Defense Innovation Board
Ten Commandments of Software

Executive Summary

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

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

Motivation and Scope

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

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

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

Software Types

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

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

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

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

The DIB's Ten Commandments of Software

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

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

Commandment #3. The acquisition process for software must support the full, iterative life cycle of software. Software does not age well. It must be constantly maintained and updated, ideally in an automated fashion. The PPBES process is nominally a two (2) year timeline to request and receive funding, with initial planning occurring five (5) years prior to actual receipt, and funding must be requested by intent of use (RDT&E, procurement, and O&M). But this fiscal separation does not match the process of software development, where all creation of code is

“development,” whether it falls within the fiscal law definition or not. As an alternative, the DoD should make use of “level of effort” (or capacity) constructs to allow continuous development and testing. Assume that low criticality software that is routinely used will require 10% of the development cost to maintain (per year) and more critical software will likely require more resources. This funding must be planned for at the time of initial development, not as an annual allocation that could be interrupted. Enhanced software capability should never be considered “ahead of need.” [All types]

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

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

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

Commandment #7. Every DoD system that includes software should have a local team of DoD software experts who are able to modify or extend the software through source code or API access. Modern weapons systems are software-driven and utilization of those systems in a rapidly changing environment will require that the system (software) be customizable by the

user. In order to do this, all fielded DoD systems that include software must also have a local team (responsible to the operational unit) that has the skills and permission to modify and extend the software through either source code (Commandment #6) or application programming interface (API) access. Local experts should have "reachback" capabilities to larger team and the ability to pull new features into their code (and generate pull requests for features that they add which should go back into the main codebase [repository]). [Types B, C, sometimes D]

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

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

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

Supporting Thoughts and Recommendations

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

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

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

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