**[Software is Never ‘Done’]**

[] This is a summary of the key points of the 292 page report.

[] DIB makes 10 primary recommendations and 16 additional recommendations to address the most critical statutory, regulatory, and cultural hurdles DoD faces in modernizing its approach to software.

[] In a year replete with reports on AI, Quantum Computing and Blockchain, it may seem mundane to write about software, but software is the foundation of all things digital, and the Dept does it exceedingly poor.

[] The above statements are made by Eric Schmidt in the opening statement of this report.

[] This report is to enable DoD as to how to develop, procure, assure, deploy and continiuously imporve software for its use in the department.

[] This study is mandated by the US Congress.

[] Hardware can be developed, procured, and maintained in a linear fashion but not software.

**[Chapter 0]**

[] In 2011, Marc Andreessen claimed in an op-ed for The Wall Street Journal that “Software Is Eating the World.”2 He argued that every industry (not just those considered to be “information technology”)

would be transformed by software—bytes rather than atoms. Eight years later, it is clear he was right.

[] Software is ubiquitous. It is part of everything the Department of Defense (DoD) does, from logistics to management to weapon systems. U.S. national security is critically dependent on the capabilities of DoD’s software.

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

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

[] China actively leverages its private industry to develop national security software (particularly in AI), recruits top students under the age of 18 to work on “intelligent weapons design,”3 and poaches U.S. software talent directly from the United States.

In Russia, Vladimir Putin has told students, that “artificial intelligence is the future, not only for Russia, but for all humankind.... Whoever becomes the leader in this sphere will become the ruler of the world.”4 We can and must outcompete with software and the people who make it, not only to maintain U.S. military superiority but also to ensure that the power that software represents is used in accordance with American values.

**[What this report is about]**

This report summarizes the assessment of the Defense Innovation Board’s (DIB’s) Software Acquisition and Practices (SWAP) study. Congress charged5 the DIB to recommend changes to statutes, regulations, processes, and culture to enable the better use of software in DoD. We took an iterative approach, mirroring the way modern software is successfully done, releasing a sequence of concept papers describing our preliminary observations and insights. (The latest versions of these are included in Appendix E.) We used those papers to encourage dialogue with a wide variety of individuals and groups to gain insights into the current

barriers to implementing modern software effectively and efficiently. This document captures key insights from these discussions in an easy-to-read format that highlights the elements that we consider critical for DoD’s success and serves as a starting point for continued discussions required to implement the changes that we recommend here.

**[Overarching themes]**

[] Software is a part of almost everything with which we interact in our daily

lives, either directly through embedded computation.

[] Software drives our military advantage: what makes weapon systems sophisticated is the software, not (just) the hardware.

[] Speed and cycle time are the most important metrics for software. Most DoD software projects are currently managed using “waterfall” development processes, which involve spending years on developing requirements, taking bids and selecting contractors, and then executing programs that must meet the listed requirements before they are “done.” This results in software that takes so long to reach the field that it is often not well matched to the current needs of the user or tactics of our adversaries, which have often changed significantly while the software was being written, tested, and accepted.

[] Faster reduces risk because it demands focus on the critical functionality rather than over-specification or bloated requirements. It also means we can identify trouble earlier and take faster corrective action, which reduces cost, time, and risk. Faster leads to increased reliability: the more quickly software/code is in the hands of users, the more quickly feedback can focus on efforts to deploy greater capability. Faster gives us a tactical advantage on the battlefield by allowing operation and response inside our adversaries’ observe–orient–decide–act (OODA) loops. Faster is more secure. Faster is possible.

[] As Steve Jobs observed,8 one of the major differences between hardware and software is that for hardware the “dynamic range” (ratio between the best in class and average performance) is, at most, 2:1. But, the difference between the best software developer and an average software developer can be 50:1, or even 100:1, and putting great developers on a team with other great developers amplifies this effect.

[] Testing and validation of software is also much different than for hardware, both in terms of the ability to automate but also in the potential vulnerabilities found in software that is not kept up to date. Software is never “done” and must be managed as an enduring capability that is treated differently than hardware.

**[Create and maintain cross-program/cross-Service digital infrastructure]**

Current practice in DoD programs is that each individual program builds its own infrastructure for computing, development, testing, and deployment, and there is little ability to build richer development and testing capabilities that are possible by making use of common infrastructure. Instead, we need to create, scale, and optimize an enterprise-level architecture and supporting infrastructure that enables creation and initial fielding of software within six months and continuous delivery of improvements on a three-month cycle.

