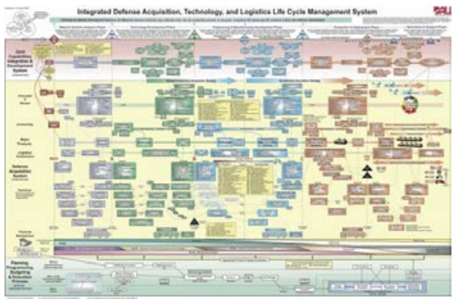
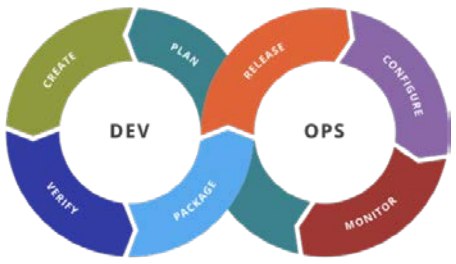
Defense Innovation Board Do's and Don'ts for Software

Version 0.6, last modified 2 Oct 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's Ten Commandments of Software.

Executive Summary

Observed practice (Don'ts)	Desired state (Do's)	Obstacles
 Defense Acquisition University, June 2010	 https://commons.wikimedia.org/wiki/File:Devops-toolchain.svg	In work
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	In work
Define success as 100% compliance with requirements	Accept 70% solutions[1] in a short time (months) and add functionality in rapid iterations (weeks)	In work
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	In work
Apply hardware life-cycle management processes to software	Take advantage of the fact that software is essentially free to duplicate, distribute, and modify	In work
Require customized software solutions to match DoD practices	For common functions, purchase existing software and change DoD processes to use existing apps	In work

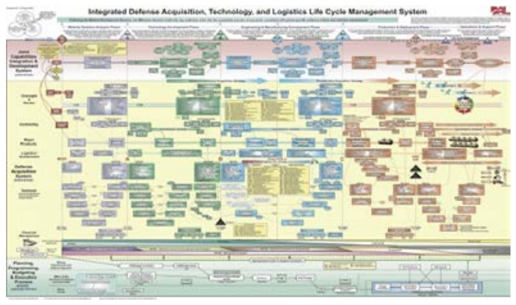
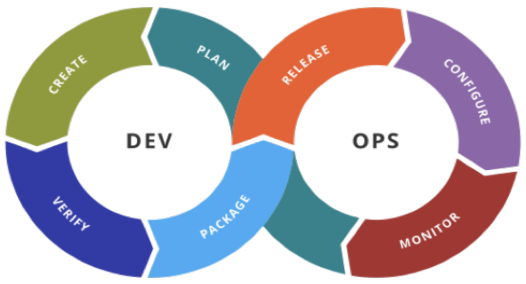
Integrated Defense Acquisition, Technology, and Logistics Life Cycle Management System

[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.

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)	In work
Evaluate cyber security after the systems have been completed, separately from OT&E	Use validated software development platforms that permit continuous integration & evaluation (DevSecOps)	In work
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	In work
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]	In work
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")	In work

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**
 Integrated Defense Acquisition, Technology, and Logistics Life Cycle Management System Defense Acquisition University, June 2010	 CREATE PLAN RELEASE CONFIGURE DEV OPS VERIFY PACKAGE MONITOR 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

[2] As noted in the DIB's 10 Commandments of Software
Acronyms defined: Application Programming Interface (API).

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 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 worth doing:

[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.

- 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 built 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.

- 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 worth doing:

- 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.

[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.*"

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.