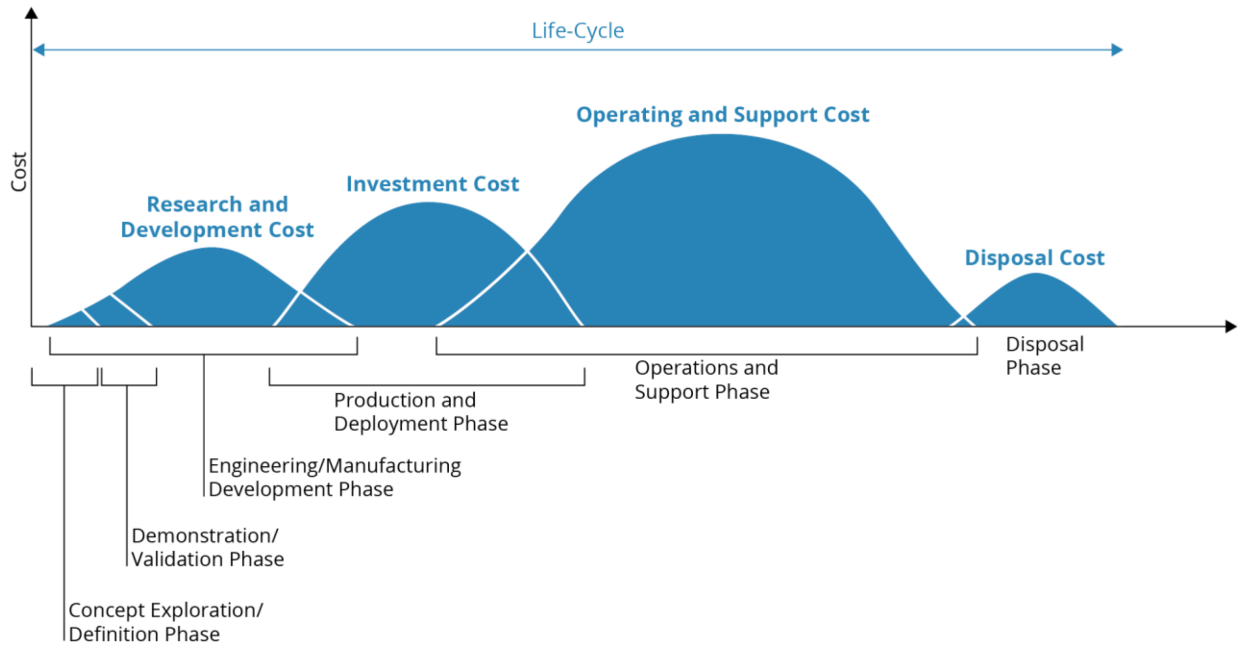


Figure 3. Notional DoD Weapon System Cost Profile (Defense Acquisition University).

In addition, the classic approach of developing detailed technical requirements far in advance of performing product design needs to be replaced. The new paradigm must begin with an architecture that will support the requirements and scale associated with needs for future compatibility (e.g., modularity security, or interoperability). Also, using an agile approach, a program can incorporate the best available technologies and methods throughout the entire life cycle and avoid a development cycle is longer than the useful life of the technology it is built on. Getting these things wrong is not recoverable. Establishing detailed requirements over a period of years before beginning, to be followed by long development efforts punctuated by major design reviews (i.e., Software Requirements Review, Preliminary Design Review, Critical Design Review, Test Readiness Review, Production Readiness Review) that require a span of years between events are inherently problematic for software projects for at least two reasons. First, these review events are designed around hardware development spirals that are time-late and provide little in the way of in-stride knowledge of software coding activities that can be used to aid in real-time decision making. Second, development teams are in frequent contact with users and adjusting requirements as they go, which up-ends the value of major design reviews that are out of cadence with the development teams. DevSecOps implementation methods such as feature demonstrations and cycle planning events provide much more frequent and valuable information on which program offices can engage to make sure the best value is being created.

Defending a budget has to be done in terms of providing value. Different programs value different things—increasing performance, reducing cost, minimizing the number of humans-in-the- loop—so there is no one size fits all measure. But in an agile environment, knowing what to measure to show value is possible because of the tight connection to the user/warfighter. Those users are able to see the value they need because they are able to evaluate and have an impact on the working software. This highlights the need to collect and share the measures that show improvement against a baseline in smaller increments.

How do we do cost for “sustainment” when we are adding new features?

The first step is to eliminate the concept of sustaining a fixed base of performance. Software can no longer be thought of as a fixed hardware product like a radar, a bomb, or a tank. That leads to orphaned deployments that need unique sustainment and a growth of spending that does not deliver new functionality (see Figure 4).

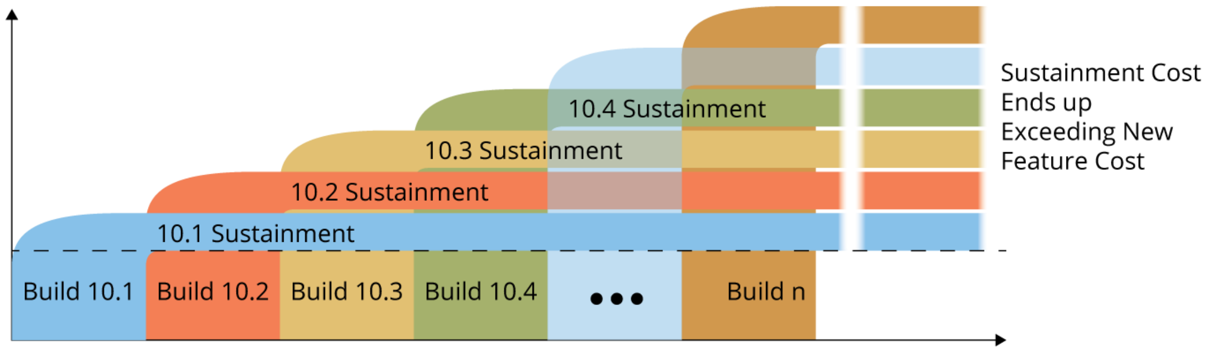


Figure 4. Layers of Sustainment to Manage Unique Deployments

Software can continue to evolve and be redeployed for comparatively little cost (see Figure 2). Users continue to need and demand greater performance and improved features, if for no other reason than to retain parity with warfighting threats. Also internal vulnerabilities and environmental updates must be continuously deployed to support ever improving cyber protections. The most secure software is the one that is most recently updated. Lastly, new capabilities for improved warfighting advantage are most often affordably delivered through changes to fielded products.

Software development is a very different way of delivering military capability. It should be considered more like a service of evolving performance. When new features are needed, they get put in the backlog, prioritized, and scheduled for a release cycle (see Figure 5). If the program is closer to providing satisfactory overall performance, then the program can dial down to the minimum level needed to satisfy the users and keep the environment and applications cyber-secure. It can be thought of as recursive decisions on how many (software) "squadrons" are required for our current mission set and then fund those teams at the needed staffing level to create, scale, or optimize the software (depending on the stage of continuous development). Because these patterns can be scaled up and down by need in a well-orchestrated way, new contracting models are available that might not have been used in the past. For example, fixed price contracts for a development program was strongly discouraged, but under this model, where schedule and team sizes are managed and capability is grown according to a rigorous plan (Figure 1), a wider array of business, contracting and remuneration models can be explored.

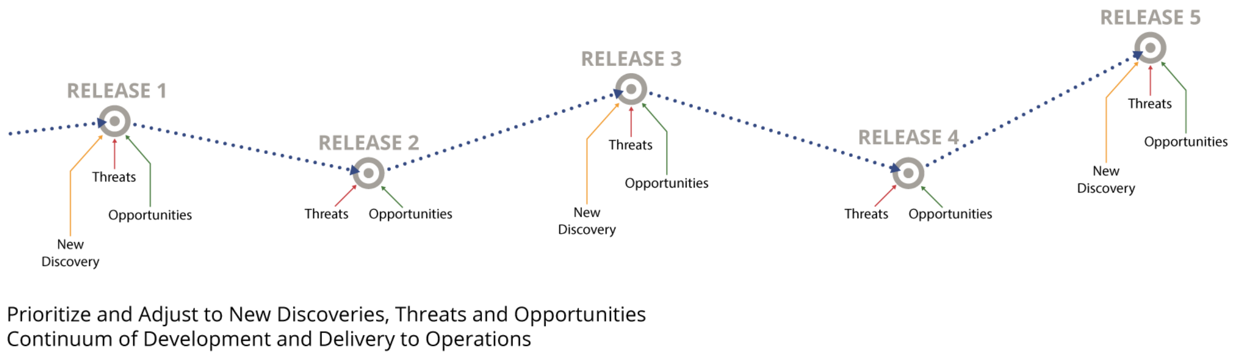


Figure 5. Release Cycle With New Opportunities, Discoveries and Response to Threats (Carnegie Mellon University, Software Engineering Institute).

Two financial protections built into acquisition laws and regulations need to be reexamined in the light of software being continuously engineered, vice sustained: Nunn-McCurdy and the Anti-Deficiency Act. The continuous engineering pipeline will continue to push out improved capability until the code base is retired. While Nunn-McCurdy is a valid constraint for large hardware acquisitions, it does not apply to software efforts. In a similar vein, software should also never trigger the Anti-Deficiency Act - just like keeping a ship full of fuel, or paying for air-traffic controllers; we know we are going to be doing these things for a long time. To build a ship that will need fuel for 40 years does not invoke the ADA. Therefore, starting a software project that will incrementally deliver new functionality for the foreseeable future should not do so either.

Why is ESLOC a bad metric to use for cost assessment?

The thing we really want to estimate and then measure is the effort required to develop, integrate, and test the warfighting capability that is delivered by software. SLOC might have been a used as a surrogate for estimating the effort required, but it has never been accurate. Not all software is the same, not all developers are the same, and not all development challenges use the same approaches to reduce problems into solutions. For example, in a project there may things like detailed algorithms that require deep expertise and detailed study to properly implement small amounts of code, running alongside large volumes of automatically generated code of relatively trivial complexity. Many different levels of effort are needed to create a line of code that will deliver military capability, and estimations of source code volume is an inherently problematic and error-filled approach to describing the capability thus produced. That's why DevSecOps efforts use measures of relative effort like story points to communicate across a particular set of teams how much effort it will take to turn a requirement into working software that meets an agreed upon definition of done within a set cadence of activity. Because these story points are particular to a specific team, they do not accurately transition to generally prescribable measures of cost.

Estimating by projecting the lines of code starts the effort from the end and works backwards. SLOC is an output metric (something to know when the job is done—akin to predicting what size clothing your child will wear as an adult). It does not capture the human scale of effort. Traditional models like COCOMO or SEER attempt to use a variety of parameters in their models to capture things like formality, volatility, team capabilities, maturity and others. However, these surrogates for effort have well documented error sources and have failed time and again to accurately capture the cost of executing a software program. There are also inherent assumptions built into these models that are obviated by performing agile development of capability models running on a software platform.

In the beginning stages of DoD's transformation to DevSecOps methods, the development and operations community will need to work closely with the cost community to derive new ways of predicting how fast capability can be achieved. For example, estimating how many teams worth of effort will be needed to invest in a given period of time to get the functionality needed. As they do this, it needs to be with the understanding that the methods are constantly changing and the estimation methods will have to evolve too. New parameters are needed, and more will be discovered and evolve over time.

Appendix F: SWAP Working Group Reports

The information in this appendix was developed based on feedback and analysis performed by members of a working group that included subject matter experts (SMEs) within the Department who provided input for consideration to the SWAP study. The working group was asked to: (1) distill the feedback received from case studies, interviews, literature reviews, and feedback from the Board members into main issue points; (2) as SMEs identify the statutory, regulatory, and cultural obstacles to achieving the Board's vision for a desired end state; and (3) provide suggested language to remove the barriers.

The following reports were generated by 10 subgroups:

- Acquisition Strategy
- Appropriations
- Contracting
- Data and Metrics
- Infrastructure
- Requirements
- Security Accreditation/Certification
- Sustainment and Maintenance
- Test and Evaluation
- Workforce

These reports describe input to the SWAP study and the specific views and ideas for change in the reports do not necessarily reflect the final views of the SWAP study. These reports have been lightly edited by the study for consistency with the terminology of this report and are included to provide context and insight into our final themes, lines of effort, and recommendations.

Appendix F.1: Acquisition Strategy Subgroup Report

Contributing authors: Melissa Naroski Merker (lead), Jeff Boleng, Nicolas Chaillan, Ben FitzGerald, Jonathan Mostowski, Don Johnson, COL Harry Culclasure (809 Panel), Gabe Nelson (809 Panel), Larry Asch (809 Panel), Nick Tsiopanas (809 Panel), Nick Kosmidis.
Additional advice / assistance from MITRE, IDA, and DAU

This appendix examines pain points, obstacles, change ideas, and future vision for the Defense Innovation Board (DIB) Software Acquisition and Practices (SWAP) Study in the area of Acquisition Strategy and Oversight (i.e., *Acquisition Environment*). In 2017 the Office of the DASD(C3CB) under the ASD(A) commissioned an IT acquisition study with Deloitte. The study recommended the following attributes of an effective and efficient IT acquisition structure:

- *Fast* to incorporate current technology and make efficient use of Agency resources
- *Flexible* and adaptable to support rapid changes in technology and input from stakeholders about capability needs
- *Collaborative* to seek stakeholder involvement and input to be incorporated throughout

In a previous study completed in September 2016, Deloitte also provided key findings on commercial IT practices. Findings were taken into consideration when forming the proposals following in this appendix. The team recognizes that DoD is falling short of the preferred attributes outlined above with the current IT acquisition structure, in addition to multiple statutory, regulatory, and cultural issues that currently hinder an effective and efficient DoD acquisition environment that would benefit from reform.

Pain Points

Acquisition Policy Environment. DoD lacks a cohesive acquisition policy architecture and robust policy for software acquisition. Existing policies, to include tangential or supplemental policies that are integral to the operation of the defense acquisition system, do not fit well together and result in discrepancies, conflicts, and gaps. The defense acquisition system is monolithic, compiled in pieces as needs arose instead of as an integrated and evolving environment. It has proven unable to keep up with or remain ahead of the pace of change and technological advancements that require speed and agility. While it has regularly been revised, the changes tend to be conservative and incremental, requiring the agreement of too many parties protecting narrow interests and who are reluctant to relinquish authority or evolve. The system remains focused on oversight and situational control rather than insight and trust. The policies, practices, and documents become quickly entrenched and manifest themselves in the form of the Department's culture, leading to additional bureaucracy and decreased levels of organizational trust, that are difficult to rapidly reverse. Furthermore, the environment is risk averse, seeking out what is perceived to be the "safest" route to get things done, stifling the innovation and risk- taking that's required to maintain an advantage over adversaries.

As an example, one DoD weapons system program, which is implementing a DevSecOps pipeline to enable agile capability releases, informed us it took 18 months to get approval of a Test and

Evaluation Master Plan (TEMP). The process within the TEMP drove them into sequential developmental and operational test—which is antithetical to continuous delivery under the DevSecOps concept.

Governance and Management. The Department lacks a strategic approach that recognizes software's criticality as the backbone and nervous system of the Department's mission and operations, often leading to widespread duplication of capabilities that could be consolidated and scaled at an enterprise level (whether Service-enterprise or OSD-enterprise). This absence of any strategy, compounded by a long-standing lack of organizational trust in the Department, is exemplified by various situations in the software environment. For example, the lack of reciprocity on matters such as security standards, architecture, and compliance methods—my way is "better" (insert "less expensive," "more efficient," "more effective") than your way, or, "our requirements/processes are unique," regardless of validity. Further, DoD issues separate policies on matters such as cloud, architecture, and risk management, with no unified approach at the strategic level. Management and governance of these matters takes the form of prolific numbers of senior working groups (or equivalent) that make few decisions but have frequent meetings. DoD's lack of an overarching strategic plan for key technologies, with a robust decision making framework that pushes responsibility and authority down to the lowest executable level, creates inefficiency, duplication, and waste.

Organization and Culture. DoD lacks an organizational structure with clear responsibility and authority for software acquisition and management; there are confusing roles and responsibilities between DoD CIO, USD(A&S), and the DoD CMO. This state of ambiguity leads to overlap, inefficiency, and unnecessary bureaucracy; and it is replicated at the Service level. The result is a slow, rigid, siloed organization unable to adapt in the present and plan for the future in order to maintain competitive advantage. DoD is not a change-ready environment and the acquisition system was not designed for rapid change. DoD employees tend to receive change mandates rather than participating in them. A case in point is that when DoD issues a policy, the Services will implement their own supporting version or "supplemental guidance," which expands the policy and introduces multiple layers of bureaucracy, eliminating any semblance of flexibility that was intended by the original policy issued. For example, the Department issued DoD Instruction 5000.75 in February 2018, a tailored requirements and acquisition approach for business systems. Subsequently, the Army produced accompanying implementation guidance—91 pages—which introduces additional forms, templates, processes, and time constraints.

Desired (end) state An acquisition system that enables rapid delivery of cost-efficient, relevant software capability through the application of creative compliance and fact-based critical thinking under a logical and minimal policy framework. The Department treats software as a national security capability and continuously retrains the workforce to be able to adapt to an ever-changing technology environment, embraces continuous collaboration between user and developers, embraces changing requirements, accepts and take risks, and deliver adversary- countering capabilities to the warfighter. Executing the approach requires an end state with an efficient contracting environment; a culture that rewards informed risk-taking and fast failures; the use of limits or guardrails instead of prescriptive requirements that limit creativity; outcome-based metrics that focus on value vs. execution against a plan; and a move away from traditional funding models and compliance-driven management.

Obstacles The Department operates with a general lack of urgency regarding its software—it is not recognized or treated as a national security capability. There is an aversion to informed risk-taking regarding new and innovative approaches to doing business and adopting emerging (or even simply relevant) technologies, even though it's risky, or riskier, to continue using outdated technologies that are not secure or facing obsolescence in the face of evolving threats. Dramatic changes in policy or process are viewed as risky yet our current ways of operation are not despite a known degradation in strategic advantage previously enjoyed over adversaries. The inability to evolve and support rapid changes in technology and input from stakeholders about capability needs is bred through organizational silos and stovepipes that stifle the collaboration necessary to develop and operationalize software. Further, stakeholder involvement is limited by following restrictive controls, timelines, and processes in a sequential manner that impedes progress and results in a lower state of readiness. The duplication of authorities and responsibilities among organizations both horizontally and vertically, within the defense acquisition system only exacerbates an already complex environment where a protectionist culture is ingrained and the workforce is not incentivized to change. In its endeavors to improve the status quo, "help" from Congress over the past decades translates into entrenched policies, processes, and procedures—"cultural norms" that are difficult to reverse.

Ideas for Change

Acquisition Policy Environment. Define software as a critical national security capability under Section 805 of FY16 NDAA "Use of Alternative Acquisition Paths to Acquire Critical National Security Capabilities." Create an acquisition policy framework that recognizes that software is ubiquitous and will be part of all acquisition policy models. Recommend the creation of a clear, efficient acquisition path for acquiring non-embedded software capability. Reconcile and resolve discrepancies among supplemental policies that lead to conflicts. Consider the following tenets in development of a reformed software acquisition policy:

- Emphasis on quickly delivering working software
- Encourage projects and pilot efforts that serve to reduce risk and complexity - fail fast
- Reimagine program structures and program offices—i.e., accommodate move to "as-a-service" capabilities, agile, microservices, and micro-applications
- Iterative, incremental development practices based on agile methods
- Rapid adoption of emerging technologies through piloting or prototyping
- Elimination of traditional A, B, C milestones; replaced by more sprint-centric decision points
- Elimination of arbitrary phases or merge phases to reflect rapid, agile development methods
- Tailor in requirements (statutory, regulatory—i.e., documentation) rather than tailor out;

start with a minimum set

- No big-bang testing with sequential DT/OT; move to fully integrated test approaches driven by automated testing as well as regular, automated cybersecurity scanning
- Use a “guardrail-based” (upper/lower limit) approach for software requirements rather than defining every requirement up front
- Track value-driven outcome metrics which can be easily and continuously generated rather than measuring execution against a plan

Governance and Management - Software as an Asset. Develop an enterprise-level Strategic Technology Plan that reinforces the concept of software as a national security capability. Include an approach for enterprise-level DevSecOps and other centralized infrastructure development and management, an approach for shared services, and applications management. The plan should recognize how disruptive technologies will be introduced into the environment on an ongoing basis. Ensure appropriate integration of a data strategy and the Department’s Cloud Strategy. Examine a Steering Committee approach for management.

Organization and Culture Reform. Examine roles and responsibilities with the intent to streamline reconcile, and resolve discrepancies for software acquisition and management among the DoD CIO, the USD(A&S) and the CMO. Re-focus the software acquisition workforce on teaming and collaboration, agility, improved role definition, career path advancement methods, continuing education and training opportunities, incentivization, and empowerment. Involve them in the change process.

Appendix F.2: Appropriations Subgroup Report

Contributing author: Jane Rathbun (lead)

The Department's current Planning, Programming, Budgeting and Execution (PPBE) system framework and process using defined Program Elements (PEs), is categorized by life-cycle–phased appropriations, and requires two years or more in lead time from plan to start of execution. This approach was designed and structured for traditional waterfall acquisition used to deliver monolithic platforms such as aircraft, ships, and vehicles. The PPBE framework and process is challenging when leveraging agile and iterative acquisition methodologies to deliver software-intensive, information-enabling capabilities through a continuous delivery process. The current process limits the ability to quickly adapt systems against rapidly changing threats and increases the barriers for integrating advancements in digital technology in a timely and effective manner.