[] The government must have experts well steeped in the software development process and architecture design to adequately manage both organic activities and contracted programs. They must have the skills to detect when contractors are going down the wrong path, choosing a bad implementation approach, or otherwise wasting government resources. This is perhaps the best argument for ensuring we have software development experience natively in the government, rather than relying primarily on external vendors; unless there are software-knowledgeable members on the core team, it is impossible to effectively monitor and manage outsourced projects. This is especially true with the movement to DevSecOps.

[] We use the term “DevSecOps” as our label for the type of culture that is needed: iterative development that deploys secure applications and software into operations in a continuing (and continuous) fashion.

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

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

[] Identify and launch programs to move out on the priority recommendations (start small, iterate quickly).

[] If you are reading this and are in a position of leadership in your organization, pass this on to others with your seal of approval and a request for your team to develop two or three plans of action for how it can be applied in your domain.

**[Chapter 1: Why does software matter for DoD?]**

[**Quote**] 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)

[] 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.

[] Software is so crucial that 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.

**[Different Types of Software for DoD and Military]**

**[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 email 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 DoDwide 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.)

**[sub-categories of combat systems]**

**[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 offthe-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 definition1 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).

**[types of computing platforms]**

**[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

**[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

[] 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.

**[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 System]**: 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.

[] 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.

[] DoD should also adopt commercial logistics and mission planning software (COTS) wherever possible and reduce its reliance on government off-the-shelf (GOTS) solutions.

[] 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.

[] 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.)

[] 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.

**[Chapter 2 – What does it look like to do Software right?]**

**[Quote]:** 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)

[] 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” adds the integration of security at all stages of development and deployment, which is essential for DoD applications.

[] DoD should use open source software when possible to speed development and deployment and leverage the work of others.

**[Best practices in Software Dev companies]**

[] Following are the best practices currently in use in many major s/w dev companies in the areas of s/w dev and project management.

**[Software Development]**

[] 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]**

[] 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]**

[] 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 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 reprioritize the software capability portfolio.

[] 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.

[] 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 single Leader]:** 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.

[] 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.

**[Technical Debt]**: 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.

**[On AI and ML]:** 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.

**[On Quantum Computing]:** 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 quantumresistant 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.

[] 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.

**[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.

[] 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.

**[Ten Commandments of Software]**

[] Make computing, storage, and bandwidth abundant to DoD developers and users.

[] All software procurement programs should start small, be iterative, and build on success ‒ or be terminated quickly.

[] The acquisition process for software must support the full, iterative life cycle of software.

[] Adopt a DevSecOps culture for software systems.

[] Automate testing of software to enable critical updates to be deployed in days to weeks, not months or years.

[] Every purpose-built DoD software system should include source code as a deliverable.

[] 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.

[] Only run operating systems that are receiving (and utilizing) regular security updates for newly discovered security vulnerabilities.

[] 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 generated by DoD systems - in development and deployment - should be stored, mined, and made available for machine learning.

**[Supporting Thought and Recommendations]**

[] Establish Computer Science as a DoD Core Competency.

[] Use commercial process and software to adopt and implement standard business practices within the Services.

[] Move to a model of continuous hardware refresh in which computers are treated as a consumable with a 2-3 year lifetime.

**[DIB Guide – Detecting Agile BS]**

**[Key flags that a project is not really agile]**

[] Nobody on the software development team is talking with and observing the users of the software in action; we mean the actual users of the actual code.

[] Continuous feedback from users to the development team (bug reports, users assessments) is not available. Talking once at the beginning of a program to verify requirements doesn’t count!

[] Meeting requirements is treated as more important than getting something useful into the field as quickly as possible.

[] Stakeholders (dev, test, ops, security, contracting, contractors, end-users, etc.) are acting more-or-less autonomously (e.g., ‘it’s not my job.’)

[] End users of the software are missing-in-action throughout development; at a minimum they should be present during Release Planning and User Acceptance Testing.

[] DevSecOps culture is lacking if manual processes are tolerated when such processes can and should be automated (e.g., automated testing, continuous integration, continuous delivery).

**[Some Common Tools currently in use by agile teams]**

[] Git, ClearCase, or Subversion - version control system for tracking changes to source code. Git is the de facto open source standard for modern software development.

[] BitBucket or GitHub - Repository hosting sites. Also provide issues tracking, continuous integration “apps” and other productivity tools. Widely used by the open source community.

