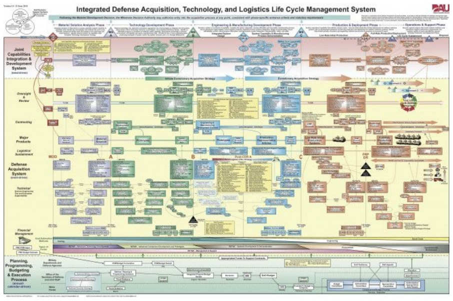
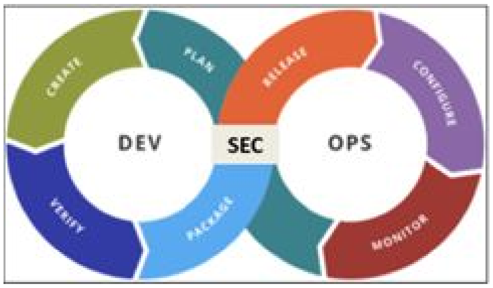
# Defense Innovation Board Do's and Don'ts for Software

Authors: Michael McQuade, Milo Medin, Richard Murray

Version 0.7, last modified 1 Nov 2018

This document provides a summary of the Defense Innovation Board's (DIB's) observations on software practices in the DoD and a set of recommendations for a more modern set of acquisition and development principles. These recommendations build on the DIB Ten Commandments of Software. In addition, we indicate some of the specific statutory, regulatory, and policy obstacles to implementing modern software practices that need to be changed.

## Executive Summary

| Observed practice (Don'ts) | Desired state (Do's) | Obstacles |
|---|---|---|
|  Integrated Defense Acquisition, Technology, and Logistics Life Cycle Management System<br>Defense Acquisition University, June 2010 |  CREATE<br>PLAN<br>RELEASE<br>CONFIGURE<br>DEV<br>SEC<br>OPS<br>VERIFY<br>PACKAGE<br>MONITOR<br>https://commons.wikimedia.org/wiki/File:Devops-toolchain.svg<br>(modifications licensed CC-BY-SA) | 10 U.S.C. §2334<br>10 U.S.C. §2399<br>10 U.S.C. §2430<br>10 U.S.C  §2433a<br>10 U.S.C. §2460<br>10 U.S.C. §2464<br>DODI 5000.02, par 5.c.(2) and 5.c.(3)(c)-(d) |
| Spend 2 years on excessively detailed requirements development | Require developers to meet with end users, then start small and iterate to quickly deliver useful code | DODI 5000.02, par 5.c.(2)<br>CJCSI 3170.01I App A.1.b |
| Define success as 100% compliance with requirements | Accept 70% solutions[1] in a short time (months) and add functionality in rapid iterations (weeks) | 10 U.S.C. §2399<br>OMB Cir A-11 pp 42-43 |
| Require OT&E to certify compliance after development and before approval to deploy | Create automated test environments to enable continuous (and secure) integration and deployment to shift testing left | 10 U.S.C. §139b/d<br>10 U.S.C. §2399<br>Cultural |
| Apply hardware life-cycle management processes to software | Take advantage of the fact that software is essentially free to duplicate, distribute, and modify | 10 U.S.C. §2334<br>10 U.S.C. §2399<br>10 U.S.C. §2430 |

[1] 70% is notional. The point is to deliver the simplest, most useful functionality to the warfighter quickly.

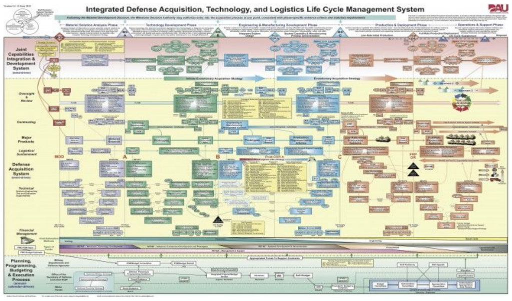
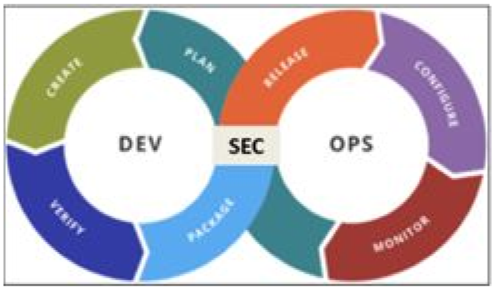
*Acronyms defined*: Operational Test and Evaluation (OT&E); Joint Capabilities Integration and Development System (JCIDS); Apps is short for applications; Specs is short for specifications.