Pain Points and Obstacles

Appropriation methods intended for hardware systems and platforms are not consistent with the speed and technology pace of modern software and how it is successfully acquired and deployed. DoD continues to acquire and fund information-centric systems using processes designed for hardware-centric platforms. Current funding decision processes and data structures do not effectively support leading software development practices. As a result, DoD is not effective in leveraging and adapting at the pace of innovation seen in industry. Differentiating continuous iteration and continuous delivery of software workload into hardware-defined phases (Research, Development, Test & Evaluation (RDT&E), Procurement, or Operations and Maintenance (O&M)) is meaningless in a world view where software is never done - not because planned work isn't accomplished but because modern methods allow a project to continuously improve, adapt to evolving threats, and take advantage of rapid technology advances. There should be no final fielding or sustainment element in continuous engineering. System defined program elements hinder the ability to deliver holistic capabilities and services and do not enable real-time resource, requirements, performance, and schedule trades across systems without significant work.

Establishing a culture of experimentation, adaptation and risk-taking is difficult. The Department requires a process that supports early adoption of the most modern information-centric technologies and enables continuous process and capability improvement. The Deputy Secretary of Defense directed aggressive steps "…to ensure we are employing emerging technologies to meet warfighter needs; and to increase speed and agility in technology development and procurement." The current cycle of planning, budgeting, and executing across appropriation categories slows acquisition, development, and execution to a pace that is not sustainable for mission success.

Desired state. The desired state for the Department would be one in which continuous capability deployment throughout a software program's life cycle is possible, and the lengthy two-plus year lead times for programming and budgeting is removed. This would provide flexibility to execute desired features with the speed and agility necessary to meet the rapid changes in threats, information technologies, processes, and services. The single appropriation across the life cycle of a capability will enable continuous development, security, and operations (DevSecOps); allow for minimum viable product delivery at a relevant speed; support the use of managed (or cross PoR/enterprise) services; provide for greater transparency for information-centric capabilities; and provide the flexibility to pursue the most effective solution available at the time of acquisition without current restrictions of appropriations.

Ideas for change. *A new multi-year appropriation for Digital Technology needs to be established for each Military Defense Department and the Fourth Estate.* This appropriation fund would provide a single two-year appropriation for the life-cycle management of software-intensive and infrastructure-technology capabilities. This could be a stand-alone appropriation, or fall under the umbrella of an already established appropriation, with the appropriate caveats that allow it to behave as the single source of funding across the life cycle. The Department would seek to couple this new appropriation with the movement to a capability or service portfolio management construct. A project framework within each capability PE (i.e., logistics or intelligence) would represent the systems and key investments supporting the delivery of information-centric capabilities such as data conditioning and process reengineering. Capability portfolio management would better enable agile/iterative force development and management decisions to include realignment of resources from one system to another system or process reengineering effort within the portfolio to increase the velocity of minimum viable product output and overall capability delivery. PPBE decision making would be adjusted to allow for less detail in the programming process and greater specificity in the budgeting process—as close to execution as possible—to realize the benefits of agile/iterative development.

- The Components should program, budget, and execute for information and technology capabilities from one appropriation throughout the life cycle rather than using RDT&E, procurement, or O&M appropriations, which are often applied inconsistently and inaccurately. This will allow for continuous engineering.
- Within each Component-unique Budget Activity (BA), Budget Line Items (BLINs) align by functional or operational portfolios. The BLINs may be further broken into specific projects to provide an even greater level of fidelity. These projects would represent key systems and supporting activities, such as mission engineering.
- By taking a portfolio approach for obtaining software intensive capabilities, the Components can better manage the range of requirements, balance priorities, and develop portfolio approaches to enable the transition of data to information in their own portfolios and data integration across portfolios to achieve mission effects, optimize the value of cloud technology, and leverage and transition to the concept of acquisition of whole data services vice individual systems.
- This fund will be apportioned to each of the Military Departments and OSD for Fourth Estate execution.
- Governance: management execution, performance assessment, and reporting would be aligned to the portfolio framework—BA, BLI, project.

Appendix F.3: Contracting Subgroup Report

Contributing author: Jonathan Mostowski (lead)

The contacting challenges faced by DoD today are almost entirely cultural. This premise is asserted by instances of excellence throughout the Department where effective contracting methods have been executed (DDS, DIU, Kessel Run).

That said, rather than attempting to battle each cultural challenge as they arise, it is easier to create a new modern acquisition platform from which to execute contracts that starts from a point of "how should it be done" as a product of "what should we be buying."

The historical acquisition system was created to prevent fraud. The new priority is to establish technical superiority over our adversaries. While the prevention of fraud continues to be, and always will be, important, as a singular priority it serves to undermine the current identified need of speed and efficiency, which results in technical excellence for the Department.

Pain Points

Individual contracts are subject to review processes designed for large programs (of which they are likely enabling). This limits the agility of individual contract actions, even when modular contracting approaches are applied. In addition, the acquisition process is rigid and revolves around templates, boards, and checklists thus limiting the ability for innovation and streamlining execution.

Contracts focus on technical requirements instead of contractual process requirements. The contract should address overall scope (required capability), Period of Performance and price. The technical execution requirements should be separate and managed by the product owner or other technical lead.

Intellectual Property (IP) rights are often genetically incorporated without considering the layers of technology often applied to a solution. A single solution might include open source, proprietary software, and government custom code. The IP clauses should reflect all of the technology used.

Desired State

The desired state is an acquisition model that is liberated from the decades of policy and regulations that singularly focus on fraud prevention and provides for efficiency allowing DoD to keep pace with the private sector and adversaries. This can be accomplished through a new authority Congress establishes a separate *new* authority for contracting for software development and IT modernization.

Obstacles

- Requires act of Congress ⇒ work with Armed Service Committees Staffers
- There is no infrastructure to support this ⇒ establish policy for guidance
- There are no Contracting Officers with specific certifications ⇒ Leverage current certifications

- Could cause confusion on implementation (what applies, what doesn't) ⇒ A&S issues guidance

Ideas for Change

Congress establishes a separate *new* authority for contracting for software development and IT modernization

To address "Individual contracts being subject to review processes designed for large programs":

- Treat procurements as investments "what would you pay for a possible initial capability" (cultural).
- Manage programs at budget levels, allow programs to allocate funds at a project investment level (policy).
- Work with appropriators to establish working capital funds so that there is not pressure to spend funds quicker then you're ready (iterative contracts may produce more value with less money) (statute).
- Leverage incentives to make smaller purchases to take advantage of simplified acquisition procedures (cultural).
- Revise estimation models - source lines of code are irrelevant to future development efforts, estimations should be based on the team size, capability delivered, and investment focused (cultural).
- Allow for documentation and reporting substitutions to improve agility (agile reporting vs EVM) (cultural and EVM policy).
- Provide training to contracting officers, program managers, and leadership to understand the value and methods associated with agile and modular implementation (cultural).

To address "*Contracts focus on technical requirements instead of contractual process requirements*":

- Separate contract requirements (scope, PoP, and price) from technical requirements (backlog, roadmap, and stories) (cultural).
- Use statement of objectives (SOO) vs statement of work (SOW) to allow the vendor to solve the objectives how they are best suited (cultural).
- Use collaborative tools and libraries so that all content is available to all parties at all times (cultural).
- Use an agile process to manage structure and technical requirements (cultural).

- Establish a clear definition of done for the end of a sprint (code coverage, defect rate, user acceptance) (cultural).
- Use modular contracting to allow for regular investment decisions based on realized value (cultural).
- Streamline acquisition processes to allow for replacing poor performing contractors (cultural).
- Provide training to contracting officers, program managers, and leadership to understand the value and methods associated with agile and modular implementation (cultural).

To address "*Intellectual Property (IP) rights which are often genetically incorporated without considering the layers of technology often applied to a solution*":

- Establish clear and intuitive guidelines on how and when to apply existing clauses (cultural).
- Educate program managers and contracting officers on open source, proprietary, and government funded code (cultural).
- Have standard clause applications for each of the above that must be excepted vs accepted (cultural).

Appendix F.4: Data and Metrics Subgroup Report

Contributing authors: Ben FitzGerald (lead) and Matthias Maier

The Department of Defense (DoD) has long standing methods for capturing data, developing metrics, and reporting program progress, however these practices do more to obfuscate than provide insight when it comes to software and stand in the way of more effective methods. DoD's approach to data and metrics is fundamentally intertwined with its governance and compliance culture, which centers around reporting on individual programs to inform specific decisions by senior leaders and the Congress. Attempts to change DoD's data and metrics methods must therefore also address this culture and, critically, link with other reform efforts including policy, tools for the software development environment, and overall approaches to governance and investment.

Note: in the context of this appendix, data refers to` information associated with the development, maintenance, enhancement, and performance of software systems, not the substantive data that they process or generate.

Pain Points

Multiple, competing, and sometimes conflicting types of acquisition/management data and metrics are used for divergent purposes in the assessment of software in DoD. DoD has long standing practices to collect data on programs: primarily cost, schedule, and performance. These data are imperfect and do not necessarily reflect the health of software in any way but are important, particularly for satisfying existing reporting requirements. These data must be improved and linked with data and metrics focused on assessing the health of software activities. Doing so will potentially cause bureaucratic confusion and competition.

Challenges collecting meaningful data, in a low cost manner, at scale. To the extent that DoD currently collects data on its software activities, it does so through the manual entry of reporting data in separate and disparate reporting/management systems. This approach is prone to errors and incredibly time-consuming and burdensome to program offices. DoD components responsible for developing and maintaining the systems reporting information have few incentives to share such data, as they are often used against them, meaning that the data are hard to capture, include mistakes, and no constituency wants to invest in systems to automate data collection.

Inability to turn data into meaningful analysis and inability to implement decisions or changes to software activities. Even if DoD had clarity on its use of data and the ability to collect those data passively and at scale, it may not be able to meaningfully change the outcomes of its software activities and could become caught in a Cassandra predicament. The culture of decision making, acquisition policy, contracting, formality of requirements, appropriations rules and oversight mean that data driven insights do not naturally translate into improved decision making on DoD software activities.

Desired State

An operational system and culture that makes policy, investment, and program decisions based on insight and analysis developed in a transparent manner from standardized data collected automatically from software development tools.

Obstacles

The Department, in most but not all instances, does not possess the tools or analysts to achieve the desired state. Those are addressable challenges. The bigger obstacle is the culture of high level reporting, driven from Congress and OSD, on individual programs on a period basis, for example congressionally mandated annual Selected Acquisition Reports (SARs), and in turn, Defense Acquisition Executive Summary (DAES) reports that inform OSD quarterly of the same information. This approach means that data are not strategically collected at the level that allows for real insight and longitudinal analysis, instead they are developed at a summary level to minimally meet requirements and avoid further scrutiny. Most importantly, they do not provide the real-time tools to enable a software program manager to manage her program.

While there are few legislative barriers to implementing the desired state, Congressional action may be required to create the right incentives for DoD to generate, capture, and use data in useful ways. Congress should also address its own oversight culture, which can sometimes drive much of the behavior the Congress dislikes.

Ideas for Change

- Congress could establish, via an NDAA provision, new data-driven methods for governance of software development, maintenance, and performance.[20] The new approach should require on demand access to standard data with reviews occurring on a standard calendar, rather than the current approach of manually developed, periodic reports.
- DoD must establish the data sources, methods, and metrics required for better analysis, insight, and subsequent management of software development activities. This action does not require Congressional action but will likely stall without external intervention and may require explicit and specific Congressional requirements to strategically collect, access, and share data for analysis and decision making.
- Key steps for implementation:
    - Identification of existing definitive data sources (e.g., DAVE, FPDS[21]);
    - Establishment of robust data crosswalks to analyze data across systems and use cases;

[20] Congress could build on Secs 911-913 of FY2018 NDAA
[21] Defense Acquisition Visibility Environment (DAVE) https://dave.acq.osd.mil/; Federal Procurement Data System (FPDS) https://www.fpds.gov/

- Identification and mitigation of any significant gaps in existing data, with priority placed on building out functionality from existing applications where possible;
- Establishment of mechanisms to ensure data sharing and transparency (i.e., require all components to share their data);
- Disambiguation of roles and responsibilities, e.g., OSD = policy/governance ≠ program review. Components = execution;
- Linking data and metrics to governance and policy analysis and decision making.

Appendix F.5: Infrastructure Working Group Report

Contributing author: Jeff Boleng (lead)

Despite several years of effort to “move DoD to the cloud,” significant friction still exists for DoD to easily leverage the required compute, storage, and bandwidth infrastructure that the commercial world so readily enjoys. The major obstacle is not at all technical, but is broadly one of accessibility: the ability to specify, contract for, pay for, connect to, secure, and continuously monitor sufficient modern computing infrastructure. Modern computing infrastructure refers primarily to cloud-based computing technologies and stacks. “Cloud-based” does not necessarily presuppose commercial cloud, but could also be on premises or hybrid cloud solutions. Similarly, “computing technologies and stacks” can run the full spectrum from infrastructure, to platform, to function, to software as a Service (IaaS, PaaS, FaaS, SaaS).

Pain Points and Obstacles

How much cloud do I need? Countless developers and IT professionals have wrestled with this question, and often the answer is to “dive in,” move some apps, see what is needed, and then scale and tweak from there. The Department’s culture hampers our ability to even take a “leap of faith” like this. We must be able to precisely size and cost our cloud requirements before ever starting to experiment or prototype. It should become more clear why this analysis paralysis exists as the below pain points are outlined and considered.

How do I buy cloud? Oh, just head on over to FedRAMP, pick an approved provider, sign up and you’re on your way… FedRAMP? Is that a cloud? What about GovCloud, cloud.gov (not the same thing by the way), and MilCloud (is that version 1.0 or 2.0?)? What’s the difference between AWS GovCloud and Azure Government? Can I just sign up with a credit card like a normal private citizen and start hosting my compute and data in the cloud? Sadly, the answer is a definitive and resounding NO! Even if you know which “government-approved” cloud you’re moving to, it’s just not easy to contract for it or buy it.

There is not space here to answer all these rhetorical questions. For a good description of the difficulty of buying cloud, please refer to the DoD Cloud Acquisition Guidebook at https://www.dau.mil/tools/t/DoD-Cloud-Acquisition-Guidebook. Here the Defense Acquisition University (DAU) outlines the multiple activities that need to be accomplished to contract for cloud services. Starting with the dreaded IT Business Case Analysis (BCA), moving on to applying the DoD Cloud Security Requirements Guide (SRG - more on this soon), to getting an Authorization to Operate (ATO), ensuring DISA approves of your Boundary Cloud Access Point (BCAP) and your Cyber Security Service Provider (CCSSP), and lastly to applying the DFARS supplementary rule to your cloud contact. No friction here right?

How do I know my cloud is secure? Easy. FedRAMP pre-evaluates and approves Cloud Service Providers (CSSPs) for Information Impact Levels (IILs) 2, 4, 5, and 6 (don’t ask about levels 1 and 3; apparently we over specified and they aren’t necessary any longer). Whew, now things are making sense… Not so fast, the FedRAMP IILs are for US Government cloud use, but not DoD![22]

[22] Don’t ask… we know DoD is part of the US Government.

We need FedRAMP+ for DoD use, and DISA doesn't evaluate Cloud Service Providers (CSPs), only Cloud Service Offerings (CSOs). Huh? Be sure to go through the DoD Cloud Computing SRG, ensure those extra security controls are in place for FedRAMP+, and you're on your way. Again, not so fast Program Manager (or small business owner)! How are you and your customers going to access the fancy new cloud you just finally got on contract?

How do I access my cloud? The cloud, sort of by definition, implies ease of access, right? The National Institute of Standards and Technology (NIST) definition in SP 800-145 defines cloud computing as "a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction." Well, if you're a DoD user, you need to ensure you've got a BCAP in place between your application/service and your users. It's OK and accurate to immediately envision bottleneck and single point of failure here.[23] Mis-configuring and under-provisioning BCAPs is the norm rather than the exception, so even with all that compute and storage in the cloud that you somehow ran the contracting gauntlet to get, you're going to severely lack adequate bandwidth and likely suffer from significant latency. Friction++.

How do I pay for cloud? The best part of cloud computing is that I can only pay for what I use. A true consumption-based cost model. Just like a utility. Not so for Government and DoD though. The Anti-Deficiency Act doesn't allow us to pay for cloud computing like a utility. A common way around this is to pay a third party contractor to buy the cloud service for us. This results in a situation where we estimate the highest charges we could ever incur in a year, add a bit of padding to that (say 20-30%), pay the third party, and we've paid for our cloud. What happens if we don't use it all up by the end of the year? Nothing (i.e., no refunds). Money spent. The third party contractor makes (quite?) a bit of extra profit for "taking the risk off the government." So much for consumption-based payments.

Desired State

The ability to provision, pay for, consume, access, and monitor cloud computing (compute, storage, and bandwidth) the same way any commercial organization does. It is understood that there are unique DoD security requirements, but that should only affect cloud pricing (say 1.5 to 2 times commercial, worst case), and not any of the other procedures to easily access cloud computing technologies and resources.

Obstacles

Significant obstacles remain to easily leverage commercially equivalent compute, storage, and bandwidth infrastructure. Contracting, security procedures (not necessarily requirements), network access (i.e., a modern technological approach to BCAP), and billing all loom large. The most important of these is the DoD's inability to contract and pay for cloud computing on a consumption basis.

[23] There are better ways to do this, like zero trust networks. The commercial world has some really good examples and architectures that don't require this man-in-the-middle attack called a BCAP which actually breaks end-to-end encryption by design…

Ideas for Change

Establish a DoD enterprise ability to procure, provision, pay for, and use cloud that is no different from the commercial entry points for cloud computing. The Joint Enterprise Defense Infrastructure (JEDI) Cloud initiative is a bold attempt at this solution and should be awarded. Cloud.gov (which is ironically hosted in GovCloud) is another promising program that is already very straightforward to provision and buy, but is limited to IIL 2 data and applications. The objective cloud procurement and billing contract must include the ability to truly pay for consumption of cloud services and not be artificially limited by the Anti-Deficiency Act. Modern software demands the ability to consume and pay for cloud services just as we do any other utility.

In addition to this, DoD should establish a common, enterprise ability to develop software solutions in the "easy-to-acquire-and-provision" cloud that is fully accredited by design of the process, tools, and pipeline. Said another way, DoD should stop the security accreditation of individual applications, but *should instead invest in accrediting the ability to produce software*. The pipeline, automated tooling, procedures, and operational monitoring and auditing of software should be the focus and target of security accreditation, not each individual application and version of an operating system or application.

Another essential and necessary, though not sufficient, change that must occur is to adopt modern commercial approaches to software and system security in the cloud that does NOT involve BCAPs, Internet Access (choke) Points (IAPs), or CSSPs that cannot be performed entirely by trusted commercial entities. DoD must adopt modern cloud security approaches such as zero trust networks[24], micro-segmentation, and eliminate the perimeter approach to network security and trust that is based on assigned IP address or network connection point. Perimeter-based security cannot scale to accommodate the bandwidth, traffic, and latency demands of modern cloud access, applications, and services. Furthermore, it is a failed architectural practice that has proven to be readily exploitable by adversaries and is especially vulnerable to insider threats.

[24] https://www.oreilly.com/library/view/zero-trust-networks/9781491962183/ch01.html

Appendix F.6: Requirements Subgroup Report

Contributing authors: Fred Gregory (lead), Philomena Zimmerman, Jeff Boleng, Margaret Palmieri, Jennifer Edgin, Owen Seely, Victoria Cuff, and Donald Johnson

The Department of Defense (DoD) in 2003 institutionalized the identification and validation of requirements via the Joint Capability Integration and Development System (JCIDS). Created to support the statutory responsibility of the Joint Requirements Oversight Council (JROC), it is one of three processes (Acquisition, Requirements, and Funding) that support the Defense Acquisition System (DAS). Considered revolutionary in its design, moving DoD from a threat-based to a capability-based model, it has begun to show its age in today's era of software-intensive systems intending to leverage agile software practices. These evolving agile practices upend traditional industrial-age process attempts to credibly and accurately predict a future 15-20 years away, necessitating unimaginable precision and foresight upfront in support to capability development. The requirement process, writ large, must adapt to support delivering capabilities at the speed of relevance; processes, cultures, and expectations of the Service and Joint Force requirement communities.

Pain Points

A byproduct of top-level requirement flow down is rigidity and over specificity at the derived requirements level, that greatly hinders agile software design. Capability validated by the JROC does not proscribe requirement allocation to either hardware or software solutions. However, the resulting flowdown of derived requirements incorporated into the source selection/contract award and the subsequent allocation of these between hardware and software by the prime can ultimately discourage software design flexibility. The decisions, often made years before software coding even begins, locks the prime and the government into a proscribed path that often does not produce the desired warfighter capability within the needed time frame. Preserving software design flexibility must be a key component throughout the requirements validation process. "Requirers" will need to learn to settle for "less" not "more" at capability need inception.

Too often exquisite requirements, intended to be 100 percent correct, are levied on a system that in turn drives extensive complex software requirements and design, affecting development, integration, and system test. Today's requirements process more closely mimics the "big-bang" theory often vilified by industry, government, and Congress. As the warfighting community loses faith in the acquisition community's ability to meet their commitments through timely incremental improvements, the temptation to "gold-plate" a requirement becomes more prevalent. Likewise, as the acquisition community is forced to defend shifting warfighter priorities in budget deliberations and Congressional engagements, the temptation to "lock requirements down early" permeates acquisition strategies. With both of these choices in play, exquisite requirements must be described perfectly at capability inception in order to maintain a low-risk acquisition program - obviously an impossible outcome.

Data sets are siloed within programs - a common Law of Requirements is that programs of record (PoR) try to avoid dependencies with other PoRs. By tying SW to a PoR, it becomes *nearly impossible to transfer that code across systems and data environments*. Data “lakes,” “pools,” and “ponds” will be the foundation for future weapon system data repositories, and the requirements process must be flexible enough to accommodate this new archetype. Breaking from the past mold of tying software code to a program of record and a specific data environment frees the program manager from the arduous task of integrating seams across multiple PORs.