[] Jenkins, Circle CI or Travis CI - continuous integration service used to build and test BitBucket and GitHub software projects

[] Chef, Ansible, or Puppet - software for writing system configuration “recipes” and streamlining the task of configuring and maintaining a collection of servers

[] Docker - computer program that performs operating-system-level virtualization, also known as “containerization”

[] Kubernetes or Docker Swarm for Container orchestration

[] Jira or Pivotal Tracker - issues reporting, tracking, and management

[] Automated build tools, like Maven, Grable, Cmake, and Apache Ant

[] Automated testing tools, like Selenium, Cucumber, J-Unit

[] A centralized artifacts repository, like Nexus, Artifactory, or Maven

[] Automated security tools for static and dynamic code analysis and container security, like Sonarqube, OWASP ZAP, Fortify, Nessus, Twistlock, Aqua, and more.

[] Automated code review tools, like Code Climate

[] Automated monitoring tools, like Nagios, Splunk, New Relic, and ELK

**[Questions to Ask Programming Teams]**

**[How do you test your code?]**

[] (Wrong answers: “we have a testing organization,” “OT&E is responsible for testing”)

[] Advanced version: what tool suite are you using for unit tests, regression testing, functional tests, security scans, and deployment certification?

**[How automated are your development, testing, security, and deployment pipelines?]**

[] Advanced version: what tool suite are you using for continuous integration (CI), continuous deployment (CD), regression testing, program documentation; is your infrastructure defined by code?

**[Who are your users and how are you interacting with them?]**

[] Advanced version: what mechanisms are you using to get direct feedback from your users? What tool suite are you using for issue reporting and tracking? How do you allocate issues to programming teams? How to you inform users that their issues are being addressed and/or have been resolved?

**[What is your (current and future) cycle time for releases to your users?]**

[] Advanced version: what software platforms to you support? Are you using containers? What configuration management tools do you use?

**[Questions for Program Management]**

[] How many programmers are part of the organizations that owns the budget and milestones for the program? (Wrong answers: “we don’t know,” “zero,” “it depends on how you define a programmer”)

[] What are your management metrics for development and operations; how are they used to inform priorities, detect problems; how often are they accessed and used by leadership?

[] What have you learned in your past three sprint cycles and what did you do about it? (Wrong answers: “what’s a sprint cycle?,” “we are waiting to get approval from management”)

[] Who are the users that you deliver value to each sprint cycle? Can we talk to them? (Wrong answers: “we don’t directly deploy our code to users”)

**[Questions for Customers and Users]**

[] How do you communicate with the developers? Did they observe your relevant teams working and ask questions that indicated a deep understanding of your needs? When is the last time they sat with you and talked about features you would like to see implemented?

[] How do you send in suggestions for new features or report issues or bugs in the code? What type of feedback do you get to your requests/reports? Are you ever asked to try prototypes of new software features and observed using them?

[] What is the time it takes for a requested feature to show up in the application?

**[Questions for Program Leadership]**

[] Are teams delivering working software to at least some subset of real users every iteration (including the first) and gathering feedback? (alt: every two weeks)

[] Is there a product charter that lays out the mission and strategic goals? Do all members of the team understand both, and are they able to see how their work contributes to both?

[] Is feedback from users turned into concrete work items for sprint teams on timelines shorter than one month?

[] Are teams empowered to change the requirements based on user feedback?

[] Are teams empowered to change their process based on what they learn?

[] Is the full ecosystem of your project agile? (Agile programming teams followed by linear, bureaucratic deployment is a failure.)

**[Technical practices and processes that enable a development environment to deliver value toward those goals include]**

[] Organization of activities through discrete “user stories” that can be broken down into smaller components and continually prioritized by the product owner

[] Relatively short “sprints” (often two weeks), each ending in a retrospective, that enable measurement and learning throughout the process

[] Blameless postmortems that allow for maximum learning and speedy recovery from failures

[] Automated testing, security, and deployment

[] Testing (including user testing) and security should be shifted to the left and be part of the day-to-day operations within the development teams

[] Continuous integration, in which developers integrate code into a shared repository several times a day, and check-ins are then verified by an automated build for early problem detection

[] Continuous delivery or continuous deployment, in which the software is seamlessly deployed into staging and production environments

[] Trunk-based development, in which team members work in small batches and develop off of trunk or master, rather than long-lived feature branches

[] Version control for all production artifacts including open source and third party libraries

[] Infrastructure as code: version control for all configuration, networking requirements, container orchestration files, continuous integration/continuous delivery (CI/CD) pipeline files