| | | |
|---|---|---|
| | | 48 CFR 207.106<br>DODI 5000.02 |
| Require customized software solutions to match DoD practices | For common functions, purchase existing software and change DoD processes to use existing apps | Culture |
| Use legacy languages and operating systems that are hard to support and insecure | Use modern software languages and operating systems (with all patches up-to-date) | 10 U.S.C. §2334<br>DoDI 5000.02, Enclosure 11<br>Culture |
| Evaluate cyber security after the systems have been completed, separately from OT&E | Use validated software development platforms that permit continuous integration & evaluation (DevSecOps) | DOT&E Memos<br>Culture |
| Consider development and sustainment of software as entirely separate phases of acquisition | Treat software development as a continuous activity, adding functionality across its life cycle | 10 U.S.C. §2399<br>10 U.S.C. §2430<br>10 U.S.C. §2460<br>10 U.S.C. §2464<br>DODI 5000.02, par 5.c.(2) and 5.c.(3)(c)-(d) |
| Depend almost entirely on outside vendors for all product development and sustainment | Require source code as a deliverable on all purpose-built DoD software contracts. Continuous development and integration, rather than sustainment, should be a part of all contracts. DoD personnel should be trained to extend the software through source code or API access[2] | Culture<br>(no apparent statutory obstacle)<br>FAR/DFARS technical data rights |
| Turn documents like this into a process and enforce compliance | Hire competent people with appropriate expertise in software to implement the desired state and give them the freedom to do so ("competence trumps process") | Culture |

[2] As noted in the DIB's 10 Commandments of Software

## Supporting Information

The information below, broken out by entry in the executive summary table, provides additional information and a rationale for each desired state.

| Don’t | Do |
|---|---|
| <br> Integrated Defense Acquisition, Technology, and Logistics Life Cycle Management System<br>Defense Acquisition University, June 2010 | <br> CREATE<br>PLAN<br>RELEASE<br>CONFIGURE<br>DEV<br>SEC<br>OPS<br>VERIFY<br>PACKAGE<br>MONITOR<br>https://commons.wikimedia.org/wiki/File:Devops-toolchain.svg |

The DoD 5000 process, depicted on the left, provides a detailed DoD process for setting requirements for complex systems and ensuring that delivered systems are compliant with those requirements. The DoD’s “one size fits all” approach to acquisition has attempted to apply this model to software systems, where it is wholly inappropriate. Software is different than hardware. Modern software methods make use of a much more iterative process, often referred to as “DevOps,” in which development and deployment (operations) are a continuous process, as depicted on the right. A key aspect of DevOps is continuous delivery of improved functionality through interaction with the end user.

Why this is hard to do, but also worth doing:[3]

- DoD 5000 is designed to give OSD, the Services, and Congress some level of visibility and oversight into the development, acquisition, and sustainment of large weapons systems. While this directive may be useful for weapons systems with multi-billion dollar unit costs, it does not make sense for most software systems.

- While having one consistent procurement process is desirable in many cases, the cost of using that same process on software is that software is delivered late to need, costs substantially more than the proposed estimates, and cannot easily be continuously updated and optimized.

- Moving to a software development approach will enable the DoD to move from a *specify, develop, acquire, sustain* mentality to a more modern (and more useful) *create, scale, optimize* (DevOps/DevSecOps) mentality. Enabling rapid iteration will create a system in

[3] These comments and the similar ones that follow for other area were obtained by soliciting feedback on this document from people familiar with government acquisition processes and modern software development environments.

