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:

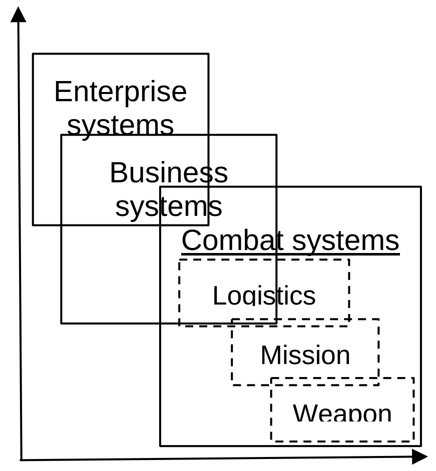


Figure 1.1. Different types of software.

- *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.

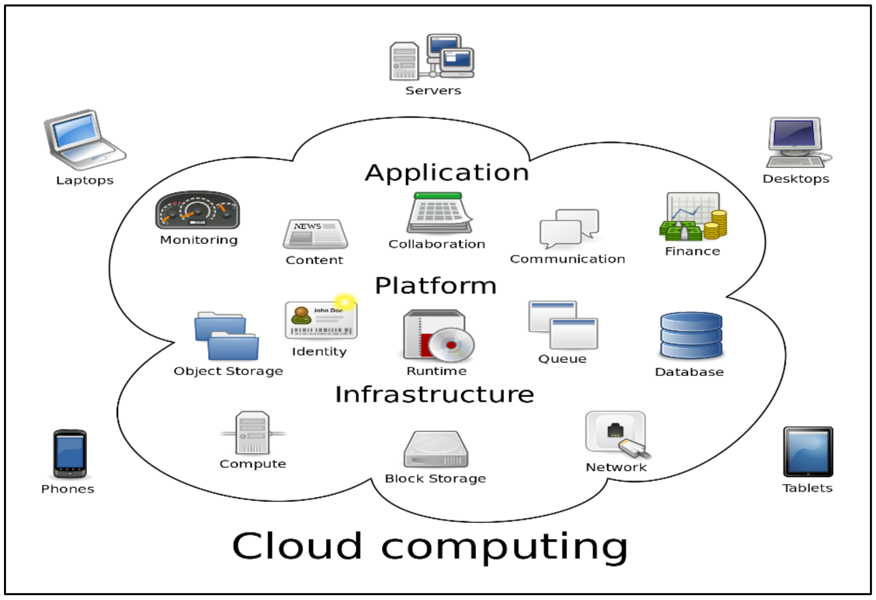


Figure 1.2. Cloud computing environment.
[Image by Sam Johnston is licensed under CC BY-SA 3.0]

- *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).

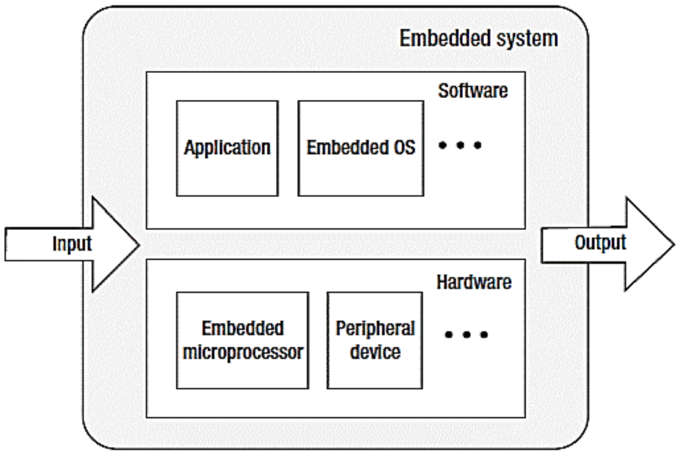


Figure 1.3. Embedded system architecture.
[Image from Ebrary.net]

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).