Example. The Navy operates forward at sea and on-shore at maritime operations centers (MOCs). Command and control between sea and shore is a key aspect of how they fight—they need shared battlespace awareness at aligned actions across distributed units at best. However, the systems afloat and ashore are not always the same because ships need systems that are hardened for combat at sea. If a new algorithm can help manage supply and logistics on the cloud ashore, it may not run the same at sea because different system exists afloat. Extrapolating across Services, the USAF writes an algorithm to optimize F-16 maintenance, however it is highly unlikely that the Navy can pick it up and apply it to F-18s. This depends on the vertical integration of the algorithm, data, and system (PoR).

Desired state. Go from Sailor (Airman, Rifleman, etc)-stated need to software delivery in their hands within days to support future conflicts. This necessitates a process for concept/requirements determination/setting that takes advantage of the agility in software development and software products to increase the agility and modifiability in our systems. Requirements flow down must also maintain a broad-based approach into the lowest levels of design. We also note that one of the overarching agile principles is that “increments are small.” Fast requirements, fast deployments and fast test cycles for usefulness are tough to accomplish with huge, monolithic software projects. Start small, stay small! Finally, recognizing that documenting and contracting for a moving target is not easy but must be done.

Obstacles. Breaking the tyranny of siloed PoRs will require a concerted effort across the Department, Combat Support Agencies, and will require Congressional engagement and support. Considerable cultural barriers must also be overcome as the algorithms themselves become capability, and the methods used to document, validate, and maintain currency enter the mainstream. Complexity and dependencies among multiple elements prevent widespread usage of Family-of-Systems (FoS) and System-of-Systems (SoS) requirement documents. Government requirements and acquisition communities take on extra oversight burden when they take a FoS or SoS approach because they have to manage all the pieces coming together effectively. Lastly, current statutory guidance does not promote, encourage, or reward the use of agile software development practices or environments.

Ideas for Change

- The Joint Staff should consider revising JCIDS guidance to separate functionality that needs high variability from the functionality that deemed “more stable” (e.g., types of signals to analyze vs. allowable space for the antenna). Then implement a “software box” approach for each, one in which the contours of the box are shaped by the functionality variability

- OSD should consider identifying automated software generation areas that can apply to specific domains
- The Joint Staff should consider revising JCIDS guidance to document stable concepts, not speculative ideas.
    - Specifying needed capabilities is important up front, however it must be acknowledged that initial software requirements need to be "just barely good enough" for the situation at hand or, in other words, "document late"
    - Acknowledge that software requirement documents will iterate, iterate, iterate. JCIDS must change from a "one-pass" mentality to a "first of many" model that is inherently agile delegating approval to the lowest possible level

- DoD should consider instituting a distributed model-based approach to requirements development extended across the enterprise
    - The model should be used to develop result-based metrics for requirement evaluation

- The Joint Staff should consider revising JCIDS guidance to focus on user needs, bypassing the JCIDS process as needed to facilitate rapid software development. Guidance should specifically account for user communities (e.g., Tactical Action Officer (TAO), Maritime Operations Center (MOC) director) that do not have one specific PoR assigned to them, but use multiple systems and data from those systems to be effective
- OSD and the Joint Staff should consider creating "umbrella" software programs around "roles" (e.g., USAF Kessel Run)

Appendix F.7: Security Accreditation/Certification Subgroup Report

Contributing authors: Leo Garciga (lead), Tom Morton, and Ana Kreiensieck

The Department's current Security Certification and Accreditation (C&A) process is a complicated and time-consuming process that is measured in months and years. The process is typically seen as a serial process that occurs after development with a checklist mentality. While this fits with a waterfall approach to development, the Department is changing to an agile, DevSecOps approach. The overall security paradigm must change from one where updates to software happen optimistically on a yearly basis to one where software is updated weekly or daily in response to emerging threats and this is recognized as more secure than the slow, static process. Additionally, we must strive to accredit the process, tools, and platforms to allow and enable Continuous ATO when software changes meet the required thresholds.

Pain Points

Complex, time-consuming, and misapplied process. Although developing and operating software securely is a primary concern, the means to achieve and demonstrate security is overly complex and hampered by inconsistent and outdated/misapplied policy and implementation practices (e.g., overlaying historical DoD Information Assurance Certification and Accreditation Process (DIACAP) process over Risk Management Framework (RMF) controls for individual pieces of software versus system accreditation). The sense is that the Certification and Accreditation (C&A) process is primarily a "check-the-box" documentary process, adds little value to the overall security of the system, and is likely to overlook flaws in the design, implementation, and the environment in which the software operates.

No way to calculate total costs of C&A process. The Department needs to be able to calculate the true and component costs for implementing the RMF and C&A in order to identify inefficiencies, duplicative capabilities, and redundant or overlapping security products and services that are being acquired or developed. Absent a set of metrics it is difficult to prioritize risk areas, investments, and evaluating risk reduction and return on investment.

Lack of top-down security requirements. The Department has not decomposed security requirements from an enterprise level to a mission level to a functional implementation level. Programs waste resources implementing security controls that should be inherited.

Lack of automation. The C&A process is predominantly a manual process which makes it a very low process. Programs must plan in terms of months and years to get a product through the security accreditation process. This slow process does not provide the warfighter the timely, modern solutions that are needed.

Desired State

Accredit the process, not the product. Done correctly, security is applied from the beginning of software development using automated tools. Before transitioning into operations, an Authorizing Official (AO) reviews the process under which the software was developed and accepts the risk as determined from various scans and tests. The AO signs a Continuous ATO so that as long as the process remains intact and is continuously operationally monitored, the subsequent software releases are accredited.

Obstacles

Two primary obstacles are culture change and workforce skills. The current security culture is that security is a checkbox activity at the end of the development process. As RMF is implemented, this is beginning to change the culture of security from compliance to continuous risk assessment. However, the process is still very manual. The culture change needs to include using automation to speed up risk assessment and continuous risk monitoring of operational software.

The other obstacle is the security and accreditation workforce skill set. While tools can provide reports and speed up security activities like scans and code analysis, it takes a particular skill set to understand those inputs and recommend or make at-risk decisions. The current security workforce must be trained in these new skills.

Ideas for Change

Embrace DevSecOps. The Department should embrace DevSecOps (not just DevOps) and provide the necessary resources to develop the common software components and automation to assemble, test, accredit, and operate software systems. DevSecOps also includes policy-supported processes, certified libraries, tools, and an operational platform (with appropriately instrumented run-time software), and a toolchain reference to implementation to produce “born secure” software.

Automate, Automate, Automate! The Department needs to provide automated tools and services needed to integrated continuous monitoring with the development life cycle, enable continuous assessment and accreditation, and delegate decision making at the lowest level possible. Examples of automation are using static code analysis during the “build” stage, running automated unit tests, functional test, regression tests, integration tests, and resiliency/performance tests during the “test” stage, using dynamic code analysis, fuzzing scans, running container security scans, STIG compliance scans, and 508 compliance scans during the “secure” stage, and running continuous monitoring tools and ensuring logs are being pushed to the appropriate entity during the “monitoring” and “operational” stages.

Define top-down implementation requirements. The Department needs to ensure that each Joint Capability Area (JCA) flows-down its strategy, best practices, and implementation requirements/guidance for security and accreditation to allow the Component responsible for implementing the software to appropriately tailor RMF and plan the development, accreditation, and operation of the software. Furthermore, each JCA should endeavor to clearly state its risk profile and tolerance so that the RMF can be applied effectively and appropriately mitigate identified risks.

Education is necessary at all levels. As security is “baked in” to software during the development process, people must be educated about what that means as different tools look at different security aspects. They must also be educated in what it means to bring different security reports together and make a risk decision, both during development, and continuously during operations.

Culturally, people must learn to appreciate that speed helps increase security. Security is improved when changes and updates can be made quickly to an application. Using automation, software can be reviewed and updated quickly. The AO must also be able to review documentation and make a risk decision quickly and make that decision on the process and not the product and document it in a Continuous ATO.

Appendix F.8: Sustainment and Modernization Subgroup Report

Contributing authors: Kenneth Watson (lead), Stephen Michaluk, and Bernard Reger. Additional advice / assistance from SEI

Improving the materiel readiness of our fielded weapon systems and equipment is an imperative across the Department in accordance with the new National Defense Strategy.[25] The time is now to shift from our traditional, hardware-centric focus and identify what core[26] means for software intensive weapon systems and associated software engineering capabilities. Software is a foundational building material for the engineering of systems, enabling almost 100 percent of the integrated functionality of cyber-physical systems, especially mission- and safety-critical software-reliant systems. More simply, these systems cannot function without software.

For fielded weapon systems and military equipment, software life-cycle activities follow somewhat predictable cycles of corrective, perfective, adaptive, and preventative modifications while major modifications drive new periods of development. Software development activities, even those following agile methods, encounter a phase where the program transitions from adding new features to supporting and sustaining day-to-day use and operations. At that point, development changes and signals a move to "sustainers" within the organic industrial base. Therefore, sustainment may be defined as the sum of all actions and activities necessary to support a weapon system or military equipment after it has been fielded.

Prioritizing the transition to software sustainment during requirements and engineering development is critical to timely, effective, and affordable sustainment, regardless of how software engineering organizations are structured and resourced. Software sustainment organizations must be engaged and embedded at the earliest design stages to ensure we can keep pace with new capabilities as systems become operational. Lastly, access to software source code, emphasizing an early focus on designing for sustainment, and investment into establishing and modernizing system integration laboratories, are just a few of the challenges faced by the DoD software enterprise.

Pain Points

Applying a hardware maintenance mindset to software hinders DoD's ability to better leverage the organic software engineering infrastructure. DoD maintenance policies and maintenance-related Congressional statutes have traditionally been optimized for hardware and are difficult to change due to long standing policies, practices, inertia, and incentives. The goal of hardware maintenance is to repair and restore form, fit, and function. This mindset does not align well with the ever evolving nature of software. The scope of software engineering for sustainment mitigates defects and vulnerabilities, fact-of-life interface changes, and add new enhancements. Software is never done and any time it is "touched," it triggers the software engineering development life

[25] "Summary of the 2018 National Defense Strategy" (Washington, DC: Department of Defense, 2018), https://dod.defense.gov/Portals/1/Documents/pubs/2018-National-Defense-Strategy-Summary.pdf.

[26] As defined in 10 USC 2464, *Core logistics capabilities.*

cycle which produces a new configuration. Therefore, any system that is dependent on software to remain operational, is always in a state of continuous engineering during sustainment (or O&S phase of the life cycle).

DoD's acquisition process is not emphasizing an upfront focus on design for software sustainment and a seamless transition to organic sustainment. It is critical that software be designed to be more affordably sustained with high assurance and the ability to integrate changes and enhancements more rapidly to provide a continual operational capability to the warfighter. Moreover, software must be decoupled from hardware to the greatest extent possible in order to enable leveraging rapid and continuous hardware improvements. We need to place increased emphasis in acquisition on designing in software sustainability with a consistent emphasis on how DoD contracts for software as well as the span of requirements, architecture, design, development, and test. Additionally, this includes making provisions for timely access to the necessary range of software technical data to enable timely and effective organic software engineering and rapid re-hosting. It is essential that DoD and industry work collaboratively to meet the increasing software sustainment demand.

Public Private Partnerships (PPPs) provide one means to leverage DoD and industry capabilities as a team to deliver warfighter capability. However, PPPs and other options are not being considered up front and leveraged across DoD as an inherent element of the acquisition and engineering strategy of programs. This team strategy may facilitate mutual access to the technical data inherent in executing the software development life cycle.

Limited visibility of DoD organic software engineering infrastructure, capabilities, workload, and resources. Title 10 USC 2464 establishes a key imperative for DoD to establish core Government Owned Government Operated (GOGO) capabilities as a ready and controlled source of technical competence and resources for national security. DoD's focus has traditionally been on hardware and therefore there has seen significant Service and DoD enterprise focus on hardware GOGO capabilities and infrastructure for core. However, there has been significantly less upfront acquisition focus and visibility on what core means for software intensive systems and the associated GOGO software engineering capability. For the traditional DoD hardware-centric model, core capability is based on individual weapon systems or platforms at the depot level. All systems operate interdependently in a net-centric environment, where force structure and execution of mission capabilities are products of a system-of-systems capability. In a software intensive environment "Go to War" analysis of what core means as it relates to software requires more strategic thinking about core than just focusing on individual weapon systems or platforms (aircraft, ship, tank, etc.) as hardware. The hardware-centric focus on weapon systems likely underestimates the scope and magnitude of what should be considered a core requirement in a software intensive systems operational environment.

Desired State. Require government integrated software sustainment participation from the very beginning of development activities.

Ideas for Change

- Title 10 USC 2460 should be revised to replace the term software maintenance with the term software sustainment and a definition that is consistent with a continuous engineering approach across the life cycle.
- DoD should establish a capability for visibility into the size and composition of DoD's software sustainment portfolio, demographics, and infrastructure to better inform enterprise investment and program decisions.
- A DoD working group should be established to leverage on-going individual Service efforts and create a DoD contracting and acquisition guide for software and software sustainment patterned after the approach that led to the creation of the DoD Open Systems Architecture Contracting Guide.
- Acquisition Strategy, RFP/Evaluation Criteria, and Systems Engineering Plan should address software sustainability, re-hosting, and transition to sustainment as an acquisition priority. The engineering strategy and plan should engage software sustainment engineers upfront and co-locates government software sustainment engineers on the contractor software development teams to enable effectively and timely transition to an organic sustainment capability.
- The definition of "core capabilities" in 10 USC 2464 should be revisited in light of warfighter dependence on software intensive systems to determine the scope of DoD's core organic software engineering capability, and we should engage with Congress on the proposed revision to clarify the intent and extent of key terminology used in the current statute.
- DoD should revise industrial base policy to include software and DoD's organic software engineering capabilities and infrastructure. Start enterprise planning and investment to establish and modernize organic System Integration Labs (SILs), software engineering environments, and technical infrastructure; invest in R&D to advance organic software engineering infrastructure capabilities.

Appendix F.9: Test and Evaluation Subgroup Report

Contributing authors: Amy Henninger and Greg Zacharias

The fundamental purpose of DoD test and evaluation (T&E) is to provide knowledge that helps decision makers manage the risk involved in developing, producing, operating, and sustaining systems and capabilities. While colloquially referred to as a single construct, T&E is composed of two distinct functions: obtaining the data and assessing the data. This distinction is important because the T&E community will report "pain points" in both functions. There are also two major types of test: Developmental Test (DT) and Operational Test (OT). DT, by nature, is "experimental," performed on behalf of the Program Management Office (PMO), supporting a formative evaluation and identifying design elements that will drive mission-critical capability to inform the evolution of component and system design. OT is "evaluative," performed by and on behalf of the warfighter, supporting a summative evaluation of system capabilities to support warfighting missions across the operational envelope.

Because T&E has historically occurred toward the end of, often, a long and costly acquisition process (e.g., requirements, design, and development), it can be perceived as simply adding time and cost to an already late and over-budget effort; PMOs therefore can view this "last step" T&E as simply making the situation worse. And if T&E finds a system substantially defective, necessitating expensive re-engineering of the design late in developing, it adds to the perception that T&E simply adds cost and time to project execution. A continuous iterative T&E model is clearly called for, occurring alongside design and development, where T&E can both; catch defects early so they can be solved quickly and cheaply and inform/shape system requirements based on early feedback from the warfighter. Experience shows that active, early involvement by independent testers—combined with a PMO who responds to the independent testers' advice—makes a positive difference to program outcomes. We have seen this in modern iterative approaches, such as agile development, applied effectively in DoD, especially in Major Automated Information Systems (MAIS).[27] Taken together, these observations point to the need to move away from what can be a linear waterfall process segregated by siloes, to a more iterative and collaborative model that fuses all development, test, processes, tools, and information to enable the continuous delivery of tested capability. T&E can then be viewed as saving time/cost in development, instead of adding time/cost.

Pain Points and Obstacles

DoD lacks the enterprise digital infrastructure needed to test the broad spectrum of software types and across the span of T&E to support developmental efficiency (in DT) and operational effectiveness (in OT). Digital models of test articles (e.g., "Digital Twins") are not always available and not built to common standards. T&E environments, including threat surrogates or models, are often program-focused and funded, with short-term development goals and narrowly-scoped capabilities defined by the program. Building (and re-building) representative T&E environments is time and cost prohibitive for individual programs and results in duplicative infrastructure investments across DoD. Moreover, current T&E practices in the Services, including those

[27] FY16 DOT&E Annual Report.

focused on software-intensive systems, do not adequately test systems in Joint and Coalition environments, nor do they consistently use appropriate risk-based, mission-focused testing.

DoD lacks the enterprise data management and analytics capability needed to support the evaluation of test data in accordance with the pace of modern iterative software methods. As data required to make informed acquisition decisions continues to grow due to higher resolution measurements, higher acquisition rates, and other additional requirements for software intensive systems (e.g., interdependency, need to operate in system-of-systems, family-of-systems, Joint, and Coalition environments), the need for a T&E infrastructure to collect, aggregate, and analyze this data must likewise evolve to keep pace. More timely data fusion will require improvements in data management techniques, access speeds, data access policies, data verification techniques, and the availability of more intelligent and agile tools. Without this infrastructure, and within the current paradigm, we are failing to adequately gather and analyze these highly diverse and complex datasets, which leads to invalid assessments of acquisition program progress and system performance, undercuts mission readiness, and places warfighters at risk. This gap becomes an even more prominent choke point in an iterative cycle. Thus, even if we mitigate the first pain point with modernized realistic test environments, and had the capability to collect the appropriate mix/quantity of data in testing, we would still not have the analytics horsepower to turn around an assessment to support the pace of an Agile/DevSecOps iterative cycle.

DoD lacks the resources needed to adequately emulate advanced cyber adversaries, to support fielding of trusted, survivable, and resilient software-intensive defense systems. Various oversight entities (e.g., NDAAs and GAO Reports) have acknowledged this gap, and past DOT&E Annual Reports have documented a significant number of adverse cyber findings in OT that should not require an operational environment to discover. While the gap exists now (in the absence of modern software methods), it will become an even more prominent choke point in a rapid development and operational fielding paradigm. We do not have the advanced cyber test resources (manpower, methods, and environment) to support a true Agile/DevSecOps approach to developing, testing, and fielding the broad range of software-intensive systems needed by DoD now and in the future, in an environment increasingly populated by advanced cyber adversaries.

DoD lacks a modern software intellectual property (IP) strategy to support T&E in a rapid software development and fielding environment. Overcoming this pain point is critical to overcoming all of the three previously described pain points. Specifically, none of the previously described pain points is fully achievable without sufficient access to necessary technical data associated with the software deliverables. Software acquisition processes are and will continue to be suboptimal (with respect to time and risk) without access to relevant technical data and this gap will become an even more prominent choke point in an Agile/DevSecOps-based paradigm without that access. A modern software IP strategy must include access to software environments (e.g., source code, build tools, test scripts, and cybersecurity artifacts/risk assessments) so tests are repeatable, extendable, and reusable. This strategy will also have to strike a balance with the IP rights of the innovator (usually industry) to ensure continued engagement of DoD with leading-edge technology organizations.

A modern software IP strategy would support the three previously described pain points via:

- Enhance our ability to operationalize the concept of “digital twins,” with sufficient access to the source code of a given system (balancing DoD and innovator IP rights), so as to be able to adequately represent that system.
- Support the instrumentation of software-intensive systems as needed during testing.
- Support cyber vulnerability assessments and the assignment of risks to residual vulnerabilities, via access to system data (e.g., code and technical data).

Desired State

While DoD does a fair amount of “integrated testing” now (across DT and OT), that is not the same as “integrating T&E with the Voice of the End User continuously and alongside software development.”[28] T&E must strive for continuous software testing, automated and integrated into the development cycle to the fullest extent possible, across the entirety of DoD’s software portfolio. The qualifier, “fullest extent possible” is important, as many experts have acknowledged that no single “one size fits all” approach will work best across the entire DoD software portfolio all of the time.[29,30] In this envisioned state, independent testers would work alongside developers and operators to help software development programs succeed and deliver capability at the speed of need. T&E would no longer be perceived as “slowing things down” or “costing money post-development” because it occurs toward the end of a highly linear and inefficient process, but would instead be associated with saving time and money during development. This vision, applied across the entire DoD software portfolio (i.e., beyond just IT or MAIS) requires the right kinds of tools, architectures and standards (see first three pain points), access to the right kind of data (see second and fourth pain points), and an ability to partner with and work alongside the developer, while yet maintaining independence and objectivity in our assessments.

Ideas for Change

Build the enterprise-level digital infrastructure needed to streamline software development and testing across the full DoD software portfolio. Beyond the DevSecOps platform (or Digital Technology concept), DoD requires a digital *engineering* infrastructure to streamline integration and testing. This suggests that the DevSecOps platform must be made available to all DoD software developers and:

- Integrated with (systems-level) model-based/digital engineering infrastructure, including digital twin(s),
- Integrated with existing T&E infrastructure (e.g., open-air ranges, labs, and other test facilities),
- Integrated with comprehensive tactical/mission-level infrastructure, and
- Available to others who could benefit (e.g., analysis, training, and planning).

[28] Steven Hutchison, “Test and Evaluation for Agile Information Technologies,” *ITEA Journal* 31(2010): 461.
[29] 2018 Defense Science Board Task Force on Design and Acquisition of Software for Defense Systems.
[30] Boehm and Turner, 2009. Balancing Agility and Discipline: A Guide for the Perplexed. Addison-Wesley. Boston, MA.

Even with this kind of complete testing infrastructure providing the capability to collect the appropriate mix/quantity of data in testing, we would still not have the analytics horsepower to turn around an assessment sufficiently rapidly to support the pace of an Agile/DevSecOps iterative cycle. We must develop the enterprise knowledge management and data analytics capability for rapid analysis/presentation of technical data to support deployment decisions at each iterative cycle.

Finally, to advance our cyber test resources such that we can achieve overmatch to our most capable adversaries while yet supporting the pace of the modern software development, DoD should expand DOT&E's current capability to obtain state-of-the-art cyber capabilities on a fee-for-service basis. This provides a straightforward way to acquire skilled cyber personnel from leading institutions (e.g., academia, university affiliated or federally funded research and development centers), to help the DoD to keep pace with advanced cyber adversaries.