*Acronyms defined*: Office of the Secretary of Defense (OSD), OODA is short for the the decision cycle of Observe, Orient, Decide, and Act.

which the US can update software at least as fast as our adversaries can change tactics, allowing us to get inside their OODA loop.

| **Don't** | **Do** |
|---|---|
| Spend 2 years on excessively detailed requirements development | Require developers to meet with end users, then start small and iterate to quickly deliver useful code |
| Define success as 100% compliance to requirements | Accept 70% solutions in a short time (months) and add functionality in rapid iterations (weeks) |

Developing major weapons systems is costly and time consuming, so it is important that the delivered system meets the needs of the user. The DoD attempts to meet these needs with a lengthy process in which a series of requirements are established, and a successful program is one that meets those requirements (ideally close to the program's cost and schedule estimates). Software, however, is different. When done right, it is easy to quickly deploy new software that improves functionality and, when necessary, rapidly rollback deployed code. It is more useful to get something simple working quickly (time-constrained execution) and then exploit the ability to iterate rapidly in order to get the remaining desired functionality (which will often change in any case, either in response to user needs or adversarial tactics).

Why this is hard to do, but also why it is worth doing:

- Global deployment of software on systems which are not always network-connected (e.g., an aircraft carrier or submarine underway) introduces very real problems around version management, training, and wisely managing changes to mission critical systems.

- In the world of non-military, consumer Internet applications, it is easy to glibly talk about continuous deployment and delivery. In these environments, it is easy to execute and the consequences for messing up (such as making something incredibly confusing or hard to find) are minor. The same is not always true for DoD systems -- and DoD software projects rarely offer scalable and applicable solutions to address the need for continuous development.

- Creating an approach (and the supporting platforms) that enables the DoD to achieve continuous deployment is a non-trivial task and will have different challenges than the process for a consumer Internet application. The DoD must lay out strategies for mitigating these challenges. Fortunately, there are tools that can be build upon: many solutions have already been developed in consumer industries that require failsafe applications with security complexities.

- Continuous deployment depends on the entire ecosystem, not just the front-end software development.

- Make sure to focus on product design and *product* management, which prioritizes delivery of capability to meet the changing needs of users, rather than program/project management, which focus on execution against a pre-approved plan. This shift is key to user engagement, research, and design.

| Don’t | Do |
|---|---|
| Require OT&E to certify compliance after development and before approval to deploy | Create automated test environments to enable continuous (and secure) integration and deployment to shift testing left |
| Evaluate cyber security after the system has been completed, separately from OT&E | Use validated software development platforms that permit continuous integration and evaluation |

Why this is hard to do, but also worth doing:

- The DoD typically performs a cyber evaluation on software only after delivery of the initial product. Modern software approaches have not always explicitly addressed cyber security (though this is changing with “DevSecOps”). This omission has given DoD decision-makers an easy “out” for dismissing recommendations (or setting up roadblocks) for DevOps strategies like continuous deployment. Cyber security concerns must be addressed head on, and in a manner that demonstrates better security in realistic circumstances. Until then, change is unlikely.
- More dynamic approaches to address the cyber security concerns must be developed and implemented through some amount of logic and a fair bit of data. Case studies of red teaming also help: *Hack the Pentagon* should be able to provide some true examples that generate concern. It may be necessary to obtain access to some additional good data that goes beyond what corporations are willing to share publicly.
- To succeed, it will be important not to assume that it will be clear how these recommendations solve for all cyber security concerns. Recommendations should make explicit statements about what can be accomplished, taking away the reasons to say "no."

| Don’t | Do |
|---|---|
| Apply hardware life cycle management processes to software | Take advantage of the fact that software is essentially free to duplicate, distribute, and modify |
| Consider development and sustainment of software as entirely separate phases of acquisition | Treat software development as a continuous activity, adding functionality across its life cycle |

Why this is hard to do, but also worth doing:

- Program of record funding is specifically broken out into development and sustainment. These distinct categories of appropriations lead program managers and acquisition professionals to the conclusion that new functionality can only be added within development contracts and that money allocated for sustainment cannot be used to add new features. Vendor evaluation for development and sustainment contracts are different; vendors on sustainment contracts often do not have the same development competencies and frequently are not the people who built the original system. To create an environment that will support a DevOps/DevSecOps approach, DoD Commands and

Services should jointly own the development and maintenance of software with contractors who provide more specialized capabilities. Contracts for software should focus on developing and deploying software (to operations) over the long term, rather than the typical, sequential approach - “acquiring” software followed by “sustaining” that software.

| **Don’t** | **Do** |
|---|---|
| Require customized software solutions to match DoD practices | For common functions, purchase existing software and change DoD processes to use existing apps |