[] Ability to execute A/B testing and canary deployments

[] Ability to get rapid and continuous user feedback and to test new features with users throughout the development process

|  |  |  |  |
| --- | --- | --- | --- |
|  | **High Performance** | **Medium Performance** | **Low Performance** |
| **Deployment frequency**  How often does your organization  deploy code? | On demand  (multiple deploys  per day) | Between once  per week and  once per month | Between once  per week and  once per month |
| **Lead time for changes**  What is your lead time for changes  (i.e., how long does it take to go from  code-commit to code successfully  running in production)? | Less than one  hour | Between one  week and one  month | Between one  week and one  month\* |
| **Mean time to recover (MTTR)**  How long does it generally take to  restore service when a service  incident occurs (e.g., unplanned  outage, service impairment)? | Less than one  hour | Less than one  day | Between one  week and one  day |
| **Change failure rate**  What percentage of changes results  either in degraded service or  subsequently requires remediation  (e.g., leads to service impairment,  service outage, requires a hotfix,  rollback, fix forward, patch)? | 0-15% | 0-15% | 31-45% |

**[Warning signs that you may have screwed up your development environment]**

[] If teams cannot effectively track progress toward defined goals and objectives roughly every two weeks

[] If teams cannot rapidly deploy various environments that mirror production to test their code such as in development, QA, and staging

[] If teams cannot have real-time feedback regarding their code building, passing tests, and passing security scans

[] If it takes months for end users to be able to see changes and provide feedback

[] If teams cannot rapidly roll-back to previous versions or perform rolling-update to new versions without downtime

[] If recovering from incidents results in significant drama or the assignment of blame

[] If having code ready to deploy is a big event (it should happen routinely and without drama)

[] If changes to the software frequently result in breaking it

[] If developers are not empowered to change the code or build new functionality based on user feedback, or to change their process based on what they learn.

**[Compute and Infra capabilities that are required for DoD]**

**[Scalable Compute]**: Access to computing resources should never be a limiting factor when developing code. Modern cloud environments provide mechanisms to provide any developer with a powerful computing environment that can easily scale with the needs of an individual programmer, a product development team, or an entire enterprise.

**[Containerization]**: Container technology provides sandbox environments in which to test new software without exposing the larger system to the new code. It “packages up” an application with all of the operating system services required for executing the application and allowing that application to run in a virtualized environment. Containers allow isolation of components (that communicate with each other through well-defined channels) and provide a way to “freeze” a software configuration of an application without freezing the underlying physical hardware and operating system.

**[Ci/CD Pipeline]**: A platform that provides the CI/CD pipeline is used for automated testing, security, and deployment. This includes license access for security tools and a centralized artifacts repository with tools, databases, and a base operating system (OS) with an existing authorization to operate (ATO).

**[Infrastructure as code: automated config, updating, distribution and recovery management]**: Manual configuration management of operating systems and middleware platforms leads to inconsistencies in fielded systems and drives up the operating costs due to the labor hours required for systems administration. Modern software processes avoid this by implementing “infrastructure as code,” which replaces

manual processes for provisioning infrastructure with automated processes that use machine-readable definition files to manage and provision containers, virtual machines, networking, and other components. Adopting infrastructure as code and software distribution tools in a standardized way streamlines uniformity of deployment and testing of changes, which are both vital to realizing the benefits of agile development processes.

**[Federated identity management and authentication backend with common log file management and analysis]**: Common identity management across military, government, and contractors greatly simplifies the assignment of permissions for accessing information across multiple systems and allows rapid and accurate auditing of code. The ability to audit access to information across multiple systems enables the detection of inappropriate access to information, and can be used to develop the

patterns of life that are essential for proper threat analysis. Common identity management can ease the integration of multi-factor authentication across servers, desktops, and mobile devices. Along with public key infrastructure (PKI) integration, it allows verification of both the service being accessed by the user and the user accessing information from the service.

**[Firewall configuration and network access control lists]**: Having a common set of OS and application configurations allows network access control not just through network equipment, but at the server itself. Pruning unnecessary services and forcing information transfer only through intentional interfaces reduce the attack surface and make servers more resilient against penetration. Server-to-server communication can be encrypted to protect from network interception and authenticated so that software services can only communicate with authorized software elements.

**[Client software]**: Remote login through remote desktop access is common throughout DoD. This greatly increases the difficulty of integrating mobile platforms and of permitting embedded devices to access vital information, especially from the field. It also complicates uniform identity management and multi-factor verification, which is key to

securing information. By moving to web client access mobile integration – and development - is greatly eased. It also becomes possible to leverage industry innovation, as this is where the commercial sector is heading for all interactions.