Appendix F.10: Workforce Subgroup Report

Contributing authors: Maj Justin Ellsworth (lead), Sean Brady, and Kevin Carter

DoD's workforce (civilian, military, and supporting contractor personnel) is our most valuable resource. The workforce's capacity to apply modern technology and software practices to meet the mission is the only way we can remain relevant in increasingly technical fighting domains, especially against our sophisticated peers, Russia and China.

Improved management of the Department's software acquisition talent will also drive success across the other subgroups and sections of this report. Policies, processes, and bureaucratic practices are never a sufficient substitute for competence.

The Department's challenges are well documented and well known by the software acquisition and engineering professionals who suffer most from the accrued technology, cultural, and leadership debt. The Workforce Subgroup identified prevalent pain points, but focused on providing concrete and actionable solutions for improving the recruitment, retention, development, and engagement of the workforce.

Pain Points

The Department's reputation as an employer is a weakness rather than a strength. Candidates base their employment decision on a variety of factors, but the organization's reputation and day-to-day work are chief among their considerations. The demand, and competition with the private sector, for an experienced and qualified workforce, is increasing as threats to our data security become more sophisticated. DoD has a reputation as an antiquated employer that rewards time in grade rather than competence and most often outsources its technical execution. Technical employees often serve as oversight or move away from "hands-on-keyboard" as they advance in their careers; no longer contributing to creative or innovative execution.

The Department does not adequately understand which competencies and skill sets are possessed and needed within its software acquisition and engineering workforce. Without the ability to distinguish the workforce, DoD cannot effectively drive human capital initiatives. Furthermore, there is no enterprise-wide talent management system to manage the workforce (e.g., geographically or by skills), which leads to bureaucratic silos and the inability to leverage the Total Force.

The Department has not prioritized a comprehensive recruiting strategy or campaign targeting civilians (90 percent of the acquisition workforce) for technical positions. When candidates do apply, they face an "overly complex and lengthy hiring process (that) frequently results in the Government losing potential employees to private sector organizations with more streamlined hiring processes," according to the President's Management Agenda.[31]

[31] "President's Management Agenda: Modernizing Government for the 21st Century," (Washington, DC: Office of Management and Budget, April 2018), 20, https://www.whitehouse.gov/omb/management/pma/.

There is no comprehensive training or development program that prepares the software acquisition and technical workforce to adequately deploy modern development tools and methodologies within our dynamic environments. Hiring top technical talent into the Department will never be a silver bullet. The Department also needs to consider how to equip, reward, promote, and empower its existing workforce.

The Department is unable to leverage modern tools that are common in the private sector and our personal lives (e.g., cloud storage and collaborative software) due to bureaucratic barriers. Top talent expects access to these tools to meet mission demands, and their absence may discourage qualified candidates from applying or staying. Although the Department has pockets of innovation and entrepreneurship within rapid fielding offices across the Services, this culture has not scaled to the larger acquisition programs and offices. Long-cycle times, bureaucratic silos, and information-hoarding prevail.

Desired State

The Department requires a workforce capable of acquiring, building, and delivering software and technology in real time, as threats and demands emerge. This workforce should resemble successful technology companies that must move quickly to meet market challenges. They do so by promoting an agile culture, celebrating innovation, learning from calculated failures, and valuing people over process.

The Department's workforce embraced commercial best practices for the rapid recruitment of talented professionals. Once onboarded quickly, they will use modern tools and continuously learn in state-of-the-art training environments, bringing in the best from industry and academia, while pursuing private-public exchange programs to broaden their skill sets.

Obstacles

The bureaucratic culture of the Department creates significant barriers compared to a commercial sector ecosystem that moves at the speed of relevance. These barriers are now ingrained within the institution, perpetuating a risk-averse environment that represents the most significant obstacle to reform. While there are minor legislative solutions to achieving the desired state, we believe that the Department has the necessary authorities and flexibilities, but has shown lack of impetus to move to the modern era of talent management.

While small pockets of expertise and progress exist, the Department as a whole lacks sufficient understanding of current software development practices and talent management models that support them. Studies on the workforce dating back 35 years that show "limited evidence these different efforts had any lasting impact or resulted in meaningful outcomes."[32]

[32] McLendon, Michael H.; Shull, Forrest; Miller, Christopher, "DoD's Software Sustainment Ecosystem: Needed Skill Sets," (Naval Postgraduate School, Monterey, California, April 30, 2018).

Ideas for Change

Foundational. Taking into account history and the significant challenges with changing the culture in a bureaucracy, the Department should *empower a small cadre of Highly Qualified Experts and innovative Department employees to execute changes* from this report. This cadre is empowered with the authority to create, eliminate, and change policies within the Department for organizations beyond themselves. If needed, create a software acquisition workforce fund similar to the existing Defense Acquisition Workforce Development Fund (DAWDF). As called out by the Defense Science Board, the purpose of this fund will be to hire and train a cadre of modern software acquisition experts. This fund should also be used to provide Agile, Tech, and DevSecOps coaches in Program Offices to support transformations, adoption of modern software practice and sharing lessons across the enterprise.[33]

Workforce Foundations. The Department must develop a core occupational series based on current core competencies and skills for software acquisition and engineering. This occupational series should encompass all workforce roles required for modern software development and acquisition - engineers, designers, product managers, etc. Additionally, the Department should create a unique identifier or endorsement of qualified (experience & training) individuals who are capable of serving on an acquisition for software. This includes the development of a modern talent marketplace (and associated knowledge and skill tags/badges) to track these individuals. The competencies for this series should be flexible enough to evolve alongside technology, something that has constrained the 2110 IT Series.

Contractor Reforms. Defense contractors develop the majority of software in the Department. The Department should incentivize defense contractors that demonstrate modern software methodologies; this may take the form of software factory demonstrations and rapid software delivery challenges when evaluating proposals. Additional consideration should be given to contractors with demonstrated excellence creating commercially successful software.

Recruitment and Hiring. The Department must overhaul its recruiting and hiring process to use simple position titles and descriptions, educate hiring managers to leverage all hiring authorities, engage subject-matter experts as reviewers, and streamline the onboarding process to take weeks instead of months. The Department needs to embrace private-sector hiring methods to attract and onboard top talent from non-traditional backgrounds (e.g., hackers and entrepreneurs). Too often, these types of candidates are passed over or require special authorities to join the Department, due to lack of education or regular pay stubs. Furthermore, the Department must develop a strategic recruitment program that targets civilians, similar to its recruitment strategy for military members. This includes prioritizing experience and skills over cookie-cutter commercial certifications or educational credentials.

Development, Advancement, Engagement, and Retention. The Department must pilot development programs that provide comprehensive training for all software acquisition professionals, developers, and associated functions. Programs should be built in partnership with

[33] Design and Acquisition of Software for Defense Systems," Defense Science Board, Feb. 2018, https://www.acq.osd.mil/dsb/reports.htm.

academia and industry, leveraging commercial training solutions rather than custom and expensive Federal solutions. This will include continuing education courses to help the workforce stay current and ensure technical literacy across the acquisition workforce. The Department must emphasize promoting and rewarding those that have proven both commitment and technical competence. Continually looking outside the Department is demoralizing and insulting to existing professionals that demonstrate innovation, excellence, and the ability to deliver already. The Department should incentivize and provide software practitioners access to modern engagement and collaboration platforms to connect, share their skills and knowledge, and develop solutions leveraging the full enterprise.

Finally, the Department should encourage greater private-public sector fluidity within its workforce. Federal employees who come from the private sector bring with them best practices, modern methodologies, and exposure to new technologies. Federal employees who leave bring their understanding of our unique mission and constraints, helping the private sector develop offerings and services that meet our needs.

Appendix G: Analysis the Old-Fashioned Way: A Look at Past DoD Software Projects

The Department has been building and buying software for decades. The study's initial idea was to take a cutting edge machine learning tool, hook it up to the Department's databases, and do an analysis across all of the plentiful software data collected over the years.

Unfortunately, initial attempts at analysis quickly led to the realization that the Department had never strategically collected data on its software. The data that have been collected cover only a subset of the systems the Department acquires and are typically collected by hand, with all the potential for erroneous or missing values that that implies. The granularity at which data are collected also does not typically support insight into specific questions of acquisition performance. Without massive data calls, enormous amounts of PDF scanning, and an impossible number of non-disclosure agreements, a comprehensive analysis would not be possible.

Instead, the SWAP members broke the analysis into two main efforts:

1. Analysis of the available data in order to test the board's hypotheses as they evolve. Subject Matter Experts who are familiar with the existing data and its constraints explored the available data in search of insights that would confirm or refute the board's hypotheses about DoD software acquisition performance. These results are described in this appendix.

2. Application of cutting edge machine learning and other modern analytical techniques to datasets from outside of DoD, to support reasoning about the type of insights that could be gained and reported, if the Department had access to more comprehensive data about its software. These results are described in Appendix D.

G.1 Data Used in This Analysis

The focus of this study is on software-intensive programs—and the specific software scope within these programs—presenting top-level insights into software acquisition performance. We focused our analysis on a few major data sources collected by the Department, which can provide insight on these issues.

The data in our first source are known as Software Resources Data Reports (SRDRs). The SRDR data were selected for use because they are specifically focused on the software activities of DoD acquisition programs. The SRDR is a contract data deliverable that formalizes the reporting of software metrics data and is the primary source of data on software projects and their performance. The SRDR reports are provided at the project level or subsystem level, not at the DoD Acquisition Program level. The data points included in the analyses reported here are representative of software builds, increments, or releases. In many cases, there are multiple data points in the set that represent different subsystems or projects from the same program.

The SRDR applies to all major contracts and subcontracts, regardless of contract type, for contractors developing or producing software elements that meet specific criteria[34] and with a projected software effort greater than $20M.

SRDR reports are designed to record both the estimates and actual results of new software development efforts or upgrades, with the goal of supporting cost estimation. The reports collect many characteristics about software activities in both structured and unstructured formats. The primary data analyzed in our work were size, effort, and schedule. Notably absent from the SRDRs are any data about quality. Defect data have been optional until recently and hence were not reported.

Other data sources used to explore some of the assumptions and recommendations of the DIB are the IPMR (Integrated Program Management Report) and SAR (Selected Acquisition Report) datasets. Programs in these datasets fall into the category of Major Defense Acquisition Programs (MDAPs). These datasets include:

1. Software development effort measured in labor hours, software size, and development activity duration metrics delivered as mandated respective to contractual agreements.

2. Software development performance as identified within each contract report. However, each contract contained common elements supporting both software and non-software activity on contracts. These were treated in proportion to the weight of software activity cost on contract. These reports contain data for measuring contractor’s cost compared to budget baselines on Department acquisition contracts as well as projections of cost at completion.

3. Planned and executed schedule milestone dates reported to the Department at the aggregate program level as required by acquisition policy. This information is included as a part of a comprehensive summary of total program cost, schedule, and unit cost breach information.

These software development effort metrics, contract performance, and program level schedule data represent the best source of product development, contract cost, and schedule performance information available on various projects throughout DoD. In addition, these datasets are also independently validated by agencies within the Department and subject to audits that require maximum fidelity to accounting standards.

It is worth noting that these datasets provide the best available information on DoD software acquisition, but are mainly limited to contract cost and budget performance (versus technical functionality performance) and were collected by hand. This scenario seems to address larger structural and cultural problems:

[34] Specifically, “within acquisition category (ACAT) I and IA programs and pre-MDAP and pre-MAIS programs, subsequent to milestone A approval.”

- The Department has no real acquisition data system that holds anything more than top-level data on our largest programs.
- There is no automated collection of acquisition data, despite the fact that software tools and infrastructures, from which data can be automatically extracted, are integral parts of the state of the practice in the software industry.
- For much of the limited software-specific data that we do have (for example, source lines of code, or SLOC), this study has argued that they do not provide meaningful technical insight. Metrics like SLOC are not what the private sector would use to assess and manage programs.
- Leadership often relies on experience and trusted advisors because timely, authoritative data are not available for real analysis.

G.2 Software Development Project Analysis

One area of analysis focused on the SRDR data to describe, at an enterprise- or portfolio-level, what the Department is able to say about its software based on the software-specific data. As described above, SRDR data are more project- or subcomponent-focused versus program- or contract-focused; indeed, it is not easy and perhaps not possible to create a program-level understanding of software activities from the SRDR data.

The results reported here address 3 three questions:

1. How well do software projects perform in terms of effort and schedule?
2. Is there a difference in project performance related to the size of the project and the use of agile development?
3. How long do software projects take to reach completion?

The source of the data was the May 2018 compilation file published by members of the Software Resources Data Report Working Group. This file contains 3993 submissions that yielded 475 initial reports of planning estimates, 598 reports of final actual values, and 295 pairs of initial and final reports. Upon further investigation, 131 pairs contained full life-cycle information and therefore serve as a better dataset for studying effort and schedule growth. Thus, while we base our conclusions in this section on the best available data for software, it is important to keep in mind the data represent only a small subset of the Department's software.

The results presented below were primarily based on common statistical methods. Although a variety of additional explorations were conducted, the results were not found to be stable or to have achieved high confidence. These included dynamic simulation modeling, causal learning, and analysis with repetitive partitioning and regression trees.

Software Project Effort and Schedule Performance

In the current DoD acquisition life cycle, substantial effort goes into defining requirements upfront in extensive detail, and projecting the cost and schedule for achieving the capabilities so described. Despite that, it is often said that the Department has problems acquiring the software capabilities it needs within budget and schedule. This analysis explored whether there was support for this conventional wisdom.

DoD projects in the dataset generally do indeed experience substantial effort growth. As seen in the following figure, the median number of estimated hours is 22,250 while the median number of actual hours is 30,120. (Note that the vast majority of points lie above the green line, indicating that actual values were greater than estimated.) The median rate of growth is 25%. However, there are some projects that expend less than their estimated effort, sometimes by a substantial amount as reflected by the points within the red circle. Unfortunately, based on the data reported we cannot discern whether they delivered the full committed functionality or not.

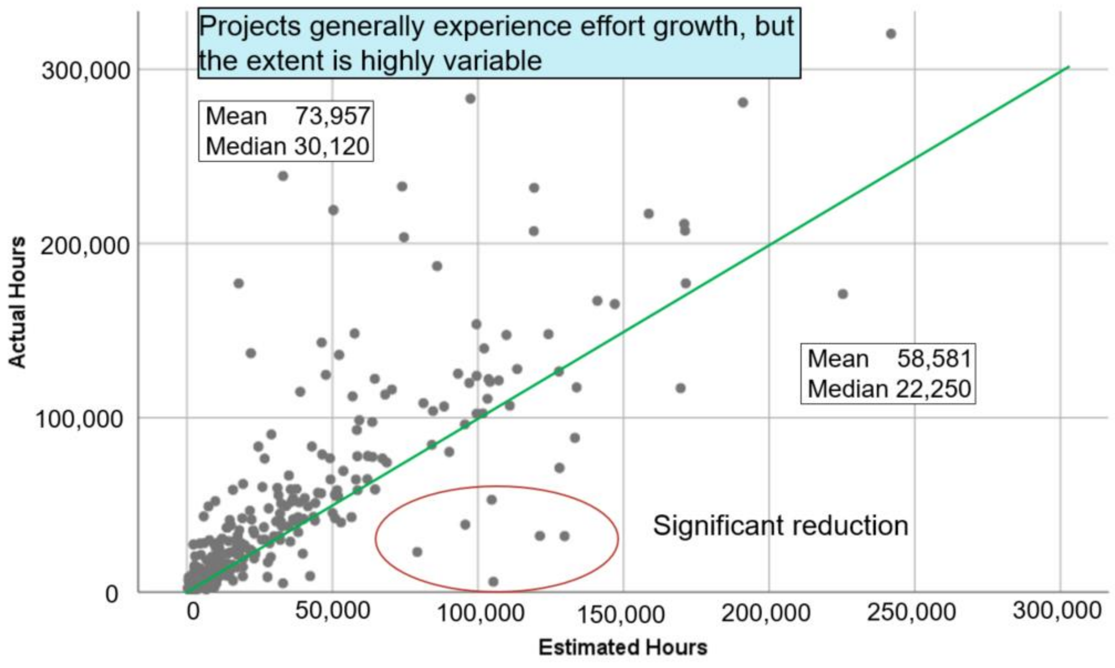


Figure G.1. Estimated and actual project hours for project with less than 300,000 estimated hours.

The growth in project duration is generally not as large as the growth in effort. The median planned duration is 28 months and the actual duration is 34.9 months. The median growth in duration is 12%.

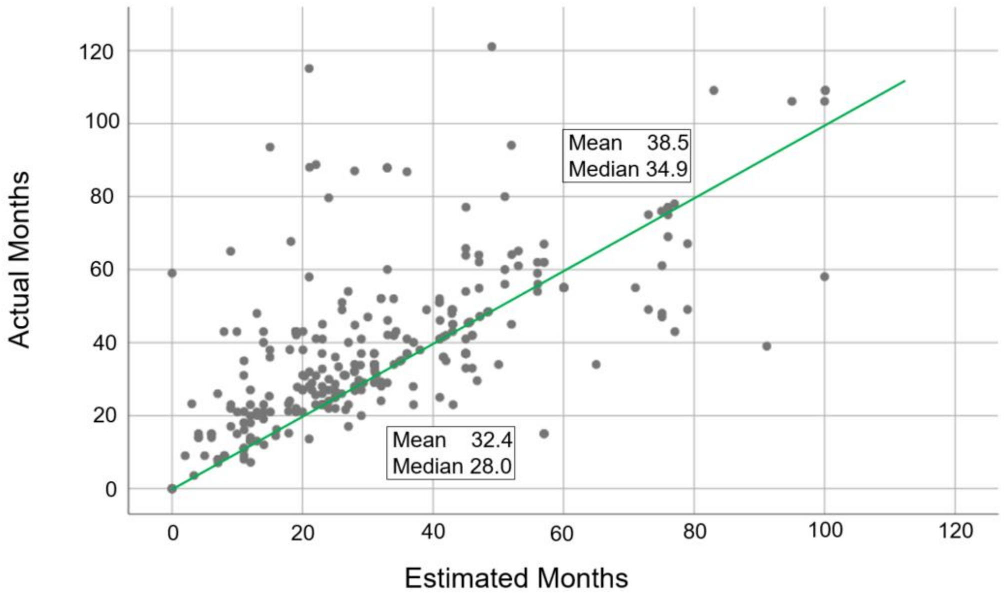


Figure G.2. Estimated and actual project duration.

Interestingly, effort and duration growth are only weakly correlated and the highly skewed nature of their distributions means that averages create a more negative impression of performance than may be warranted. That is, the average exaggerates the degree of growth across the portfolio of projects. Nonetheless, in the data we have available, overruns of effort and duration are the norm.

Does Project Size Affect Performance?

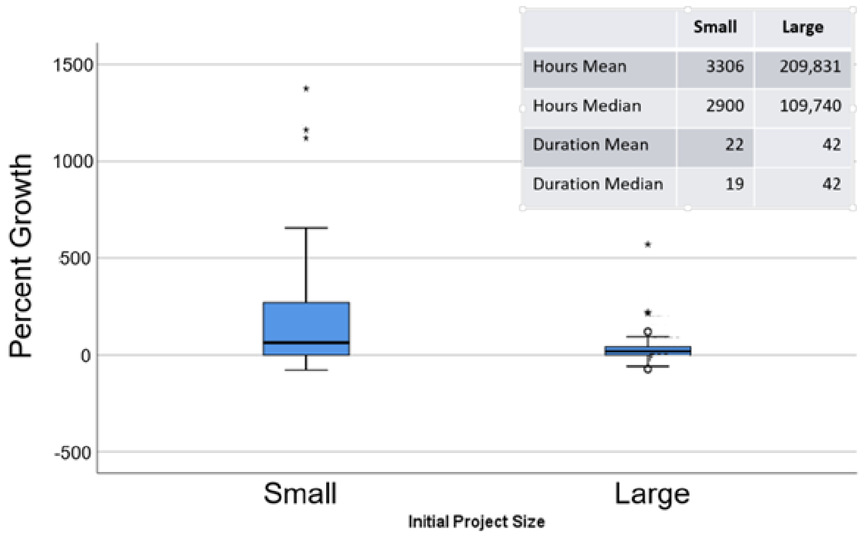
The DIB has recommended that software programs should start small. The next analysis examined the historical data available to test whether small programs performed better than large ones, at least in terms of delivering capabilities on time and within budget.

To perform this analysis, projects were categorized in terms of their estimated equivalent source lines of code (ESLOC)[35] and effort. ESLOC is not collected but computed from the detailed SLOC measures that are collected: ESLOC combines the different sources of lines of code, new, modified, reused, and autogenerated, into a single count. Projects that were in the lower and upper quartiles on both effort and ESLOC measures were labelled as small and large projects respectively. This yielded 53 small and 55 large projects. An analysis of variance was conducted for growth in effort and duration.

The results found that small projects do not outperform large projects. Large projects do have less effort growth on a percentage basis but more growth in terms of raw hours. Surprisingly,

[35] Elsewhere in this report, we reflect on the problems inherent with using SLOC as a measure. However, this is a key measure that has been collected historically by the department and so represents the best available data for this analysis.

schedule growth is very similar. Variation in performance overwhelms any apparent difference and the results do not achieve statistical significance.



	Small	Large
Hours Mean	3306	209,831
Hours Median	2900	109,740
Duration Mean	22	42
Duration Median	19	42

Figure G.3. Effort growth by project size.