Business processes, financial, human resources, accounting and other “enterprise” applications in the DoD are generally not more complicated nor significantly larger in scale than those in the private sector. Commercial software, unmodified, should be deployed in nearly all circumstances. Where DoD processes are not amenable to this approach, those processes should be modified, not the software. Doing so allows the DoD to take advantage of the much larger commercial base for common functions (e.g., Concur has 25M active users for its travel software).

| **Don’t** | **Do** |
|---|---|
| Use legacy languages and operating systems that are hard to support and insecure | Use modern software languages and operating systems (with all patches up-to-date) |

Modern programming languages and software development environments have been optimized to help eliminate bugs and security vulnerabilities that were often left to programmers to avoid (an almost impossible endeavor). Additionally, outdated operating systems are a major security vulnerability and the DoD should assume that any computer running such a system will eventually be compromised.[4] Standard practice in industry is to apply security patches within 48 hours of release, though even this is probably too big a window for defense systems. Treat software vulnerabilities like perimeter defense vulnerabilities: if there is a hole in your perimeter and people are getting in, you need to patch the hole quickly and effectively.

Why this is hard to do, but also worth doing:

- DoD looks at the cost of upgrading hardware as a major cost that is tied to “modernization.” But hardware should be thought of as a consumable like any other, such as fuel and parts, that must be continually replaced for a weapon system to maintain operational capability. This change would require DoD to provide a stable annual budget that paid for new hardware and software capability.

[4] See the DIB 10 Commandments of Software supporting thoughts and recommendations. “*Move to a model of continuous hardware refresh in which computers are treated as a consumable with a 2-3 year lifetime.”*

- The advantage of using modern hardware and operating systems on DoD systems are manifold: better security, better functionality, reduced (unit) costs, and lower overall maintenance costs.

| Don’t | Do |
|---|---|
| Turn documents like this into a process and enforce compliance | Hire competent people with appropriate expertise in software to implement the desire state and give them the freedom to do so (“competence trumps process”) |

Why this is hard to do, but also why it is worth considering doing it:

- Good engineers want to build things, not just write and evaluate contracts. If their jobs are mainly contracting or monitoring, their software skills will quickly become outdated. This can be solved in the short term by a rotational program: do not allow programmers to stay in contracting for more than 4 years, so their technical capabilities are current.
- The government must team with commercial companies to ensure that it has access to the collection of talent required to develop modern software systems, as well as develop internal talent. The DoD should increase its use of contractors whose aim is not just to provide software, but to increase the software development capabilities and competency of the department. By making use of enlisted personnel, reservists, contractors, and other resources, it is possible to create and maintain highly effective teams who contribute to national security through software development.

## Additional Obstacles

In addition to the specific obstacles listed above, we capture here a collection of statutes, regulations, processes and cultural norms that are impediments to implementing a modern set of software acquisition and development principles.

### Statutes

The statutes below provide examples of impediments to the implementation of modern software development practices in DoD systems.

*Acquisition strategy* (10 U.S.C  §2431a): 2431a(d) establishes the review process for major defense acquisition programs and is written around the framework of waterfall development for long timescale, hardware-centric programs.  In particular, this statute establishes decision-gates at Milestone A (entry into technology maturation and risk reduction), Milestone B (entry into system development and demonstration), and entry into full-rate production.  For many software programs this set of terms and approach does not make sense and is incompatible with the ability to deliver capability to the field in a rapid fashion.

*Critical cost growth in major defense acquisition programs* (10 U.S.C. §2433a [Nunn-McCurdy]): 2433 establishes the conditions under which Congress reviews a major program that has undergone critical cost growth and determines with it should continue. By the time a software program hits a Nunn-McCurdy breach it has already gone well past the point where the program should have been terminated and restarted using a different approach. All software procurement programs should start small, be iterative, and build on success − or be terminated quickly.

*Independent cost estimation and cost analysis* (10 U.S.C. §2334)

*Working capital funds* (10 U.S.C. §2208(r)):

- 2+ year lead times from plan to budget does not allow for continuous engineering
- Differentiating software development workload as Research, Development, Test and Engineering (RDT&E), Procurement, or Operations and Maintenance (O&M) is meaningless as there should be no final fielding or sustainment element to continuous engineering
- System-defined program elements hinder the ability to deliver holistic capabilities and enable real-time resource, requirements, performance and schedule trades across systems without significant work.