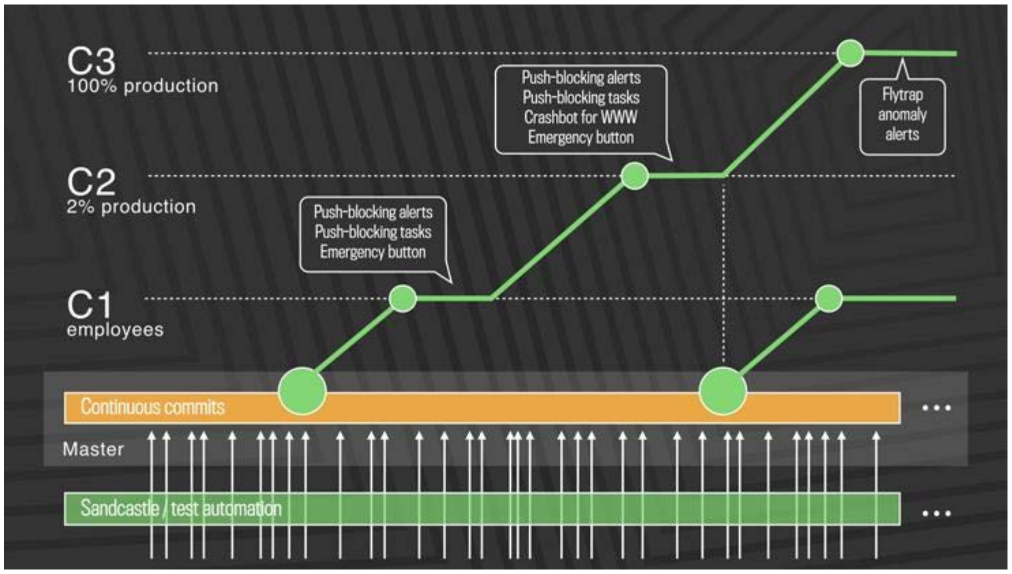


Figure 1.4. Facebook's continuous delivery process. Code updates that have passed a series of automated internal tests (bottom) land in the master development branch and are pushed out to employees (C1). In this stage, push-blocking alerts are generated if there are problems, and an emergency stop button 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

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

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]

[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.

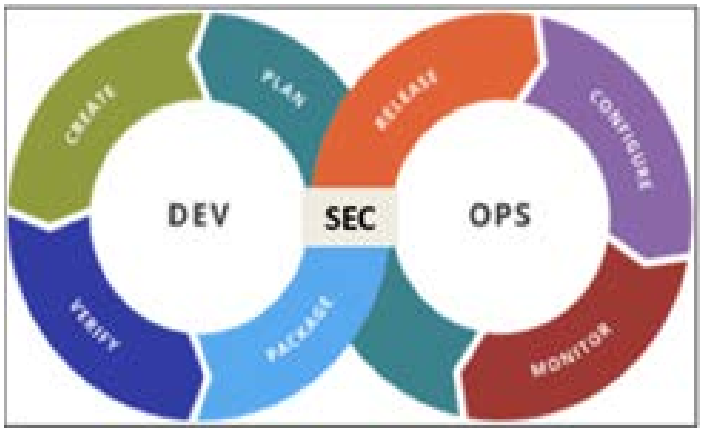


Figure 2.2. Continuous integration of development, security, and deployment (DevSecOps). [Adapted from an image by Kharnagy, licensed under CC BY-SA 4.0]

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.