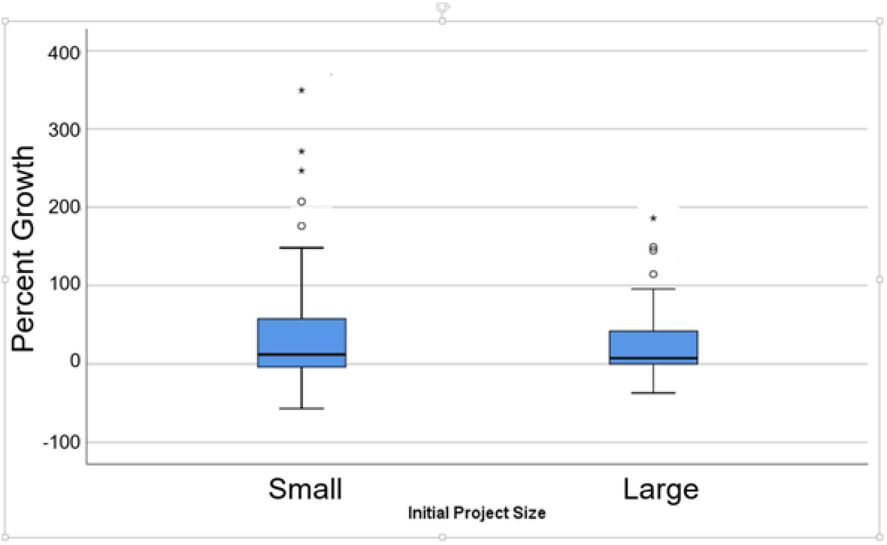


Figure G.4. Duration growth by project size.

The fact that small projects still experience the same growth as large projects does not negate the advice that projects should start small, iterate often, and be terminated early if unsuccessful, since this can still result in significant savings in costs for projects that are not performing well.

Do Development Approaches Affect Performance?

There is much interest in the software development community and DoD in the use of agile methods. While the most recently updated SRDR form explicitly calls out measures for agile projects, this has not been the case for the historical SRDR data upon which these analyses rely.

Furthermore, the identification of the development approach is captured in an open text field. This necessitated interpretation and grouping of the entries in order to perform this analysis. A significant number of projects reported using "Waterfall," "Incremental," "Spiral," or "Iterative" approaches. The remainder suggest use of a customized or hybrid approach. For the analysis here, "Waterfall" is compared to "Incremental," "Spiral," and "Iterative" projects.

Again, using ANOVA, the results indicate that effort growth does not significantly vary by development approach. However, duration growth is significantly less for projects using incremental development approaches as compared to waterfall (28% v 70% on average).

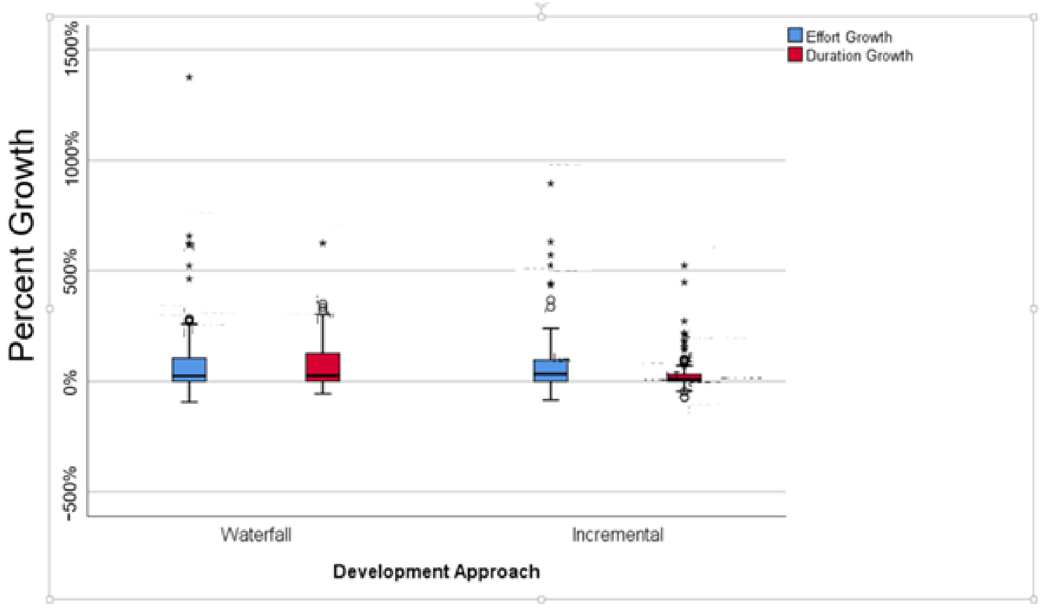
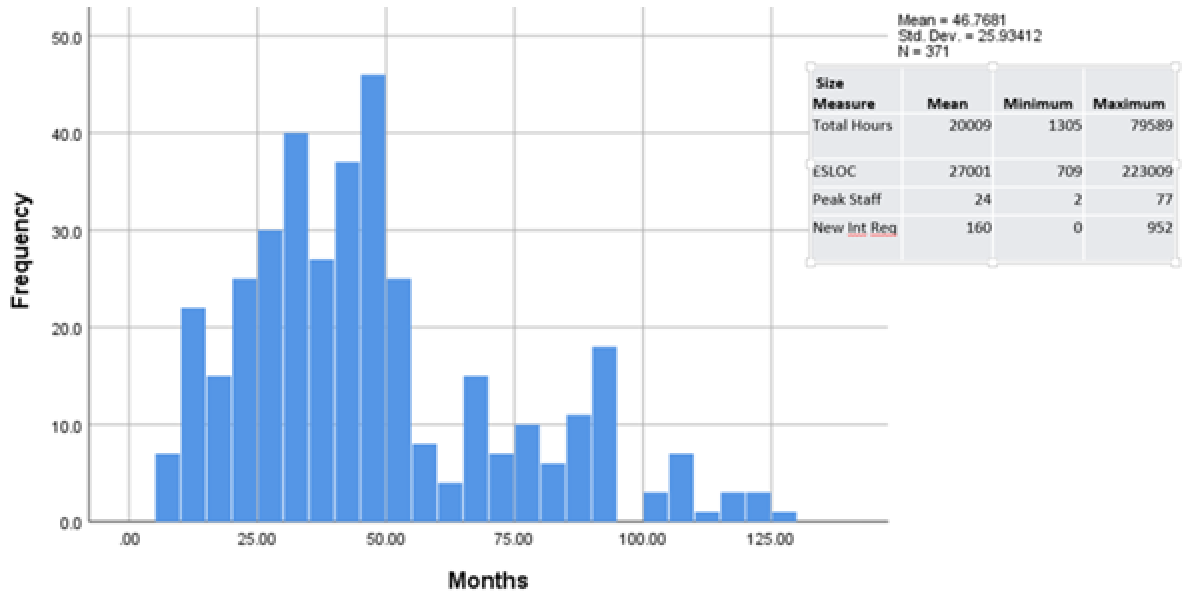


Figure G.5. Effort and duration growth by development approach.

How Long Does It Currently Take to Complete a Project/Deliver Software?

As can be seen in the following figure, it is very rare for a project to complete in 12 months or less. Out of 371 projects used for this analysis, only 21 (6%) completed in this timeframe.



Size Measure	Mean	Minimum	Maximum
Total Hours	20009	1305	79589
ESLOC	27001	709	223009
Peak Staff	24	2	77
New Int Req	160	0	952

Figure G.6. Actual duration for 371 AIS, Engineering, and Real-time projects.

Additional Insights from the SRDR Data

The preceding analyses were guided by the recommendations and proposed measures in DIB authored documents. In the course of performing those analyses, other questions and issues were posed and investigated. Briefly, these findings are:

1. Extreme variability in project performance confounds the identification of statistically significant results. This was noted above and is most likely actually due to performance and reporting inconsistencies.

2. Planned values can be useful for establishing expectations regarding reported actual effort and duration. That is, planned and actual values tend to be highly correlated with each other.

3. Planning for reuse is associated with significantly more schedule growth as compared to projects that do not plan for reuse.

The last one deserves more explanation as it is a somewhat counterintuitive result. Based on 275 projects that reported either no plan for code reuse or did plan for code reuse, the growth analysis showed no statistically significant differences in effort growth, but a significant difference in the amount of duration growth. Projects planning for code reuse had 52% duration growth as compared to only 20% for those that did not plan for code reuse. This phenomenon has been noted before and attributed to over-optimism about the amount and ease of code reuse. As the ability to reuse code falls short, unplanned effort and time go into producing new or modified code to compensate for the unrealized code reuse. Why effort growth is not significantly different is but likely at least partially related to the extreme variability in the performance measures.

Opportunities for Improving SRDR Data for Use

Issues regarding the data quality of SRDR data used here hampered the analyses. As is noted earlier, there is a substantial reduction from the number of submissions in the system to the number of usable records. At its most extreme there are 131 high quality pairs (262 records) out of the 3993 submissions included in the compilation dataset. That is, roughly 93% of the data is discarded.

The following opportunities are available for improving SRDR data for use in addition to supporting the needs of the DOD cost community. Briefly, they are:

1. Leverage data collection and reporting from automation within the software environments (software factory). Minimize the need for manual entry and transformation.
2. Capture information about the quality of the delivered system.
3. Make the data more broadly available and encourage analyses into DoD software challenges (DIB Recommendation A3).
4. Identify the information needs of the stakeholders and intended users of the data beyond the cost community.

G.3 Software Development Data Analyses

A second investigation focused on cost and schedule performance data reported on recently completed and ongoing software development efforts within DoD. As these data provided insights *within* programs (and allowed understanding how values changed over time), we expected that this analysis would allow for deeper dives that could better explain how software acquisition occurs in programs.

This information was extracted from IPMRs, which are deliverables required by most contracts. The team also reviewed SARs for the large ACAT I programs to gain perspective on programs as they evolve over time.

Poor Data Quality and Inconsistent Data Reporting

There are approximately 130 ACAT I programs reporting research and development (R&D) contract performance over the past 10 years. We discarded from our analysis:

- Contracts for which the first IPMR report showed 65% (or about two-thirds) completed in work scope, reasoning that too much of the work had occurred before data collection began;
- Contracts for which the latest IPMR reported work that was less than 70% complete, reasoning that we would not have the ability to evaluate a significant portion of work completed.

146 contracts (35%) did not meet these data quality criteria out of the total of the 413 ACAT I program development contracts for which we have data (Figure 7). The fact that more than one-third of contracts do not meet this criterion implies that DoD would benefit from improving the quality and consistency of software development performance reporting. DoD cannot comprehensively assess the performance and value of the billions of dollars in investment without insight into a third of the complete portfolio.

Additionally, there are many data that are of limited utility due inconsistencies related to reporting. These have to do with problems with filing the mandated regular reports, and a lack of contextual data (i.e., metadata) being collected in a readily analyzable form. The DIB Software Metrics Recommendations contain recommended best practices on data collection and metrics definitions to not only capture data, but to establish standards meant to enhance software development performance.

Cost and Schedule Data

The resulting list of contracts was prioritized based on the budget assigned to the software-specific development efforts, and the top 46 contracts with the largest budgets were included in this study. These 46 contracts covered roughly half of the total dollar scope for all development programs in our dataset, and thus provided a reasonable sample size for our analysis. In addition, 35 contracts for smaller ACAT II and ACAT III software intensive Command and Control (C2) and Automated Information System (AIS) programs were included in this analysis. This resulted in the study capturing 81 total contracts valued at $17.9B in software development cost over the past 10 years (2008-2018). This study did not attempt to qualify or quantify the reasons for cost and schedule growth, recognizing that growth is not always indicative of poor performance by the program and/or contractor.

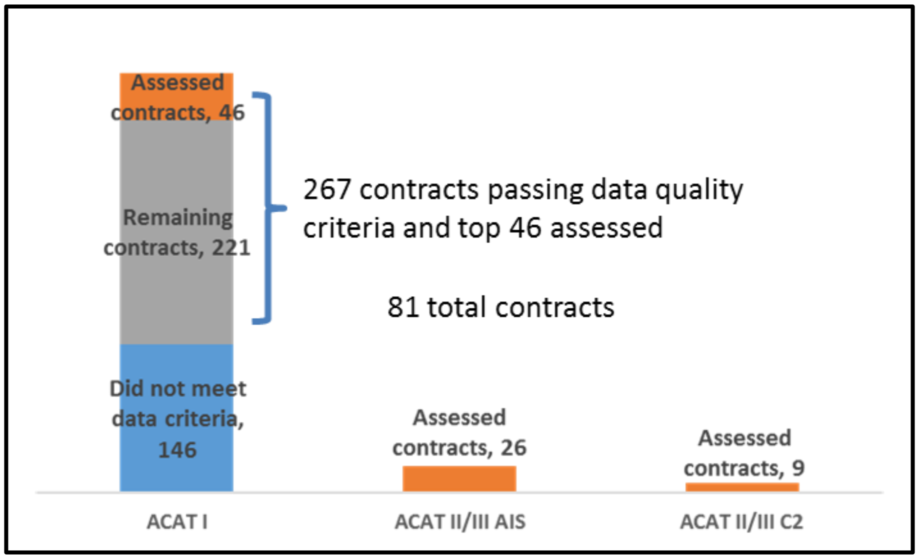


Figure G.7. Results of contract selection process.

The 81 total contracts included in this analysis covered the portfolio of DoD programs, including software intensive C2 and AIS programs as well as aircraft, radars, land vehicles, and missile weapon systems, as shown in Figure 8.

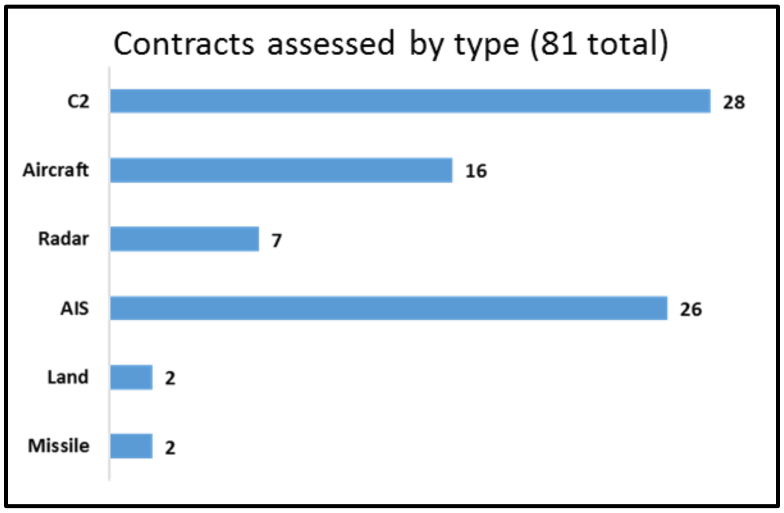


Figure G.8. Contracts analyzed by weapon system type.

Large Software Cost Growth

The analysis of IPMR data found that on average, the contracts experienced 138% cost growth. The total combined value of the software development budgets within these contracts was $7.6B at the time of initial reporting. By the time these contracts reported the latest (or in some cases, final) performance baseline, the software development budget total grew by $10.4B. Based on the analysis completed, significant software development cost growth was experienced across all platform and program types, resulting in a second observation: In general, the DoD struggles to minimize software development cost growth across the complete portfolio of projects. Figure 9 provides a summary of the 81 contracts evaluated, organized by project and by platform type. Note that the cost growth of “C2 Program A05” was truncated in the figure as it was an outlier in the analysis.

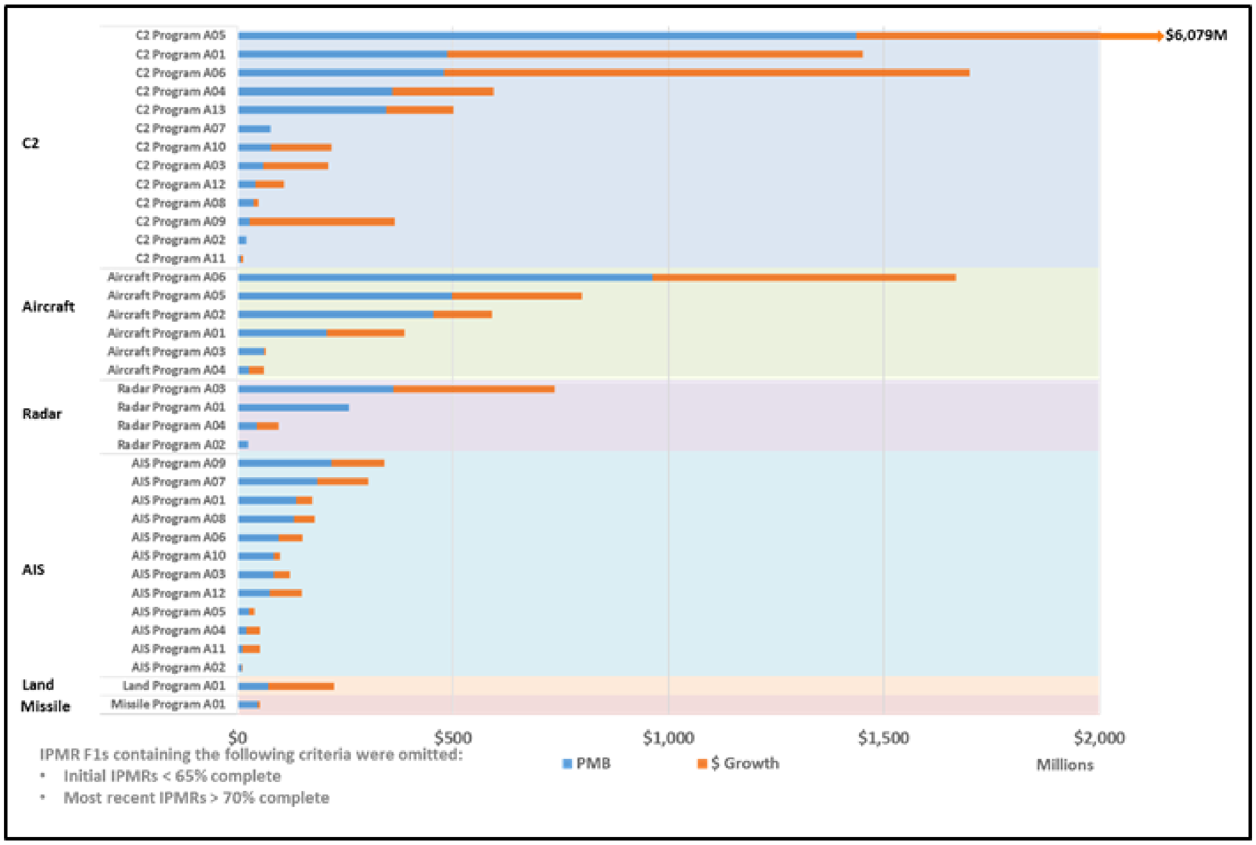


Figure G.9. Contract software development cost growth by program and by platform.

The study team used information provided by SARs and other relevant acquisition documentation to calculate project schedule growth. Figure 10 illustrates both dimensions of cost and schedule performance and identifies programs for which actual performance exceeds more than twice the baseline cost and schedule. Two programs, “AIS Program A01” and “C2 Program A02,” experienced cost or schedule growth so extreme that the bounds of the diagram axis plots were exceeded. This figure also supports the second observation that recent software development programs experience significant cost growth. The DIB SW Commandment 3 addresses cost growth by advocating that software budgets be planned upfront to support the full life cycle versus the current funding life cycle, defined around Planning, Programming, Budgeting, and Execution (PPB&E).

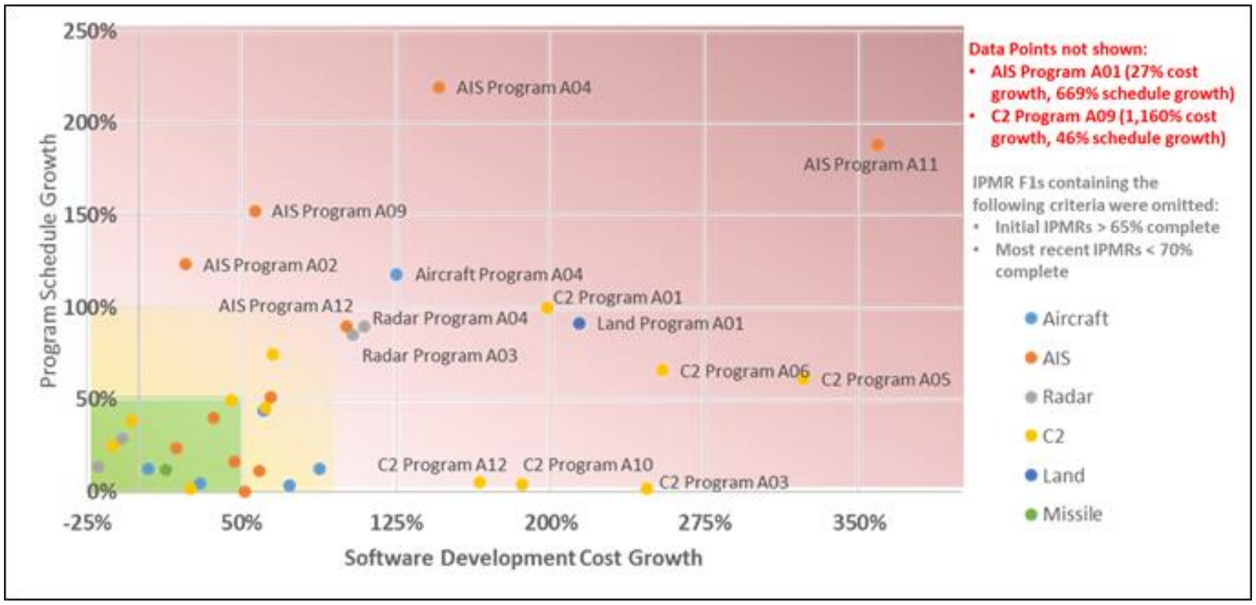


Figure G.10. Software development cost growth vs. program schedule growth

Long Planned Durations and Frequent Re-baselining

The third study observation results from a deeper look into programs with high cost growth. This research found that in numerous instances, program baselines shifted (re-baselined) during the contract period of performance. The contracts with what appear to be significant "re-rebaselining" (i.e., multiple recurring increases to the expected cost) were analyzed in further detail.