*Operational Test and Evaluation* (10 U.S.C. §139b/d, 10 U.S.C. §2399): 139 establishes the position of the Director of Operational Test and Evaluation (DOT&E) and requires that person to carry out field tests, under realistic combat conditions, of weapon systems for the purpose of determining the effectiveness and suitability of those systems in combat by typical military users. 2399(a) states that a major defense acquisition program "may not proceed beyond low-rate initial production until initial operational test and evaluation of the program, subprogram, or element is completed". 2399(b)(4) further states that the program many not proceed "until the Director [of Operational Test and Evaluation] has submitted to the Secretary of Defense the report with respect to that program under paragraph (2) and the congressional defense committees have received that report". These are obstacles for DevSecOps implementation of software, where changes should be deployed to the field quickly as part of the (continuous) development process. They are an example of a "tailgate" process for OT&E that impedes our ability to deploy software quickly and drives a set of processes in which OT&E impedes rather than enhances the software development process. Instead of this process, Congress should allow independent OT&E of software to occur in parallel with deployment and also require that OT&E cycles for software match development cycles through the use of automated workflows and test harnesses wherever possible.

Additional issues:

- Testing and evaluation (T&E) must be integrated into the development lifecycle to facilitate DevSecOps, and reduce operations and sustainment (O&S) costs. T&E should be present from requirements setting to O&S

- Programs need persistent and realistic environments that permit continuous, agile testing of all systems (embedded, networked, etc) in a representative SoS environment
- Software environments should be part of the contract deliverables and accessible to T&E, including source code, build tools, test scripts, data

*Definition of a major acquisition program* (10 U.S.C. §2430): The designation of a program as a major acquisition program triggers a set of procedures that are designed for acquisition of hardware. This includes triggering of the DoD Instruction 5000.02, which is currently tuned for hardware systems. An alternative instruction, DoD Instruction 5000.75, is better tuned for software, but can only used for defense *business* systems; it is not valid for "weapons systems".

*Depot level maintenance and repair; core logistics* (10 U.S.C. §2460, 10 U.S.C. §2464): The definitions of maintenance, repair, and logistics are based on an acquisition model that is appropriate for hardware but not well aligned with the operation of modern software. For example, §2464 says that services will "maintain and repair the weapon systems". But software is not maintained, it is *optimized* (with better performance and new functionality) on a *continuous* basis. §2460(b)(1) further states that depot level maintenance and repair "does not include the procurement of major modifications or upgrades of weapon systems that are designed to improve program performance".

Additional issues:

- DoD's challenge in shifting from applying a Hardware (HW) maintenance mindset to Software (SW) hinders DoD's ability to better leverage DoD's organic SW engineering infrastructure to deliver greater capability to the warfighter.
- DoD's acquisition process is not emphasizing an upfront focus on design for software sustainment and a seamless transition to organic software engineering sustainment to reduce the life cycle cost of software and to speed delivery of capability over the life cycle. Such upfront emphasis is critical given the scope, complexity, and mix of the growing software sustainment demand, in the face of persistent affordability concerns.
- DoD's organic software engineering capabilities and infrastructure are critical to national security, but there is limited enterprise visibility of this infrastructure, its capabilities, workload, and resources to leverage it at the enterprise level to deliver greater capability more affordably to the warfighter.

## Regulations

The regulations are the mechanism by which the DoD implements the statutes that govern its operations. They provide additional examples of impediments to the implementation of modern software development practices in DoD systems.

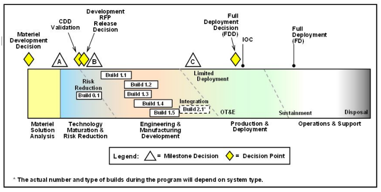
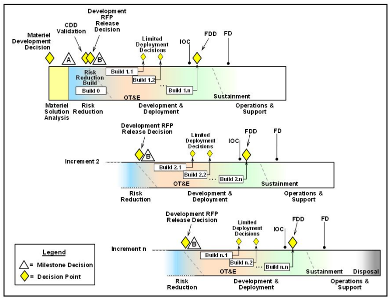
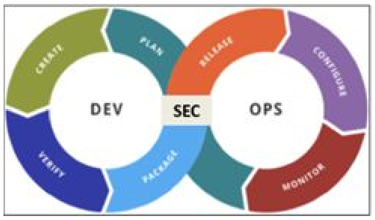
*Cost estimating system requirements* (48 CFR 252.215-7002) : These regulations set out the expectations for estimation of costs of a program against a set of system requirements. While perhaps appropriate for a hardware-oriented system, they do not take into account the type of continuous development cycle that is required to implement modern software.