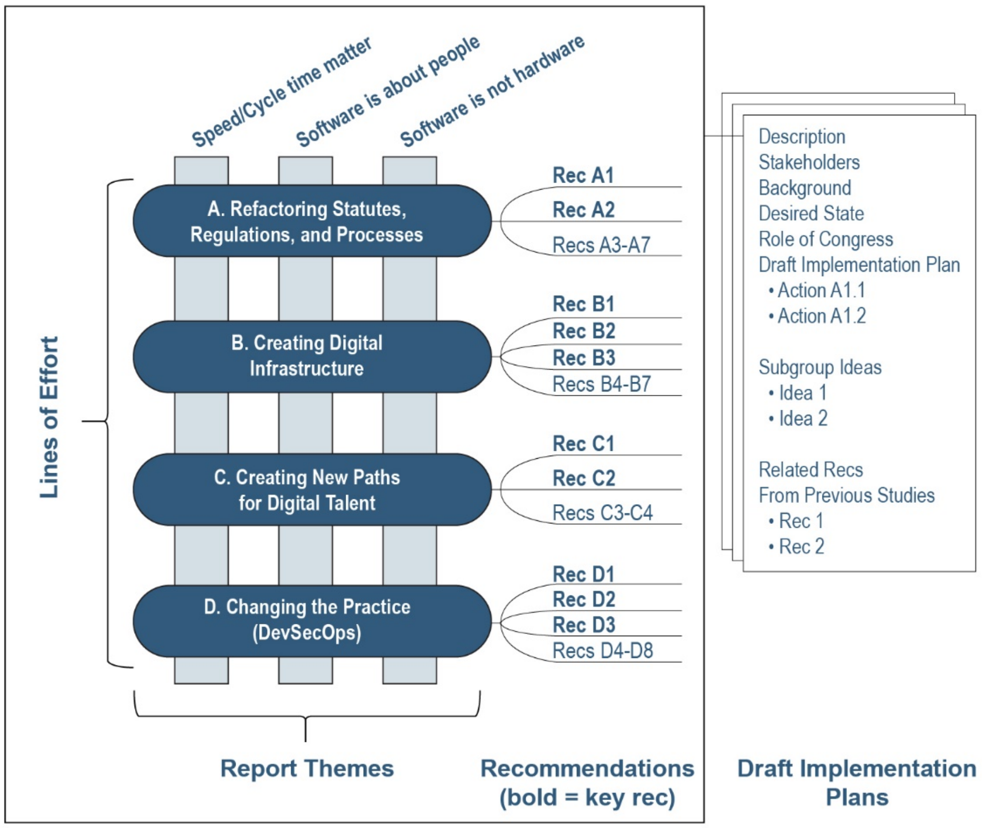


Figure 5.1 Recommendation structure. For each line of effort, a set of primary recommendations (bold) is provided, along with a set of additional recommendations for consideration. Each recommendation contains a draft implementation plan that includes background information on the rationale, vision,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

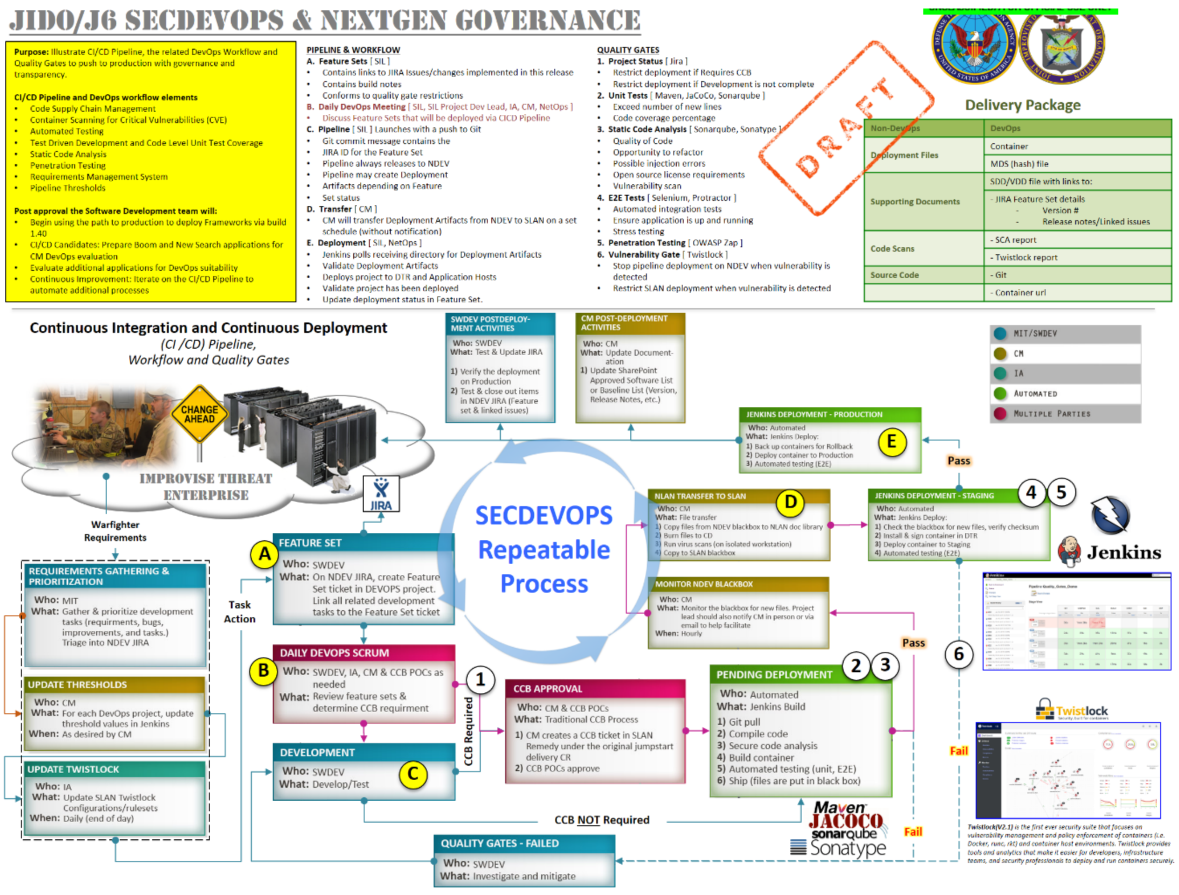
Despite these successes at bringing together stakeholders within the JIDO team, stakeholders in the program management office (PMO) sometimes describe substantial difficulties in working with the rest of the acquisition ecosystem, since on many dimensions the Agile/DevSecOps approach does not work well 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