SAR program milestones and available open source data were evaluated to provide a scale of time and functionality. It is observed that the software development effort crosses the same percent complete, as defined by the Earned Value Management (EVM) metric as the ratio of Budgeted Cost of Work Performed (BCWP) to Budget at Completion (BAC), multiple times. This represents an incremental method of adding cost, which is presumably associated with the addition of technical scope and requirements, which can result in a doubling or tripling of the total original budgeted value of the software development effort.

Figure 11 provides an example of this behavior, showing the "C2 Program A01" program effort that appears to re-baseline several times. The software development effort crosses the same percent complete point multiple times.

DIB Software Commandment 2 provides the recommendation that software development should begin small, be iterative and build on success; otherwise, be terminated quickly. DoD programs that take this approach are likely to see an improvement in performance once scope and requirements can be delimited through successful iteration. The behavior demonstrated in Figure 11 seems to indicate that to some extent, at least some programs are already behaving in an iterative way that better suits the technical work of software evolution. Unfortunately, our reporting mechanisms are not suited to reflect this reality, and in fact cannot differentiate a reasonable approach to incremental development from problematic cost or schedule growth. Looking just at the top-line numbers, these instances could be interpreted as excessive cost growth on the program, representing a problem from the Department's point of view since the predictability of performance against cost and schedule baselines are normally taken as indicators of success. What this scenario seems to point to is a need to improve our metrics collection to better reflect the underlying technical reality of software, where good performance often leads to a demand for new capabilities and new scope, as well as better educating our decision makers about how to interpret the results.

Thus this example provides more information about associated reporting issues tied to observation 5, that budgets should be contracted to support the full, iterative life cycle of the software being procured with amounts definitized proportionally to the criticality and utility of the software.

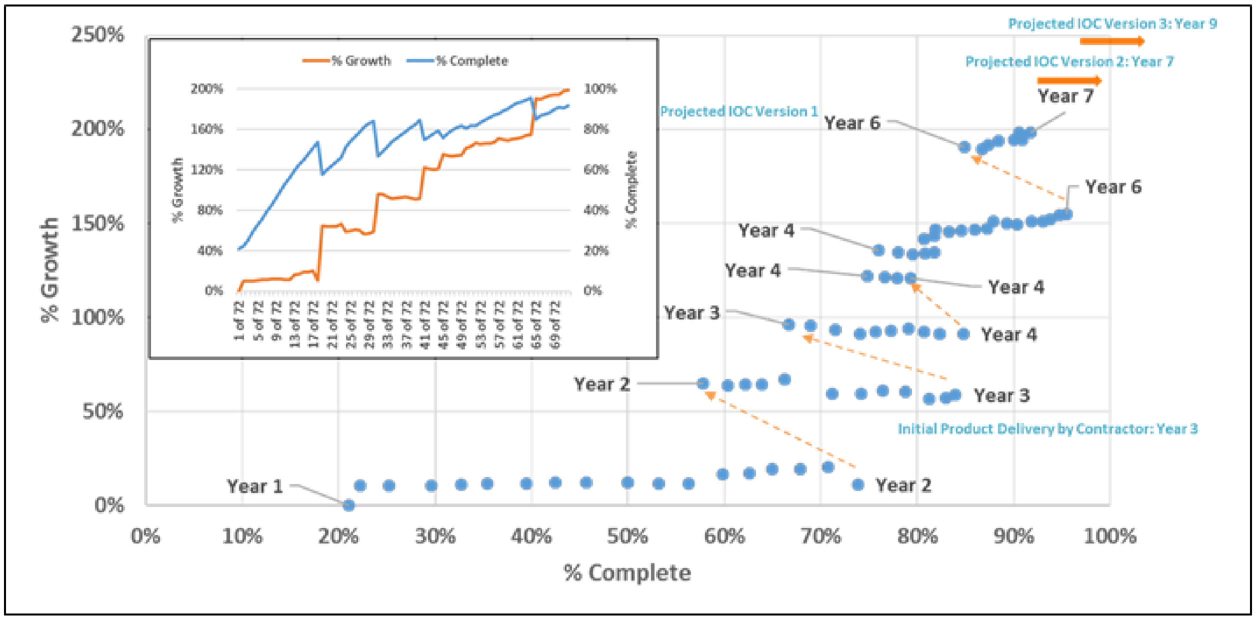


Figure G.11. C2 Program A01 performance measurement re-baselining.

Agile Software Development Can Improve Program Performance

This study researched the performance of agile development methods that are implemented in existing programs. IPMRs do not explicitly state the type of development effort being used (incremental, agile, etc.). However, an article published in the journal Defense Acquisition provided an instance where agile development was applied and considered a success story. Although this article did not name the program, we were able to identify the most likely candidate, "Aircraft Program A05," by matching the timeline presented in the article against the timeline of contracts that we could see in the program data.

The IPMR data for this program are shown in Figure 12. The contract work completed using an agile approach are shown in blue and represent a 21% cost reduction when compared to the initial budgeted value. This is in contrast to the contracts that seem to adopt a waterfall development methodology, i.e., contracts with planned long durations, which are shown in shades of orange and represent a 129% cost growth compared to the initial budgeted cost.

This analysis supports the fourth study observation that agile development may reduce cost growth compared to more traditional waterfall approaches. The DIB SW Commandment 2 also advocates that agile approaches seen in commercial development result in faster deployment of functionality and cost savings which we observe in this instance.

Though a comparison of cost is one facet of performance, more research is required to increase the certainty that better overall performance and results were achieved with agile methods.

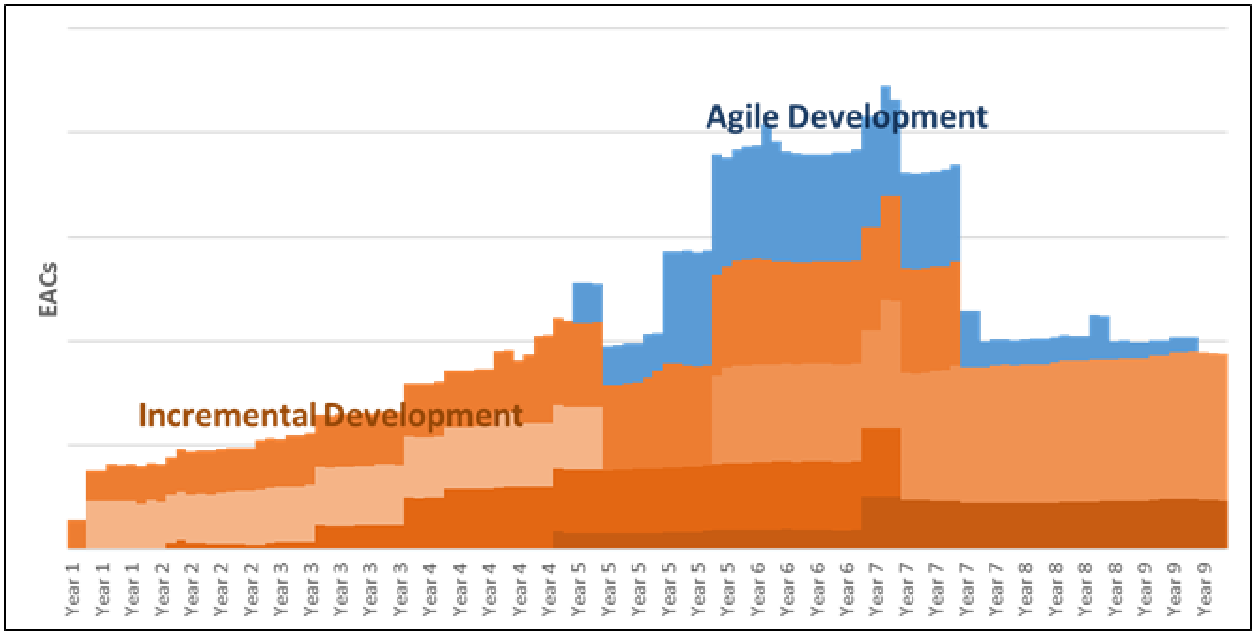


Figure G.12. Aircraft program A05: incremental vs. agile development efforts

Cost and Schedule Analysis Summary

In important ways, this analysis was typical of other efforts that aim to use Department data to examine the performance of acquisition. Due to the limited nature of the data available, our best analyses typically take months to create, with substantial time needed to find the data, to collect them, and to compile them into a structured format from multiple siloed and restricted systems.

The observations taken from data analysis of DoD program cost and schedule performance support the supposition that the current state of software acquisition is highly problematic and unsustainable relative to affordability and functionality. The DIB SW Commandments 2, 3, and 4 provide recommended measures to contain growth and increase the opportunity for cost savings by detaching software development from a hardware manufacturing industrial model and integrating software development and operations to quickly provide functionality to users and meet changing needs dictated by a dynamic global environment.

The preceding sections have described specific conclusions from the analyses our team conducted. Equally important, however, are the types of analyses we were *unable* to conduct given the data that were available.

A notable omission is that the Department is unable to address questions of *how much* software it has. Not in terms of software size but in terms of an index of how many important software systems have been acquired or are being sustained by the Department: There is no DoD or Service framework for describing the types of software intensive systems, or any inventory/ catalogue of the software in use. As a result, it is challenging to comprehend the scope and magnitude of the DoD software enterprise, and to design appropriate solutions for issues such as infrastructure or workforce that can meet the magnitude of the problem. Although done at a smaller scale, NASA's software inventory is an example of such an inventory model that is used to make strategic decisions for a federal agency.[36]

There is a large and growing body of work on software analytics, the automated or tool-assisted analysis of data about software systems (usually collected automatically) in order to make decisions. Conferences such as Mining Software Repositories[37] and Automated Software Engineering[38] annually showcase the best of the new research in these areas, and these methods are having a practical impact in commercial and government environments as well. A summary of software analytic applications lists several important questions that can be explored in this way: to name just a few, "using process data to predict overall project effort, using software process models to learn effective project changes, … using execution traces to learn normal interface usage patterns, … using bug databases to learn defect predictors that guide inspections teams to where code is most likely to fail."[39] Without access to its own software data, DoD is missing the opportunity to exploit another area of research that could provide practical benefit for improving acquisition.

In a later section of this report (Appendix H), we provide the results of a small study that was undertaken to demonstrate potential practical impacts that could be achieved if software data access could be possible in the future.

[36] NASA Engineering Handbook (https://swehb.nasa.gov/display/7150/SWE-006+-+Agency+Software+Inventory#_tabs-6).

[37] https://2018.msrconf.org/

[38] http://ase-conferences.org/

[39] T. Menzies and T. Zimmermann, "Software Analytics: So What?," in *IEEE Software*, vol. 30, no. 4, pp. 31-37, July-Aug. 2013. DOI: 10.1109/MS.2013.86

Appendix H: ~~Replacing~~ Augmenting CAPE with AI/ML

Linda Harrell, John Piorkoski, Phil Koshute, Erhan Guven, Marc Johnson (JHU/APL)
Vladimir Filkov, Farhana Sarkar, Guowei Yang, Anze Wang (UC Davis)
Steven Lee (Rotunda Solutions)

H.1 Introduction

The Defense Innovation Board (DIB) Software Acquisition and Practices (SWAP) study chartered an exploratory study to explore the use of modern tools in data analytics and Machine Learning (ML) to provide insights into cost, time, and quality of Department of Defense (DoD) software projects. The data analytics and ML effort were performed by a team from academia (University of California Davis (UC-Davis)), a university affiliated research center (The Johns Hopkins University Applied Physics Laboratory (JHU/APL)) and industry (Rotunda Solutions). Since a suitable DoD data set was not available, the three teams leveraged existing data sets that were readily available to perform ML experiments and quickly get results.

ML models were created to predict the cost, time, and other aspects of software projects and gain a deeper understanding of the potential impact of project characteristics on overall project budget and effort. The models were trained with different data sets and were constructed to predict different performance metrics throughout the software development life cycle.

The JHU/APL team developed ML models to predict software project duration and effort using the commercially available International Software Benchmarking Standards Group (ISBSG) Development and Enhancement (D&E) Repository of completed software projects. The UC-Davis team developed ML models to forecast software project duration, effort, and popularity using the publicly available GitHub repository of open-source projects. Finally, Rotunda Solutions created a defect density ML model to capture the code complexity and predict potential risk of code modules using a publicly available NASA dataset.

Additionally, the Rotunda Solutions team identified a number of opportunities for harnessing ML and Artificial Intelligence (AI) to improve the software acquisition process during different phases of the procurement cycle. This research effort is referred to as the Opportunities for Analytic Intervention. Rotunda Solutions also started development of a conceptual mock-up to explore some of these opportunities.

Overall, the three ML model development approaches demonstrated promising results aimed at improving predictions of software cost, time, and quality during different life-cycle phases.

- The JHU/APL team identified features (software metrics) that can support predictions of duration and effort at the project onset and shows that ML models have very good accuracy even with as few as 5 to 15 important features, most of which can be easily collected. It also shows how the prediction accuracy increases slightly by also including the effort expended in different life-cycle phases (e.g., planning, specification, design, build, test, and implementation). Since this analysis addresses the whole software life cycle, the APL effort is referred to as the Software Life-Cycle Prediction Model.

- The UC-Davis team shows how monitoring of software development activities over time via automated tools that capture metrics (such as the number of lines of code, the number of commits, and team size) can support accurate forecasts of duration, software effort (SWE), and software popularity. Additionally, the UC-Davis analysis showed that the ML models could obtain very good forecasting accuracy only 6 months after code development has started. Hence the UC-Davis ML model can serve as an early warning indicator. Since this analysis leveraged data obtained during software development activities to forecast future outcomes, it is referred to as the Software Development Forecasting Model.
- The Rotunda Solutions defect density model automatically processed code files and output code complexity metrics to aid efficient resource allocations and risk mitigation.

Interestingly, despite the differences in the approaches taken by JHU/APL and UC-Davis, the teams shared similar conclusions. For instance, both teams identified the team size and the project timing as being important features for the predictions.

Section H.2 of this document describes the methodology applied to the APL Software Life-Cycle Prediction Model and the UC-Davis Software Development Forecasting Model. Section H.3 summarizes the major findings of all three analyses. Section H.4 offers implications of these study results for DoD programs.

H.2 Methodology

The approaches taken for the APL Software Life-Cycle Prediction Model and the UC-Davis Software Development Forecasting Model were complementary. Table H.1 summarizes key aspects of the two approaches. These aspects include:

ML Techniques. Both studies leveraged readily available commercial or open- source ML techniques. This enabled the teams to meet the task's quick reaction turn-around timeline and also ensures that DoD government personnel and contractors can apply a similar approach when they develop their own prediction models for software projects. Although the teams developed several types of ML models, this report focuses on those with the best results: the APL Random Forest (RF) and the UC-Davis Neural Network (NN) models.

Data Sets. The APL team leveraged the 2018 International Software Benchmarking Standards Group (ISBSG) Development and Enhancement (D&E) Repository of completed software projects. This diverse database contains thousands of software projects that are described by a rich set of features that span the whole software life cycle, but most of these projects have less than one year in duration or less than two years of effort. The UC-Davis team mined the GitHub collaborative project development and repository site, which contains historical trace data captured from millions of open-source software projects. The resulting database includes hundreds of thousands projects of various sizes. Its feature set is not as rich as in the ISBSG database, but it automatically tracks development metrics including commits, discussions, and other activities.

Target Variables. The APL team focuses on predicting software project duration and effort, two of the three metrics of greatest interest to the DIB. On the other hand, the UC-Davis team aims to predict the project duration (via its proxy months committed), the number of software commits (which is an incomplete proxy for software effort), and the number of stars (which is an indicator of the popularity of a project in GitHub).

Project Tiers and Boundaries. Large differences between proposal estimates and actual outcomes for software development duration and effort cause the biggest challenges for DoD; small deviations are much more manageable. To reflect this perspective, both studies gathered their target variables into discrete tiers with boundaries shown in Figure H.1.

Performance Metrics. Both studies assessed the performance of their models with confusion matrices (which shows the distribution of predictions in terms of predicted and actual tiers) and overall accuracy.

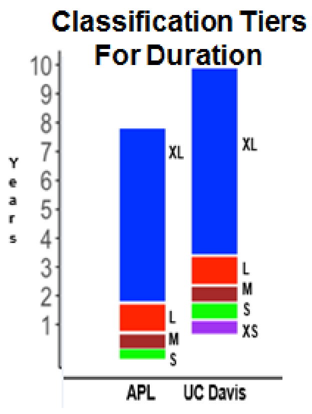
Table H.1. Key aspects of APL and UC-Davis studies

Parameter		**APL Software Life-Cycle Prediction Model**	**UC-Davis Software Development Forecast Model**
Data Set		2018 ISBSG D&E Repository	2018 GitHub Repository
Number of Projects (after preprocessing)		2,818	Approx. 127,000
Number of Features (after reduction)		176	36
Target Variables for …	Duration	Project Duration	Months Committed
	Effort	Effort	Total Number of Commits
	Popularity	N/A	Number of Stars
ML Techniques		Off-the-shelf (NB, SVM, RF)	Off-the-shelf (MR, NB, RF, NN)
Results: Overall Accuracy; Confusion Matrices		Overall accuracy: Yes Confusion Matrix: 4 tier	Overall accuracy: Yes Confusion Matrix: 5 tier
Prediction Snapshots		Early concept development and procurement; Software development in process	After 6 months of software development ; Most recent software development
Feature Reduction		Yes	Yes

Definitions: NB = Naive Bayes, SVM = Support Vector Machines, MR = Multivariate Regression, NN = Neural Networks

APL Classification Tiers

Project Size	Software Effort
Small	2 months to 0.999 year
Medium	1 year to 1.999 years
Large	2 years to 7.999 years
Extra Large	>= 8 years



UC Davis Classification Tiers

Project Size	Number of Commits	Number of Stars
Extra Small	30 - 90	0 - 2
Small	91 - 140	3 - 7
Medium	141 - 217	8 - 22
Large	218 - 384	23 - 90
Extra Large	385 - 3469	91 - 31168

Figure H.1. Classification tier boundaries.

Prediction/Forecasting Snapshots. APL made predictions at two project phases (snapshots). The first snapshot is at onset, which includes features that are available or can be estimated during the concept, proposal, and procurement stage. The second is after software development has been underway; it can include additional features as they become available. UC-Davis made predictions at three snapshots, corresponding to the time elapsed for each project: 6 months from first commit, 12 months from first commit, and most recent snapshot (1/1/2018). The most recent snapshot is taken to be the actual outcome (even if the project is still under development). For simplicity, the results with the 12-month snapshot are not discussed herein.

Feature Importance Ranking and Reduction. The APL RF and UC-Davis NN models both determined feature importance by evaluating the importance of each feature to the overall accuracy prediction and developed corresponding models with only the top ranked features.

Pre-Processing and Feature Selection. The pre-processing actions taken by the APL and UC-Davis are discussed in separate reports.

Project Context (Cluster) Creation. To fine-tune their predictive models, UC-Davis used an Autoencoder NN to group projects into four similarity clusters (i.e., contexts). A separate model NN was trained for each cluster. This technique allows for greater accuracy when project context is known early on, by, for example, tracking project metrics from the start.

H.3 Key Results and Findings

APL Software Life-Cycle Prediction Model

Table H.2 shows the performance of the APL models that predict software project duration and effort with all features included. Even with minimal data cleaning, model tweaking, or sensitivity studies, and using a very sparse and unevenly distributed data set, the ML models predict a project's size tier with an overall accuracy ranging from 57% to 74%. These are impressive results for a quick-turnaround exploratory analysis.

As expected, the prediction estimates once development is underway are better than the predictions at program onset. This is because additional features, such as the effort expended in various life-cycle phases, help to improve predictions. However, with the features included in this analysis, the improvement was slight.

Even when the ML model does not correctly predict the size of the software project, the prediction is most often in adjacent tiers rather than significantly further away. This is evident in the confusion matrix in Table H.3 and the additional confusion matrices provided in separate reports. This is important because it indicates that incorrect predictions still tend to be fairly close (e.g., an extra-large project predicted as large or vice versa).

Table H.2. Performance summary for APL prediction models (with all features)

Model	**Overall Accuracy**
Predicting Duration at Project Onset	57%
Predicting Duration after the Project is Underway	58%
Predicting Effort at Project Onset	68%
Predicting Effort after the Project is Underway	74%

Table H.3. APL confusion matrix for predicting effort as project is underway (with all features)

Accuracy values are shown as a percent of all projects of a given class		**Predicted Class**			
		S	**M**	**L**	**XL**
Actual Class	**Small (S)**	80	18	2	0.1
	Medium (M)	23	59	18	0.6
	Large (L)	2	20	73	5
	Extra Large (XL)	0.1	0.8	14	85

Table H.4 identifies the most important features that influence the predictions. Naturally, the ranking of importance for each feature varies slightly for the predictions of duration and effort and for the two different phases (at project onset versus while the software development is underway), but the discrepancies are generally slight. Encouragingly, the features in this table are generally easy to obtain or estimate: function point standards, team size, software type, project implementation date, scope, programming language. The only feature category that is time consuming to gather is the functional size estimate. Each of the features in these tables is further described in the APL report.

Table H.4 Most important features for ML accuracy predictions

Category of Feature	**Most Important Features**	**Project Phase**
Software Size	Functional Size, Relative Size, Adjusted Function Points	Project Onset
Standards for Function Point Estimates	Function Point Standards, Count Approach	Project Onset
Team	Maximum Team Size, Team Size	Project Onset
Type of Software	Industry Sector, Organization Type, Application Type, Business Area	Project Onset
Timing	Year of Project, Implementation Date	Project Onset
Scope	Project Activities, Development Type	Project Onset
Programming Language	Primary Programming Language, Language Type, Development Platform	Project Onset
Incremental Effort	Effort in the Planning Phase, Effort in Specify Phase, Effort in Design Phase, Effort in Build Phase, Effort for Implementation, Effort in Test Phase	When the Project is Underway
Cost	Total Project Cost	When the Project is Underway

Figure H.2 depicts the accuracy prediction with small subsets of the most important features, and shows how the accuracy increases as additional features are added. This figure shows that although the database includes 176 features, very good predictions can be obtained using only as few as 5 to 15 features. These features are captured in Table H.4.