*Additional requirements for major systems* (48 CFR 207.106): These regulations set out procedures for competition of contracts and are written in a manner that separates out the initial deployment of a system with the operation and sustainment of that system.  This doesn't make sense for software.

## Processes (Instructions)

The detailed processes used to implement the regulations are laid out in Department of Defense Instructions.  We illustrate here some of the specific instructions that are obstacles to implementation of modern software development practices.

*Major acquisition program development process* (DODI 5000.02, par 5.c.(2) and 5.c.(3)(c)-(d)): These portions of the DoD Instructions apply to "Defense Unique Software Intensive" programs and "Incrementally Deployed Software Intensive" programs.  While well-intentioned, they are still waterfall processes with years between the cycles of deployments (instead of weeks).  These processes may be appropriate for some embedded systems, but are not the right approach for DoD-specific software running on commercial hardware and operating systems, as the diagrams below illustrate:

| Definitely not this: | Better, but still not right: | What we need: |
|---|---|---|
| Specify, design, deploy, sustain<br><br> Materiel Development Decision<br>CDD Validation<br>Development RFP Release Decision<br>Full Deployment Decision (FDD)<br>IOC<br>Full Deployment (FD)<br>A<br>B<br>C<br>Risk Reduction<br>Build 0.1<br>Build 1.1<br>Build 1.2<br>Build 1.3<br>Build 1.4<br>Build 1.5<br>Limited Deployment<br>Integration<br>Build 2.1*<br>OT&E<br>Sustainment<br>Disposal<br>Materiel Solution Analysis<br>Technology Maturation & Risk Reduction<br>Engineering & Manufacturing Development<br>Production & Deployment<br>Operations & Support<br>Legend: △ = Milestone Decision ◆ = Decision Point<br>* The actual number and type of builds during the program will depend on system type.<br>DODI 5000.02, Figure 4. Model 2:<br>Defense Unique Software Intensive Program | <br> Materiel Development Decision<br>CDD Validation<br>Development RFP Release Decision<br>Limited Deployment Decisions<br>IOC<br>FDD<br>FD<br>A<br>B<br>Risk Reduction Build<br>Build 0<br>Build 1.1<br>Build 1.2<br>Build 1.n<br>OT&E<br>Sustainment<br>Materiel Solution Analysis<br>Risk Reduction<br>Development & Deployment<br>Operations & Support<br>Increment 2<br>Build 2.1<br>Build 2.2<br>Build 2.n<br>Increment n<br>Build n.1<br>Build n.2<br>Build n.n<br>Disposal<br>Legend<br>△ = Milestone Decision<br>◆ = Decision Point<br>DODI 5000.02, Figure 5. Model 3: Incrementally Deployed Software Intensive Program | Implement, scale, optimize<br><br> CREATE<br>PLAN<br>RELEASE<br>CONFIGURE<br>MONITOR<br>PACKAGE<br>VERIFY<br>DEV<br>SEC<br>OPS<br>https://commons.wikimedia.org/wiki/File:Devops-toolchain.svg<br>(modifications licensed CC-BY-SA) |
| Waterfall development | Waterfall development with overlapping builds | Continuous integration and deployment (DevSecOps) |

*Requirements for programs containing information technology* (DoDI 5000.02, Enclosure 11):  This enclosure attempts to define the requirements for ensuring information security.  It is written under the assumption that the standard waterfall process is being used.

*Preparation, Submission, and Execution of the Budget - Acceptance* (OMB Cir A-11, II.10): This document is the primary document that instructs agencies how to prepare and submit budget requests for OMB review and approval. Section II.10 describes the conditions for acceptance of an acquired item by the government, and requires that the asset meets the requirements of the contract. The impact of this procedure is that it establishes a “100% compliance” mentality in order for the government to accept a software “asset”.

## Culture

In this final section we catalog a list of culture items that do not necessarily require changes in statutes, regulations, or instructions, but rather a change in the way that DoD personnel interpret implement their processes. Changing the culture of DoD is a complex process, depending in large part on incentivizing the behaviors that will lead to the desired state.