●	MIT/SWDEV
●	CM
●	IA
●	AUTOMATED
●	MULTIPLE PARTIES

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)

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

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

C DEVELOPMENT
Who: SWDEV
What: Develop/Test

1 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

2 3 PENDING DEPLOYMENT
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

D NLAN TRANSFER TO SLAN
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

4 5 JENKINS DEPLOYMENT - STAGING
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)

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

SWDEV POSTDEPLOYMENT 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 Documentation
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.

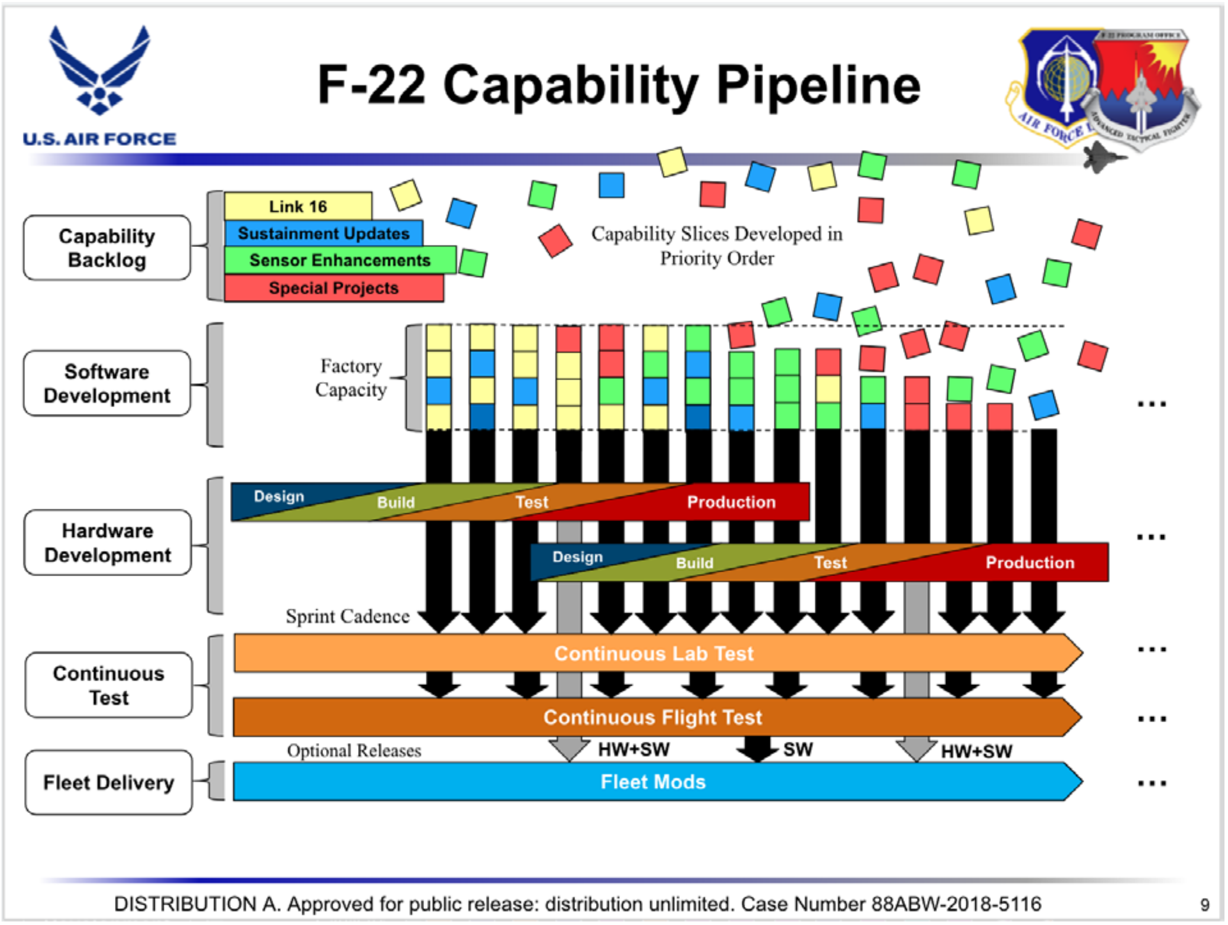
Contracts. The single most significant observation is the failure to prioritize.