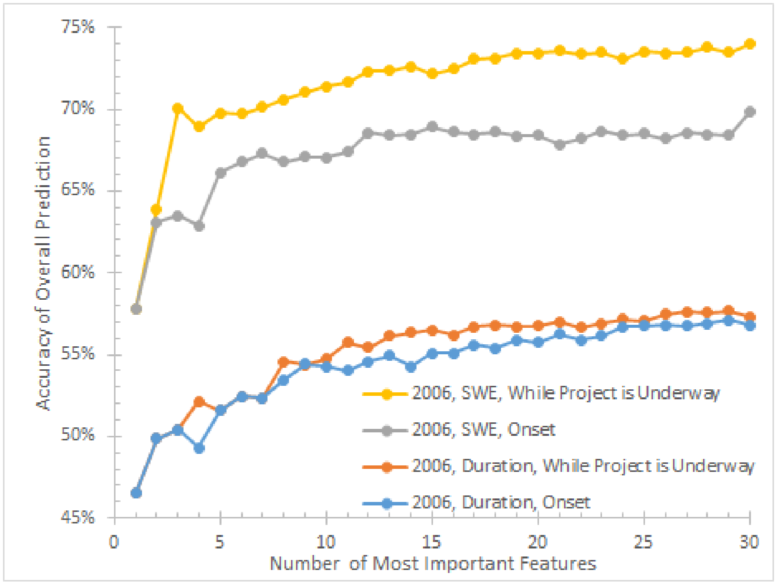


Figure H.2. Accuracy of APL's software project duration and software effort (with reduced, prioritized feature set).

The APL Software Life-Cycle Prediction model results clearly show that ML models can quickly be developed and trained using only a relatively small number of projects, a very small number of features, and a large amount of missing data. Furthermore, the resulting predictions for a software project's duration and total effort can be reasonably accurate at the project onset, and can then improve slightly over time by tracking the effort that is expended over the life cycle. Only about 5 to 15 features are required to achieve reasonable predictions. The most important features for the predictions were identified; most of them are easy to obtain or estimate.

UC-Davis Software Development Forecasting Model

UC-Davis developed models that predict project duration, number of commits, and popularity using all available historical data of completed projects in the January 2018 snapshot, starting from the first commit of software. Table 3.4 shows the best-case overall prediction accuracies that can be obtained with these models and all of this data. The best-case overall accuracy of the prediction estimate for project duration is 84% and the best-case overall accuracy of the prediction estimate for the number of commits is 72%. Predictions for popularity were less accurate. These results indicate that the features in the GitHub database will be very useful for predicting software project duration and to a lesser extent the predictions for the number of commits. It appears that additional features will be necessary to improve the predictions for software popularity.

Additionally, Table H.5 also shows that the best-case overall accuracy results for these models vary for different context clusters of similar projects. For instance, the accuracy values for each target variable increase within certain clusters; accuracy is greater in Cluster 1 by 16% for project duration and by 24% for number of commits and in Cluster 4 by 13% for popularity. These increases suggest that clustering projects based on similar context can increase the best-case prediction accuracy and that different models may be necessary to best predict different project contexts. The descriptions of these different clusters are not available at this time, but it would be valuable to investigate this further in order to understand the project characteristics that distinguish the clusters.

Table H.6 shows the best-case overall accuracy of the UC-Davis models that use only the 9 most important features from the full project lifetime. These results are very close to those of the models that use all available features, indicating that the reduced feature set is sufficient for accurate predictions.

Table H.5. Full lifetime (best-case) prediction accuracy

Target Variable	All Projects	Cluster 1	Cluster 2	Cluster 3	Cluster 4
Number of Projects	126,799	21,462	31,918	55,065	18,354
Project Duration (months committed)	84%	99.5%	83%	80%	78%
Number of Commits	72%	96%	70%	62%	69%
Popularity (number of stars)	49%	46%	48%	42%	62%

Table H.6. Full lifetime (best-case) prediction accuracy with reduced feature set

Target Variable	**All Features** (All Clusters)	**9 Most Important Features** (All Clusters)
Project Duration (months committed)	84%	84%
Number of Commits	72%	74%
Popularity (number of stars)	49%	48%

Table H.7 shows the accuracy results of the forecasting models, which predict the target variable in the final snapshot using features from a snapshot taken 6 months after project starts. These results are averaged over each of the 4 clusters (i.e., include 126,799 projects). These forecasting results show that data from only the first 6 months into a project can predict future outcomes, reaching accuracies of approximately 50% for both project duration and number of commits.

Table H.8 identifies the most important features that influenced the UC-Davis predictions and forecasting. This table shows that features related to teams and commit activity are the most important for the UC-Davis models.

Table H.7. Forecasting accuracy (averaged over all clusters)

Target Variable	**Prediction of target variable at last snapshot given 6 month snapshot**	**Prediction of target variable at last snapshot given all data**
Project Duration (months committed)	53%	84%
Number of Commits	50%	72%
Popularity (number of stars)	41%	49%

Table H.8. Most important features for the UC-Davis predictions and forecasting

Feature Category	**Most Important Features**
Commit Activity Data	First Commit Date, Months Committed
Team Member Data	Team Size, Number of Commenters, Number of Pull Request Mergers, Average Months Active, Standard Deviation (SD) Months Active, Average Commits per Month, SD Commits per Month

In summary, the UC-Davis analysis shows excellent results for being able to forecast project duration and the number of commits only 6 months into a project. Only 9 features are required to achieve these forecasts. The most important features for the predictions were identified; all of them easily obtained with automation tools that track software development activities. Additionally, UC-Davis uncovered clusters of projects that if better understood could lead to improved models and accuracy predictions.

Rotunda Solutions Investigation of Opportunities for Analytic Intervention

The Rotunda Solutions effort focused on identifying strategic opportunities to leverage ML and AI at key points in the overall DoD procurement process. It extended academic research and state-of-the-art quality management principles to identify opportunities to improve the likelihood of successful software development outcomes. It also developed initial conceptual mock-ups to explore potential applications, including a defect prediction platform.

Rotunda Solutions adopted a basic stage-gate model to represent the general structure and stages of a DoD procurement and project development effort. Multiple opportunities are identified in each stage where analytics, ML, and other modern techniques can assist project managers. First, analytics can provide metrics and insights to support the project manager’s yes/no/hold decision for whether the project should move to the next development stage. Second, analytics and ML can facilitate the search and interpretation of DoD procurement and development data sets so that decision makers have better access to historical data. Third, analytics can be run on this historical data to provide insights that can inform future projects. The application of modern techniques within a basic stage-gate model for a typical DoD procurement and development project can be envisioned as follows.

Stage 1: Idea Generation/Need Analysis. Analyze the internal unstructured documents from the program office and communications between suppliers and procurement officials. Then apply problem identification analytics to define the problem to be solved, considering the following 5 major groups/factors: need spotting, solution spotting, mental invention, market research, and trend. The literature shows a clear trend in savings of time and resources during the development process by maximizing the effectiveness of the idea generation stage.

Stages 2 and 3: Proposal Development and Response. Analyze internal unstructured documents from the program office and communications as they relate to proposal development and response. Use qualitative techniques such as focus groups, in-depth interviews, and surveys to determine factors associated with development success and failure. Additionally, use natural language processing (NLP) techniques to prepare the documents for further analysis. Both methods can identify key mechanisms and characteristics of software development success.

Stage 4: Contract and Award. Identify keywords through analysis of prior software contracts. Use NLP and topic extraction on legal documents surrounding the final selection of the supplier, contract vehicles, set-asides, and all stipulations to determine content. This can increase the ease of detecting associations between numerous demographic and supplier characteristics and software development performance. It also provides the ability to build a grading system and general profile of contractors and their performance on projects.

Stage 5: Software Development. Gather representative data regarding project management metrics, code base, and development metrics, and compile a list of metrics that can help identify the likelihood of success of a DoD software development project. This helps DoD in two ways: first by identifying projects that are likely to succeed or fail in each stage; and second by informing cost and time estimates for future software acquisition projects. Alternatively, analyze code to inform the development of ML tools to assist project managers and developers understand the state of their code. Potential benefits of this analysis include tools that can rapidly identify errors and increase efficiency for automation, audits, process checkpoints, and standardization.

Stage 6: Implementation. Harness available information on users, development, delivery personnel, and performance metrics of the software system. Measure the efficacy of the deployed or implemented software systems through metrics such as dependability, system performance, extensibility, and cross-platform functionality. This provides a postmortem analysis of the efficiency and effectiveness of the software and the development process, allowing DoD to learn from past experience and increase the likelihood of future development success.

Conceptual Mock-Ups

Rotunda Solutions aims to help DoD in four ways: (1) understand the potential impact of variables, decisions, and project characteristics on project budget and effort, based on historical data of similar projects; (2) make data-informed project decisions pertaining to the adjustment of project structure, methods, and other details; (3) create and explore what-if scenarios to promote better planning; and (4) encourage transparency and traceability of factors and decision-points affecting project performance. To this end, a number of concepts offer potential for further development and exploration. For instance, the concept of an "intelligent" burn-down chart is especially intriguing. Given sufficient sprint data and historical trend data, effort estimation tools and ML algorithms can be leveraged to make real-time predictions and issue alerts when estimates of team effort needs a closer review. Also, a defect prediction algorithm may be able to support risk mitigation activities and improve resource allocations.

Focus Area: Defect Prediction Platform

Software defect prevention is an essential part of the quality improvement process; timely identification of defects is important for efficient resource allocation, increased productivity, and risk mitigation, yet complete testing of an entire system is generally not feasible due to budget and time constraints. Studies show that the majority of software bugs are often contained within a small number of modules. To more rapidly identify these modules, Rotunda Solutions developed a system to automatically process code files and output code complexity metrics. They built off extensive industry research and tested representative NASA software modules using NN, SVM, Gaussian mixtures, and ensembles of ML techniques. The NN model performed best and was selected for production.

The NN model consists of 8 hidden layers, each layer becoming smaller until converging on a single probability to represent the existence of defects in the file. This model learns to assign importance weights to each of the 17 features and to combine these features in non-linear ways to identify any potential defects. The NN can then be used to give a probability of defects for future files. This could help the management team in three ways: (1) to recognize the likeliest modules to have defects and allocate corrective resources effectively; (2) to provide an overview of the riskiest code modules to identify opportunities to re-architect the application; and (3) to understand the risk of deployment in production by an automated code complexity review.

Caveats and Limitations

It is important to note that there are significant differences between the software repositories used in this work and important classes of software acquired by DoD. For example, embedded software used in DoD weapons platforms is typically marked by high complexity, with low tolerance for reliability, availability, safety, and security issues. Although the testbeds on which the ML approaches were applied do contain some NASA software, only a small subset at best of the systems providing data are expected to have similar characteristics. As a result, it is important to view these results as showing a potential method that would be applicable to DoD programs and could learn characteristics of interest within that environment. While the method may be of interest, the specific results summarized may not directly carry over to some types of software present in the DoD environment.

Conclusions

The Rotunda Solutions exploration outlined the potential benefits of harnessing ML/AI throughout the DoD software acquisition life cycle. These benefits include increased accuracy of budget predictions, comprehensive planning, mitigation of expensive defects, and transparency. Rotunda Solutions also identified many opportunities and applications that may improve DoD software development and estimation practices.

H.4 Implications of the Study Results for DoD

This ML study demonstrated promising results by creating models with publicly available software project data. It uncovered a promising approach (the APL Life-Cycle Prediction Model) that can be used to develop good predictions of software duration and effort in the early stages of software procurement and development. The study also uncovered another approach (the UC-Davis Forecasting Model) that can further improve project estimates once software development has been underway for 6 months or more. Finally, the Rotunda Solutions defect density model can highlight modules requiring additional resources and risk mitigation efforts.

The generalizability of these models to DoD software projects requires validation. For instance, a pilot study could be conducted with a small subset of DoD projects. Ultimately, strategies can be developed to enable DoD leadership to effectively leverage ML models.

One strategy could entail a strong centralized mandate for DoD software development teams to provide project data to DoD oversight personnel for evaluation with the APL and UC-Davis models.

A second, more streamlined and evolutionary strategy is to provide these models as tools for DoD software development teams to use as part of best practices to guide their development plans. This strategy would alleviate the exchange of data and would allow a more collaborative community effort to refine the models and resulting software development performance over time.

Appendix I: Acronyms and Glossary of Terms

Acronyms

- ACAT - acquisition category
- ACTD - advanced concept technology demonstration
- AI - artificial intelligence
- ATO - authority (or authorization) to operate
- CAPE - Cost Assessment and Performance Evaluation
- CFR - Code of Federal Regulations
- CI/CD - continuous integration/continuous delivery
- CIO - Chief Information Officer
- COCOM - combatant command
- COTS - commercial off-the-shelf
- DAU - Defense Acquisition University
- DDS - Defense Digital Service
- DFARS - Defense Federal Acquisition Regulation Supplement
- DIB - Defense Innovation Board
- DoD - Department of Defense
- DoDI - Department of Defense Instruction
- DoDIG - Department of Defense Office of Inspector General
- DOT&E - Director, Operational Test & Evaluation
- DSB - Defense Science Board
- DSS - Defense Security Service
- FACA - Federal Advisory Committee Act
- FAR - Federal Acquisition Regulation
- FARs - Federal Acquisition Regulations
- FFRDC - Federally-Funded Research and Development Center
- FFRDCs - Federally Funded Research and Development Centers
- FM - financial management
- FTEs - full-time equivalents
- GAO - General Accounting Office
- GOTS - government off-the-shelf
- GPU - graphics processing unit
- IT - information technology
- JCIDS - Joint Capabilities Integration and Development System
- JIDO - Joint Improvised-Threat Defeat Organization
- KO - contracting officer
- ML - machine learning
- MOS - military occupational specialty
- MVP - minimum viable product
- O&M - operations and maintenance
- OODA - Observe, Orient, Decide, and Act
- OSD - Office of the Secretary of Defense
- OT&E - Operation, Test & Evaluation

- OTA - Other Transaction Authority
- PAO - program acquisition office
- PM - program management
- PMO - program management office
- PPB&E - Planning, Programing, Budgeting and Execution
- R&D - research and development
- RDT&E - research, development, test, and evaluation
- RFP - request for proposals
- SAE - Service Acquisition Executive
- SE - systems engineering
- SWAP - software acquisition and practices 9study)
- T&E - Testing & Evaluation
- USC - United States Code
- USD(A&S) - Under Secretary for Defense (Acquisition and Sustainment)
- USD(C) - Under Secretary of Defense (Comptroller)
- USD(R&E) - Under Secretary of Defense (Research and Engineering)

Glossary

In this subsection we provide a short glossary of some of the terms that we use throughout the report. For each term we provide a short definition of that term, including references if it is a term used elsewhere, and then provide some context and motivation for the use of the term in this report.

Agile development [DSB00]**.** Agile development, also called "iterative" development, begins with the creation of a software factory. Development and testing sprints—a set period of time during which specific work is completed—allow a team to do rapid iterations of development, obtain user feedback, and adjust goals for the next increment. This framework allows for continuous development throughout the life of the product.

ATO (authorization to operate). Formal declaration by a Designated Approving Authority (DAA) that authorizes operation of an IT system and explicitly accepts the risk to agency operations. Obtaining an ATO is required under the Federal Information Security Management Act (FISMA) of 2002 and regulated by Federal Government and DoD guidance that specifies the minimum security requirements necessary to protect Information Technology (IT) assets.

Business systems [Sec 1.2]**.** Essentially the same as enterprise systems, but operating at a slightly smaller scale (e.g., for one of the Services). Like enterprise systems, they are interoperable, expandable, reliable, and probably based on commercial offerings. Similar functions may be customized differently by individual Services, though they should all interoperate with DoD-wide enterprise systems. Depending on their use, these systems may run in the cloud, in local data centers, or on desktop computers. Examples include software development environments and Service-specific HR, financial, and logistics systems.

CI/CD (continuous integration/continuous delivery). Continuous integration (CI) is the practice of merging all software developer working copies of code to a shared master development branch on a continuous basis. Continuous delivery is a software engineering approach in which teams produce software in short cycles, ensuring that the software can be reliably released at any time. The combination of continuous integration and continuous delivery is a common feature of DevOps (and DevSecOps) development environments.

Cloud computing [Sec 1.2]**.** Computing that is typically provided in a manner such that the specific location of the compute hardware is not relevant (and may change over time). These systems will typically be running on commercial hardware and using commercial operating systems, and the applications running on them will run even as the underlying hardware changes. The important point here is that the hardware and operating systems are generally transparent to the application and its user.

Client/server computing [Sec 1.2]**.** Computing provided by a combination of hardware resources available in a computing center (servers) as well as local computing (client). These systems will usually be running on commercial hardware and using commercial operating systems.

Combat systems [Sec 1.2]**.** Software applications that are unique to the national security space and used as part of combat operations. Combat systems may require some level of customization that may be unique to DoD, not the least of which will be specialized cybersecurity considerations to enable them to continue to function during an adversarial attack. (Note that since modern DoD enterprise and business systems depend on software, cyber attacks to disrupt operations have the potential to be just as crippling as those aimed at combat systems.)

Desktop/laptop/tablet computing [Sec 1.2]**.** Computing that is carried out on a single system, often by interacting with data sources across a network. These systems will usually be running on commercial hardware and using commercial operating systems.

DevSecOps. "DevOps" represents the integration of software development and software operations, along with the tools and culture that support rapid prototyping and deployment, early engagement with the end user, and automation and monitoring of software, and psychological safety (e.g., blameless reviews). "DevSecOps" is a more recent term that reflects the importance of integrating security into the DevOps cycle (and not bolting on security at the end). DevOps development is closely related to agile development and the two are often used interchangeably. The term DevSecOps places more focus on security as a critical element. More information: https://tech.gsa.gov/guides/understanding_differences_agile_devsecops/.

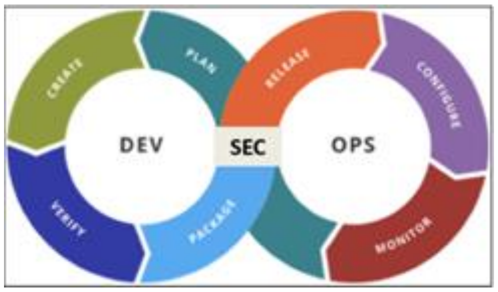


Figure I.1. Continuous integration of development, security, and deployment (DevSecOps). [Adapted from an image by Kharnagy, licensed under CC BY-SA 4.0]

DevSecOps techniques should be adopted by DoD, with appropriate tuning of approaches used by the Agile/DevOps community for mission-critical, national security applications. Open source software should be used when possible to speed development and deployment, and leverage the work of others. Waterfall development approaches (e.g., DOD-STD-2167A) should be banned and replaced with true, commercial agile processes. Thinking of software “procurement” and “sustainment” separately is also a problem: software is never “finished” but must be constantly updated to maintain capability, address ongoing security issues and potentially add or increase performance.

Moving to a DevSecOps software development approach will enable DoD to move from a specify, develop, acquire, sustain mentality to a more modern (and more useful) create, deploy, scale, optimize mentality. Enabling rapid iteration will create a system in which the US can update software at least as fast as our adversaries can change tactics, allowing us to get inside their OODA loop.

Digital Infrastructure. Enterprise-scale computing hardware and software platforms that enable rapid creation and fielding of software. Critical elements include:

- *Scalable compute*: elastic mechanisms to provide any developer with a powerful computing environment that can easily scale with the needs of an individual programmer, a product development team, or an entire organization enterprise.

- *Containerization*: sandbox environments that “package up” an application or microservices with all of the operating system services required for executing the application and allowing that application to run in a virtualized segmented environment.

- *Continuous integration/continuous delivery (CI/CD) pipeline*: platform for automated testing, security, and deployment of software, including licenses access for security tools and a centralized artifacts repository of containers with tools, databases, and operating system images.

- *Automated configuration, updating, distribution, and recovery management:* automated processes that use machine-readable definition files (stored in the same source code repository as your software source code) to manage and provision environments, containers, virtual machines, load balancing, networking, access rules, and other components.

- *Federated identity management and authentication*: common identity management for accessing information across multiple systems and allows rapid and accurate auditing of code.

- *Firewall configuration and network access control lists*: forces information transfer only through intentional interfaces to reduce the attack surface and make system servers more resilient against penetration.

- *Common information assurance (IA) profiles*: Common IA profiles integrated into the development environment and part of the development system architecture are less likely to have bugs than customized and add-on solutions.

- *Modeling and simulation capability:* The use of high fidelity simulations and digital models, enables software developers to develop and validate software more quickly with greater reliability (see also *digital twin*).

Currently, DoD programs each develop their own development and test environments, which requires redundant definition and provisioning, replicated assurance, including cyber, and extended lead times to deploy capability. Digital infrastructure, common in commercial IT, is critical to enable rapid deployment at the speed (and scale) of relevance. The Services and defense contractors will need to build on a common set of tools (instead of inventing their own) *without* just requiring that everyone use one DoD-wide (or even service-wide) platform.

Digital twin. A digital twin is a digital synthetic representation of a system or capability. Digital twins are useful in concept development, designing, developing, testing, and validation of software. The use of high fidelity simulations and digital models enables software developers to develop and validate software more quickly with greater reliability. In the future, as we leverage the use of Machine Learning (ML) and Artificial Intelligence (AI) in software design, development, and test, the ability to leverage simulation and modeling will be critical. For example, today, in the commercial world where self-driving cars are being pioneered, sensors are used to collect data on millions of miles of roads. Before software updates are pushed to the autonomous driving cars and before the first mile is every driven on real roads, the software “drives” those millions of miles through simulation. It is important that the Department and our defense industrial base develop and support similar capabilities to that of commercial industry.

Embedded computing [Sec 1.2]**.** Computing that is tied to a physical, often-customized hardware platform and that has special features that requires careful integration between software and hardware.