Data and metrics

- Multiple, competing, and sometimes conflicting types of data and metrics used, or not used, for assessing software in DOD
- Inability to collect meaningful data about software development and performance in a low cost manner, at scale
- Inability to turn data into meaningful analysis and inability to implement decisions or changes to software activities (L/R/C)

Contracts

- Individual contracts are subject to review processes designed for large programs (of which they are likely enabling). This limits the agility of individual contract actions, even when modular contracting approaches are applied. In addition, the acquisition process is rigid and revolves around templates, boards, and checklists thus limiting the ability for innovation and streamlining execution.
- Contracts focus on technical requirements instead of contractual process requirements. The contract should address overall scope, PoP, and price. The technical execution requirements should be separate and managed by the product owner or other technical lead.
- Intellectual Property (IP) rights are often generically incorporated without considering the layers of technology often applied to a solution. A single solution might include open source, proprietary SW, and government custom code. The IP clauses should reflect all of the technology that is used.

Security Accreditation

- Although developing and operating software securely is a primary concern, the means to achieve and demonstrate security is overly complex and hampered by inconsistent and outdated/misapplied policy and implementation practices (e.g. overlaying historical DoD Information Assurance Certification and Accreditation Process (DIACAP) over risk management framework (RMF) controls for individual pieces of software versus system accreditation). The sense is that the certification and accreditation process is primarily a “check- the-box” documentary process, adds little value to the overall security of the system, and is likely to overlook flaws in the design, implementation, and the environment in which the software operates.

- The DoD needs to be able to calculate the true and component costs for implementing the RMF and certification and accreditation (C&A) in order to identify inefficiencies, duplicative / capabilities, and redundant or overlapping security products and services that are being acquired or developed. Absent a set of metrics it is difficult to prioritize risk areas, investments, and evaluating risk reduction and return on -investment.
- The DoD needs to ensure that each Joint Capability Area (JCA) flow-down its strategy, best practices, and implementation requirements / guidance for security and accreditation to allow the Component responsible for implementing the software to appropriately tailor RMF and plan the development, accreditation, and operation of the software.
- The DoD needs to provide automated tools and services needed to integrate continuous monitoring with the development lifecycle, enable continuous assessment and accreditation, and delegate decision making at the lowest level possible. The DoD should embrace DevSecOps (not just DevOps) and provide policy supported processes, certified libraries, tools, and a toolchain reference implementation to produce “born secure” software

Testing and Evaluation

- The DoD lacks the realistic test environments needed to support test at the pace of modern software methods.
- The DoD lacks the modern software intellectual property (IP) regime needed to support test and evaluation at the pace of modern software methods
- The DoD lack the enterprise knowledge management/data analytics capability needed to support evaluation of test data at the pace of modern software methods

Workforce

- No defined requirements for software developers
- Antiquated policies (talent management, software development)
- Culture and knowledge (DoD, societal, defense contractors)

Appropriations/Funding

- 2+ year lead times from plan to budget does not allow for continuous engineering
- Differentiating software development workload as Research, Development, Test and Engineering (RDT&E), Procurement, or Operations and Maintenance (O&M) is meaningless as there should be no final fielding or sustainment element to continuous engineering.
- System defined program elements hinder the ability to deliver holistic capabilities and enable real-time resource, requirements, performance and schedule trades across systems without significant work.

Infrastructure

- Creating software: The DoD lacks availability of vetted, secure, reusable components, either as source code, or other digital artifacts (think hardened Docker containers or virtual machines (VMs) here). A repository of discoverable, well indexed, vetted, secure, and reusable components could go a long way. This also emphasizes the point that an

awful lot of software now-a-days is software by construction with minimal "glue" code applied.
- Building/managing/testing software: There is a general lack of available tools to build software, especially automated tools (testing/scanning/fuzzing etc.) integrated into a secure pipeline supporting rapid agile development. There is also a significant need to have a common, government owned and managed code repository that all programs could/should/must use (e.g. and government-furnished GitHub).
- Running/hosting software: The DoD needs to continually push the level of abstraction up as much as possible for programs. Traditionally programs, even cloud-based solutions, tend to start at Infrastructure as a Service (IaaS) and build their own rest of the stack. We need secure and available Platform as a Service (PaaS) and Function as a Service (FaaS) so that programs only need to focus on core business logic and not on securing a database or message bus over and over again.
- Operating/updating securely: Once developed and instantiated on a secure and available platform, we need to continually monitor, red team (automated?), and evolve the software. This requires proper instrumentation, logging, and monitoring of the platform, supporting libraries / components, and the core program code. A standard/common way to provide instrumentation and monitoring of the running services built into the infrastructure would be very helpful.