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

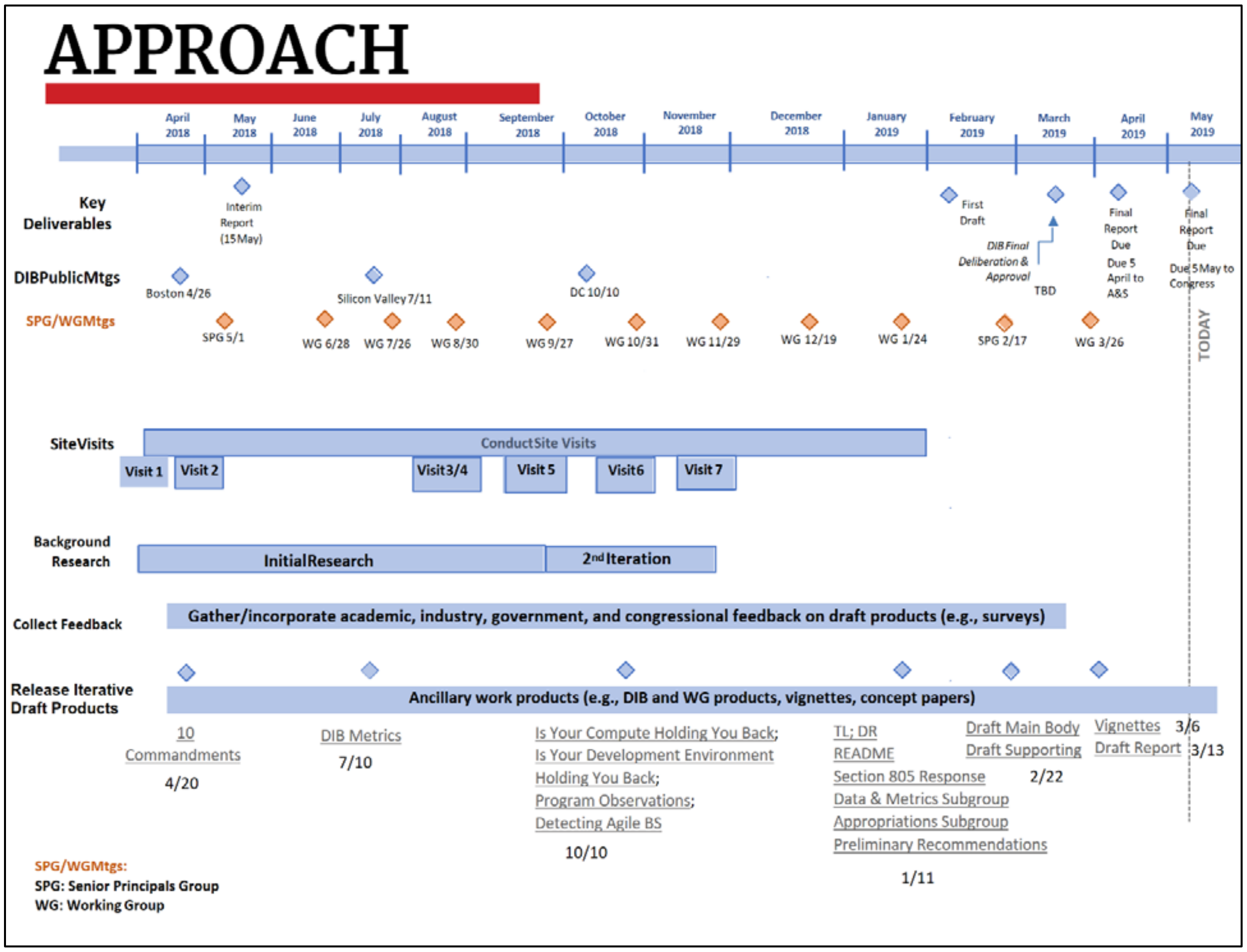
[12] The DIB first heard this term from one of the military instructors at the Air Force Academy and we now use it all the time.