**[Common information assurance (IA) profiles]**: Information assurance (IA) for DoD systems is complex, difficult, and not yet well-architected. Test, certification, and IA are almost always linear “tailgate” processes instead of being integrated into a continuous delivery cycle. Common IA profiles integrated into the development environment and

part of the development system architecture are less likely to have bugs than customized and add-on solutions.

**[Some indicators that you may have screwed up your compute environment]**

[] The headcount needed to support the system grows linearly with the number of servers or instances

[] You have older than current versions of operating systems or vendor software because it is too hard to test or validate changes

[] Unit costs for compute, network transport and storage are not declining, or are not measurable to be determined

[] Logging in via remote desktop is the normal way to access an information service

[] You depend on network firewalls to secure your compute resource from unauthorized access

[] You depend on hardware encryptors to keep your data safe from interception

[] You have to purge data on a regular basis to avoid running out of storage

[] Compute tasks are taking the same or longer time to run than they did when the system was first fielded

[] Equipment or software is in use that has been “end of lifed” by the vendor and no longer has mainstream support

[] It takes significant work to find out who accessed a given set of files or resources over a reasonable period of time

[] No one knows what part of the system is consuming the most resources or what code should be refactored for optimization

[] Multifactor authentication is not being used

[] You cannot execute a disaster recovery exercise where a current backup up of a system cannot be brought online on different hardware in less than a day

**[Programs Reviewed by the Committee for the purposes of this document]**

[] Next Generation fighter jet

[] Next Generation ground system

[] Kessel Run—AOC Pathfinder

[] Space tracking system

[] Naval radar system

[] Cross-service business system

**[Standard Questions, we asked]**

[] What is the coding environment and what languages/SW tools do you use?

[] What do the software and system architectures look like?

[] What is the computational environment (processing, comms, storage)?

[] How is software deployed and how often are updates delivered to the field?

[] What determines the cycle time for updates?

[] How does software development incorporate user feedback? What is the developer-user interface? How quickly are user issues addressed and fixed?

[] How long does it take to compile the code from scratch?

[] How much access does the DoD have to the source code?

[] How is testing done? What tool suites are used? How much is automated? How long does it take to do a full regression test?

[] How is cybersecurity testing done? How are programs/updates certified?

[] What does the workforce look like (headcounts, skill sets)? How many programmers? How much software expertise is there in the program office?

[] What is the structure of the contract with the government? How are changes, new features, and new ideas integrated into the development process?

**[Preliminary Observations]**

[] Software is being delivered to the field 2-10X slower than it could be due to outdated requirements, test requirements, and lack of trust in SW

[] Many systems are using legacy hardware and outdated architectures that make it much harder to exploit advances in computing and communications

[] Program requirements were often formulated 5+ years ago (when the threat environment + available technologies were very different => wasted effort)

[] New capabilities and features are added in multi-year (multi-decade?) development “blocks” instead of continuously and iteratively

[] Most program offices don’t have enough expertise in modern SW methods

[] Most SW teams are attempting to implement DevOps and “agile” approaches, but in most cases the capabilities are still nascent (and hence fragile)

[] Transition to DevOps is often hindered by a gov’t support structure focused on technical performance in a waterfall setting (“waterfall with sprints”)

[] Information assurance (IA) is complex, difficult, and not yet well architected

[] Test, certification and IA are almost always linear “tailgate” processes instead of being integrated into a continuous delivery cycle.

**[What should be done differently in future programs]**

[] Spend time upfront getting the architecture right: modular, automated, secure

[] Make use of platforms (hardware and software) that continuously evolve at the timescales of the commercial sector (3-5 years between HW/OS updates)

[] Start small, be iterative, and build on success ‒ or terminate quickly

[] Construct budget to support the full, iterative life cycle of the software

[] Adopt a DevOps culture: design, implement, test, deploy, evaluate, repeat

[] Automate testing of software to enable critical updates to be deployed in days to weeks, not months or years (also requires changes in testing organization)

[] Have a local team of DoD software experts who are capable of modifying or extending the software through source code or API access

[] Separate development of mission level software from development of IA-accredited platforms

**[Metrics for monitoring programs performance]**

[] Time from program launch to deployment of simplest useful functionality.

[] Time to field high priority functions (spec → ops) or fix newly found security holes

[] Time from code committed to code in use

[] Time required for regression tests (automated) and cybersecurity audit/penetration tests

[] Time required to restore service after outage

[] Automated test coverage of specs/code

[] Number of bugs caught in testing vs field use

[] Change failure rate (rollback deployed code)

[] Percentage of code available to DOD for inspection/rebuild

[] Complexity metrics

[] Development plan/environment metrics