Requirements

- A byproduct of top-level requirement flow down is rigidity and over specificity at the derived requirements level that greatly hinders agile s/w design.
- Too often exquisite requirements are levied on a system that in turn drive extensive complex software requirements and design, affecting development, integration, and system test.
- Data sets are siloed within programs: a common “law of requirements” is that programs of record try to avoid dependencies with other programs of record. This is problematic for software-based capabilities because data is often siloed within single programs of record. We have network programs to "pass" data, but the promise of artificial intelligence (AI), including machine learning (ML), is that software algorithms can leverage pools of data from disparate sources (data lakes).
- By tying software to a program of record, it becomes harder to transfer that code across systems and data environments. As a result, DoD limits code reuse within and across Services.

Modernization and sustainment

- DoD’s challenge in shifting from applying a hardware maintenance mindset to software hinders DoD’s ability to better leverage DoD’s organic software engineering infrastructure to deliver greater capability to the warfighter.
- DoD’s acquisition process is not emphasizing an upfront focus on design for software sustainment and a seamless transition to organic software engineering sustainment to reduce the life cycle cost of software and to speed delivery of capability over the life cycle. Such upfront emphasis is critical given the scope, complexity, and mix of the growing software sustainment demand, in the face of persistent affordability concerns.
- DoD’s organic software engineering capabilities and infrastructure are critical to national security, but there is limited enterprise visibility of this infrastructure, its capabilities,

workload, and resources to leverage it at the enterprise level to deliver greater capability more affordably to the warfighter.

Acquisition Strategy

- Acquisition policy framework: Create a cohesive acquisition policy architecture within which effective, efficient software acquisition policy has a home.
- Acquisition management and governance: Flip the concept of an oversight model on its head.

## Supporting Recommendations

The recommendations above are based on existing assessments and recommendations regarding DoD software acquisition and practices. A brief summary (and links to further information) of materials that provide additional details is provided here.

DIB Ten Commandments (v1.1, May 2018):

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

DSB Design and Acquisition of Software for Defense Systems recommendations (Feb 2018):

- **Rec 1: Software Factory** - A key evaluation criterion in the source selection process should be the efficacy of the offeror’s software factory.
- **Rec 2: Continuous Iterative Development** - The DoD and its defense industrial base partners should adopt continuous iterative development best practices for software, including through sustainment.

- **Rec 3: Risk Reduction and Metrics for New Programs** - For all new programs, starting immediately, the following best practices should be implemented in formal program acquisition strategies.

- **Rec 4: Current and Legacy Programs in Development, Production, and Sustainment** - For ongoing development programs, the Under Secretary of Defense for Acquisition and Sustainment (USD(A&S)) should immediately task the PMs with the PEOs for current programs to plan transition to a software factory and continuous iterative development.  Defense prime contractors should transition execution to a hybrid model, within the constraints of their current contracts. Defense prime contractors should incorporate continuous iterative development into a long-term sustainment plan. The USD(A&S) should immediately task the SAEs to provide a quarterly status update to the USD(A&S) on the transition plan for programs, per the ACAT category.

- **Rec 5: Workforce** - The U.S. Government does not have modern software development expertise in its program offices or the broader functional acquisition workforce. This requires Congressional engagement and significant investment immediately.

- **Rec 6: Software is Immortal – Software Sustainment** - Starting immediately, the USD(R&E) should direct that 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 (e.g., gtest), software tools (e.g., fuzz testing, performance test harnesses), check-in notes, code provenance, and reference and working documents informing development, test, and deployment. These should then be reflected in the source selection criteria for the RFP.

- **Rec 7: Independent Verification and Validation for Machine Learning** - Machine learning is an increasingly important component of a broad range of defense systems, including autonomous systems, and will further complicate the challenges of software acquisition.