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.

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.

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

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.

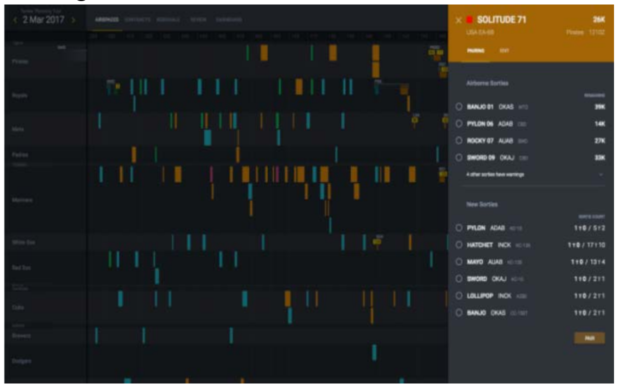
In a similar vein, Secretary of the Air Force Heather Wilson submitted a report to Congress in October 2018 that described Kessel Run's achievements to date. She wrote "The use of Agile DevOps methodologies … is proving successful and we are able to rapidly deliver cloud native applications that increase 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.

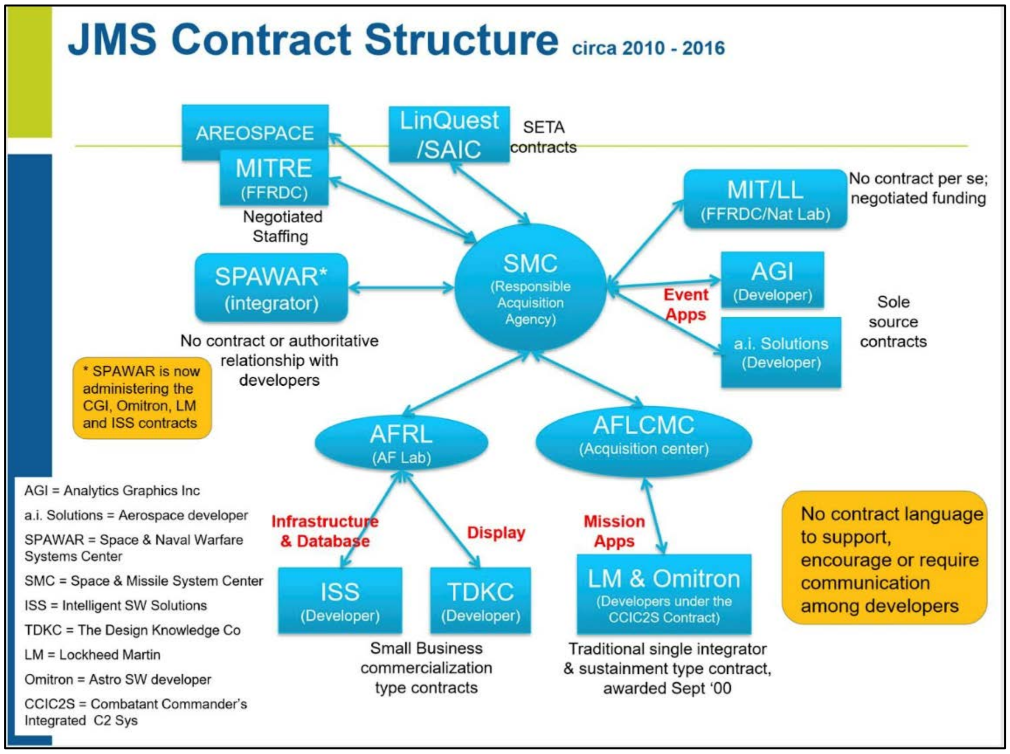
While there were many issues that led to the failure of the JMS program, the following seven are ones that are not a function of that program *per se*, but rather of the process that created it. We thus call these out as general things to look for as indications that your software (program) may be in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