Enduring capability. Refers to a class of mission software needs that will persist for the foreseeable future and should be budgeted and managed as an ongoing level of effort with a portfolio management approach to balance—in real time—maintenance, upgrades and major new functionality. An example is the acquisition, processing and distribution of data and information from overhead assets which, when separated from the sensor and satellite programs to which each iteration is traditionally attached, is an area of investment we will always be making.

Throughout this report we make reference to the modern view of software as a continuously, incrementally delivered capability and we use that definition to drive many of the recommendations we propose, especially around the use of DevSecOps. This view is characterized by rapid user feedback loops and continuous deployment to deal with that feedback and with such “maintenance” functions as cyber protection, operating system upgrades, etc. This is the overall vision we espouse for the acquisition and delivery of most types of software—think about the software to deliver spare parts management for a fighter fleet, the software to manage the movement of service personnel and their families, or the software to provide tanker scheduling for a combat air fleet in an AOR.

We believe it is also important to look at certain kinds of software that will need to be delivered against a *mission* need that will persist for long enough into the future that we should think about it as an *enduring capability* need. A good example of an enduring capability is the processing, exploitation, and distribution (PED) software that ingests data from multi-domain overhead assess, processes that data into a series of information products and makes those products available to a wide array of global users. Satellites will change, sensors will change, and the kinds of analyses will change, but the underlying software to process this chain will endure.

Historically PED has been mapped to new or upgraded satellite launches—new satellite, new ground station—and as such are mapped to long cycle times, large, non-incremental programs and oversized budgets broken into the traditional buckets of R&E, acquisition and maintenance. A different model would be to recognize the enduring need for PED capability, fund as a stable ongoing effort, manage the capability through an integrated program team/PEO responsible in real time for the portfolio trades between fixes, upgrades and new capabilities. The core is to separate software from the hardware platforms that provide it data and from the downstream systems that consume the output of the software, recognize that this software need will persist for the foreseeable future, and fund and manage the program in this fashion.

Enterprise systems [Sec 1.2]. Very large-scale software systems intended to manage a large collection of users, interface with many other systems, and generally be used at the DoD level or equivalent. These systems should always run in the cloud and should use architectures that allow interoperability, expandability, and reliability. In most cases the software should be commercial software purchased (or licensed) without modification to the underlying code, but with DoD-specific configuration. Examples include email systems, accounting systems, travel systems, and HR databases.

Logistics systems [Sec 1.2]. Any system that is used to keep track of materials, supplies, and transport as part of operational use (versus Service-scale logistics systems, with which they should interoperate). While used actively during operations, logistics systems are likely to run on commercial hardware and operating systems, allowing them to build on commercial off-the-shelf (COTS) technologies. Platform-based architectures enable integration of new capabilities and functions over time (probably on a months-long or annual time scale). Operation in the cloud or based on servers is likely.

Mission systems [Sec 1.2]. Any system used to plan and monitor ongoing operations. Similar to logistics systems, this software will typically use commercial hardware and operating systems and may be run in the cloud, on local services, or via a combination of the two (including fallback modes). Even if run locally (such as in an air operations center), they will heavily leverage cloud technologies, at least in terms of critical functions. These systems should be able to incorporate new functionality at a rate that is set by the speed at which the operational environment changes (days to months).

Mobile computing [Sec 1.2]. Computing that is carried out on a mobile device, usually connected to the network via wireless communications. These systems will usually be running on commercial operating systems using commodity chipsets.

MVP (minimum viable product). A minimum viable product is a first iteration of a software project that has just enough features to meet basic minimum functionality. It provides the foundational capabilities upon which improvements can be made. The goal of an MVP is to quickly get basic capabilities into users hands for evaluation and feedback.

Security-at-the-perimeter. An approach to security that relies on perimeter access control as the primary mechanism for protecting against intrusion.

Software-defined systems. Software-defined systems make use of the increased capability of digital computing to carry out functions that are traditionally associated with hardware. Examples include software-defined radios and software-defined networking.

Software factory [DSB18]. **A** set of software tools that programmers use to write their code, confirm it meets style and other requirements, collaborate with other members of the programming team, and automatically build, test, and document their progress. This allows teams of programmers to do iterative development with frequent feedback from users.

Technical debt. The cost that is incurred by implementing a software solution that is expedient rather than choosing a better approach that would take longer. Technical debt often accrues over the life of a program as code is expanded and patched. Technical debt can often be “paid down” by investing in refactoring or re-architecting the code.

Unit testing. A software testing method in which software programs, modules, or functions tested to determine whether they satisfy a desired set of specifications, typically by testing a large number of individual tests cases (unit test). Unit testing provides a means of detecting when errors have been inadvertently introduced into a code base.

Weapons system [Sec 1.2]. Any system that is capable of the delivery of lethal force, as well as any direct support systems used as part of the operation of the weapon. Note that our definition differs from the standard DoD definition of a weapons system, which also includes any related equipment, materials, services, personnel, and means of delivery and deployment (if applicable) required for self-sufficiency. The DoD definition would most likely include the mission and logistics functions, which we find useful to break out separately. Software on weapons systems is traditionally closely tied to hardware, but as we move to greater reliability of software-defined systems and distributed intelligence, weapons systems software is becoming increasingly hardware independent (similar to operating systems for mobile devices, which run across many different hardware platforms).

Catch Phrases

Self denial of service attack. Not letting your organization make use of tools or processes that are available to others.

Staple test. Any report that is going to be read should be thin enough to be stapled with a regular office stapler. A standard office stapler is able to staple 25 sheets of paper together => staple test limit is ~50 pages (but you can get a bit more if you bend over the staples manually).

Takeoff test. Reports should be short enough to read during takeoff, before the movies start and drinks are served (assuming you got upgraded). The average time from closing the door to hitting

10,000 ft (wifi on) at IAD is 25 minutes (15 taxi + 10 cruise). Average reading time for a page is 2 minutes => takeoff test limit is ~12 pages.

Waterfall with sprints. A too common approach to implementing agile development principles in a DoD environment. Development teams work on a rapid sprint cycle and deliver code into a test environment that takes months to complete (versus actual agile, where code would be released to users at the end of the spring).

Appendix J: Study Information

Study Membership

CHAIRS	
Richard M. Murray California Institute of Technology	**J. Michael McQuade** Carnegie Mellon University
MEMBERS	
Milo Medin Google	**Jennifer Pahlka** Code for America
Trae' Stephens Founders Fund	**Gilman Louie** Alsop Louie Partners
GOVERNMENT ADVISORS	
Jeff Boleng Special Assistant for Software Acquisition	**Bess Dopkeen (Jan '18-Jan '19)** Cost Assessment/Program Eval
FEDERALLY FUNDED RESEARCH AND DEVELOPMENT CENTERS SUPPORT	
Forrest Shull Software Engineering Institute	**Kevin Garrison** Institute for Defense Analyses
Craig Ulsh MITRE	**Nicolas Henri-Guertin** Software Engineering Institute
DATA ANALYSES AND MACHINE LEARNING SUPPORT	
John Piorkoski Johns Hopkins University Applied Physics Lab	**Steven Lee** Rotunda Solutions
Linda Harrell Johns Hopkins University Applied Physics Lab	**Vladimir Filkov** University of California Davis
Yun Kim Tecolote	**Tom Schaefer** Tecolote
Kevin Jackameit Tecolote	**Dave Zubrow** Software Engineering Institute
STUDY SUPPORT	
Courtney Barno Artlin Consulting	**Devon Hardy** Artlin Consulting
Sandra O'Dea E-3 Federal Solutions	

Software Acquisition and Practices (SWAP) Working Group
Subgroup Participation

Acquisition Strategy
Melissa Naroski Merker (lead)
Larry Asch
Jeff Boleng
Nicolas Chaillan
COL Harry Culclasure
Ben FitzGerald
Nick Kosmidis
Jonathan Mostowski
Nick Tsiopanas

Appropriations
Jane Rathbun (lead)
Maj Gen Patrick Higby
Paul Hullinger
Melissa Naroski Merker
Shannon McKitrick

Contracts
Jonathan Mostowski (lead)

Data & Metrics
Ben FitzGerald (lead)
Jeff Boleng
Victoria Cuff
Bess Dopkeen
Jon Engelbrektson
Mark Krzysko
Melissa Naroski Merker
John Seel
David Zubrow

Infrastructure
Jeff Boleng (co-lead)
John Bergin (co-lead)
Nicolas Chaillan
Victoria Cuff
Robert Gold
Amy Henninger
Richard Kutter

Infrastructure (Cont.)
Jane Rathbun
John Rusnak
Zack Schiller
Philomena Zimmerman

Requirements
Fred Gregory (lead)
Jeff Boleng
Victoria Cuff
Jennifer Edgin
Donald Johnson
Margaret Palmieri
Owen Seely
Philomena Zimmerman

Security & Accreditation
Leo Garciga (lead)
Jeff Boleng
Nicolas Chaillan
Amy Henninger
Maj Gen Patrick Higby
Ana Kreiensieck
Nicolas Lanham
Tom Morton

Sustainment & Modernization
Kenneth Watson (lead)
Stephen Michaluk
Bernard Reger

Testing & Evaluation
Greg Zacharias (lead)
Chad Bieber
Chris DeLuca
Amy Henninger
Lt Col Mark Massaro
Ryan Norman
Tom Simms
Heather Wojton
Philomena Zimmerman

Defense Innovation Board (DIB), Software Acquisition and Practices (SWAP) Study Team and Support Team Site Visits

As part of its data gathering activities, the SWAP study team visited a cross-section of ongoing software programs (both business and weapon systems) across DoD and the Services. Despite their demanding schedules, program managers and their teams (civilian and contractor) welcomed members of the study team and shared their valuable experiences in software acquisition and development, testing, and security. The knowledge gained from these collaborative sessions provided tremendous input into the study and the development of the final recommendations. The SWAP study team would like to thank all those individuals who participated in these site visits for their invaluable contribution to this study.

Date	Companies/Organizations	Locations
Mar 2018	Lockheed Martin	Fort Worth, TX
Apr 2018	Pivotal, Raytheon	Boston, MA
Aug 2018	Raytheon	Los Angeles; Aurora, CO
Aug 2018	SPAWAR, ARL	Colorado Springs, CO
Sep 2018	Lockheed Martin	Moorestown, NJ
Oct 2018	Leidos, Cerner	Rosslyn, VA
Nov 2018	Raytheon	Tucson, AZ

Special thanks to: Samantha Betting, Richard Calabrese, Tory Cuff, RDML Tom Druggan, Lt Col Thomas Gabriele, Leo Garciga, Jack Gellen, Arturo Gonzalez, Jill Hardash, Brian Henson, Cori Hughes, Lisa Jollay, CAPT Bryan Kroger, Col Jennifer Krolikowski, Lt Col Jason Lee, Myron Liszniansky, Maj Zachary McCarty, Lt Col Steve Medeiros, Kenneth Merchant, Anna Nelson, David Norley, Scott Paulsen, Kelci Pozzi, Sandy Scharn-Stevens, Terry Schooley, Thomas Scruggs, Lt Col Kenneth Thill, and Eric Todd

Government and Supplemental Program Meetings

In addition to conducting site visits, the SWAP study team engaged with a broad spectrum of offices within DoD and the Services that possess ownership of the regulations and policies that relate to the software acquisition and/or development life cycle and their associated challenges. In the spirit of practicing an agile methodology, these regular collaborative sessions resulted in cyclic user feedback. The meetings listed below are not exhaustive, but we aimed to capture the wide array of offices that provided feedback to the SWAP study team, highlighting the myriad and assorted offices within DoD that are intertwined with software.

3 – 4 APRIL 2019 MEETINGS

Office of the Under Secretary of Defense (Acquisition & Sustainment)

Office of the Under Secretary of Defense (Comptroller) & Chief Financial Officer

Department of Defense Chief Information Officer

Office of the Director, Operational Test and Evaluation

Office of the Secretary of the Air Force/AQR - Science, Technology, and Engineering

Assistant Secretary of the Navy for Research, Development and Acquisition
Office of the Chief of Naval Operations

Office of the Chief Information Officer (CIO)/G6, Department of the Army

20 – 22 MARCH 2019 MEETINGS

Office of the Secretary of Defense

Office of the Under Secretary of Defense (Acquisition & Sustainment)

Office of the Director, Operational Test and Evaluation

Cost Assessment and Program Evaluation

Office of Personnel & Readiness

Office of the Under Secretary of Defense (Research & Engineering)

Department of Defense Chief Information Officer

4 – 6 DECEMBER 2018 MEETINGS

Office of the Under Secretary of Defense (Comptroller) & Chief Financial Officer

Office of the Director, Operational Test and Evaluation

19 NOVEMBER 2018 MEETING

Supplemental Program Session

Lockheed Martin

Air Force Life Cycle Management Center
Office of the Secretary of the Air Force/Acquisition

24 SEPTEMBER 2018 MEETING

Office of the Under Secretary of Defense (Acquisition & Sustainment)

Office of the Under Secretary of Defense (Research & Engineering)

2 OCTOBER 2018 MEETING

Supplemental Program Session

Lockheed Martin

Naval Sea Systems Command

17 AUGUST 2018 MEETING

Air Force Materiel Command/Air Force Life Cycle Management Center

27 – 28 FEBRUARY 2019 MEETINGS

Joint Rapid Acquisition Center

Office of Personnel & Readiness

Defense Digital Services

Defense Security Cooperation Agency

17 – 18 JANUARY 2019 MEETINGS

Office of the Under Secretary of Defense (Acquisition & Sustainment)

Office of the Under Secretary of Defense (Research & Engineering)

Department of Defense Chief Information Officer

Office of the Under Secretary of Defense (Comptroller) & Chief Financial Officer

Cost Assessment and Program Evaluation

Office of the Chief Management Officer

Office of the Secretary of the Air Force for Acquisition, Technology, & Logistics

Office of the Secretary of the Navy
Office of the Assistant Secretary of the Navy for Research, Development, & Acquisition - Command, Control, Communications, Computers, Intelligence

Representatives from Industry

23 JULY 2018 MEETING

U.S. Navy

Naval Air Warfare Center Weapons Division

3 JULY 2018 MEETINGS

Congressional Research Service

U.S. Army Contracting Command

Charge from Congress

2018 NATIONAL DEFENSE AUTHORIZATION ACT

SEC. 872. DEFENSE INNOVATION BOARD ANALYSIS OF SOFTWARE ACQUISITION REGULATIONS.

(a) STUDY.—

(1) IN GENERAL.—Not later than 30 days after the date of the enactment of this Act, the Secretary of Defense shall direct the Defense Innovation Board to undertake a study on streamlining software development and acquisition regulations.

(2) MEMBER PARTICIPATION.—The Chairman of the Defense Innovation Board shall select appropriate members from the membership of the Board to participate in the study, and may recommend additional temporary members or contracted support personnel to the Secretary of Defense for the purposes of the study. In considering additional appointments to the study, the Secretary of Defense shall ensure that members have significant technical, legislative, or regulatory expertise and reflect diverse experiences in the public and private sector.

(3) SCOPE.—The study conducted pursuant to paragraph (1) shall—

(A) review the acquisition regulations applicable to, and organizational structures within, the Department of Defense with a view toward streamlining and improving the efficiency and effectiveness of software acquisition in order to maintain defense technology advantage;

(B) review ongoing software development and acquisition programs, including a cross section of programs that offer a variety of application types, functional communities, and scale, in order to identify case studies of best and worst practices currently in use within the Department of Defense;

(C) produce specific and detailed recommendations for any legislation, including the amendment or repeal of regulations, as well as non-legislative approaches, that the members of the Board conducting the study determine necessary to—

(i) streamline development and procurement of software;

(ii) adopt or adapt best practices from the private sector applicable to Government use;

(iii) promote rapid adoption of new technology;

(iv) improve the talent management of the software acquisition workforce, including by providing incentives for the recruitment and retention of such workforce within the Department of Defense;

(v) ensure continuing financial and ethical integrity in procurement; and

(vi) protect the best interests of the Department of Defense;
and

(D) produce such additional recommendations for legislation as such members consider appropriate.

(4) ACCESS TO INFORMATION.—The Secretary of Defense shall provide the Defense Innovation Board with timely access to appropriate information, data, resources, and analysis so that the Board may conduct a thorough and independent analysis as required under this subsection.

(b) REPORTS.—

(1) INTERIM REPORTS.—Not later than 150 days after the date of the enactment of this Act, the Secretary of Defense shall submit a report to or brief the congressional defense committees on the interim findings of the study conducted pursuant to subsection (a). The Defense Innovation Board shall provide regular updates to the Secretary of Defense and the congressional defense committees for purposes of providing the interim report.

(2) FINAL REPORT.—Not later than one year after the Secretary of Defense directs the Defense Advisory Board to conduct the study, the Board shall transmit a final report of the study to the Secretary. Not later than 30 days after receiving the final report, the Secretary of Defense shall transmit the final report, together with such comments as the Secretary determines appropriate, to the congressional defense committees.

[redacted]

THE UNDER SECRETARY OF DEFENSE
3010 DEFENSE PENTAGON
WASHINGTON, DC 20301-3010

APR 05 2018

ACQUISITION
AND SUSTAINMENT

MEMORANDUM FOR CHAIRMAN, DEFENSE INNOVATION BOARD

SUBJECT: Terms of Reference - Establishment of the Software Acquisition and Practices Subcommittee of the Defense Innovation Board

Today's advances in software are pushing new frontiers in lethality, speed, precision, accuracy, and efficiency. The Department of Defense's (DoD) ability to field and sustain weapon systems will increasingly depend on its ability to upgrade, develop, and deploy software or acquire commercial software. The technology and business of software development has undergone a radical transformation over the last decade, yet DoD's approach to assess and acquire commercial off-the-shelf software (COTS) products, use and improve existing Government off-the-shelf (GOTS) software, further develop commercial software products to meet unique government needs, or independently develop software products has changed little. This stymies progress and represents significant risk. Software is increasingly the decisive factor in determining the capabilities of modern weapon systems and is often the limiting factor for integrating sensors, platforms, and weapons. For these reasons, an analysis of the DoD's software development and acquisition practices across the range of business and weapon systems is urgently needed as part of the DoD's broader efforts at modernization and reform.

Modernizing the DoD's approach to software development and acquisition has the potential to accelerate fielding of new capabilities, reduce cost, and increase the lethality of our forces. Failure to modernize also carries costs, perpetuating the often slow, unwieldly, requirements-driven approach to software that no longer serves the warfighter or taxpayer well. Moreover, as the field of artificial intelligence progresses, employing rapid, iterative software development, as well as leveraging COTS and GOTS alternatives, will provide critical warfighting capabilities and competitive advantages.

Section 872 of the National Defense Authorization Act (NDAA) for Fiscal Year (FY) 2018 (Public Law 115-91), requires the Secretary of Defense to direct the Defense Innovation Board (DIB) to undertake a study on streamlining software development and acquisition regulations. The Secretary of Defense delegated this authority to the undersigned on 2 January 2018. As such, I am establishing the Software Acquisition and Practices (SWAP) Subcommittee of the DIB to undertake a data-driven analysis of how DoD develops, acquires, and employs software technologies and capabilities.

The NDAA for FY 2018 stipulates that the study must:
(1) Review the acquisition regulations applicable to, and organizational structures within, DoD with a view toward streamlining and improving the efficiency and effectiveness of software acquisition in order to maintain defense technology advantage;
(2) Review ongoing software development and acquisition programs, including a cross section of programs that offer a variety of application types, functional communities, and scale, in order to identify case studies of best and worst practices currently in use within DoD;

[redacted]

[redacted]

(3) Produce specific and detailed recommendations for any legislation, including the amendment or repeal of regulations, as well as non-legislative approaches, that the members of the Board conducting the study determine necessary to–
- (a) Streamline development and procurement of software;
- (b) Adopt or adapt best practices from the private sector applicable to Government use;
- (c) Promote rapid adoption of new technology;
- (d) Improve the talent management of the software acquisition workforce, including by providing incentives for the recruitment and retention of such workforce within DoD;
- (e) Ensure continuing financial and ethical integrity in procurement; and
- (f) Protect the best interests of DoD; and

(4) Produce such additional recommendations for legislation as such members consider appropriate.

The SWAP Subcommittee will provide its recommendations for any legislation, including the amendment or repeal of regulations, and actions to be considered by DoD to the DIB for full and thorough public deliberation and approval. The DIB will submit an interim report to my office not later than May 11, 2018, and a final report not later than April 5, 2019, reporting directly back to me on the study's progress as appropriate.

In conducting its work, the DIB and its subcommittees have my full support in all requests for information, data, resources, and analysis that may be relevant to its research and fact-finding under this Terms of Reference so that the DIB may conduct a thorough and independent analysis as required by section 872 of the NDAA for FY 2018. As such, the Office of the Secretary of Defense, Component Heads, and the Military Departments are directed to promptly facilitate the work of the DIB and the SWAP Subcommittee by ensuring that the DIB staff and members have timely access to any relevant personnel and information necessary to perform their duties consistent with the requirements and limitations of existing law that may be applicable.

As a subcommittee of the DIB, the SWAP Subcommittee shall not work independently of the DIB's charter and shall report its recommendations to the full DIB for public deliberation and approval, pursuant to the Federal Advisory Committee Act of 1972, as amended, the Government in the Sunshine Act of 1976, as amended, and other applicable Federal statutes and regulations. The SWAP Subcommittee does not have the authority to make decisions on behalf of the DIB nor can it report directly to any Federal representative. The members of the SWAP Subcommittee and the DIB are subject to title 18, United States Code, section 208, which governs conflicts of interest.

EllenMLord

Ellen M. Lord

[redacted]

CONTACT

Jeff Boleng | SWAP Study Program Manager
jeffrey.l.boleng.civ@mail.mil | Office (703) 571-9029

SWAP Study Staff
Courtney Barno | courtney.e.barno.ctr@mail.mil | (703) 614-5147
Devon Hardy | devon.k.hardy2.ctr@mail.mil | (703) 614-5148
Sandy O'Dea | Sandra.l.odea.ctr@mail.mil | (703) 614-5139