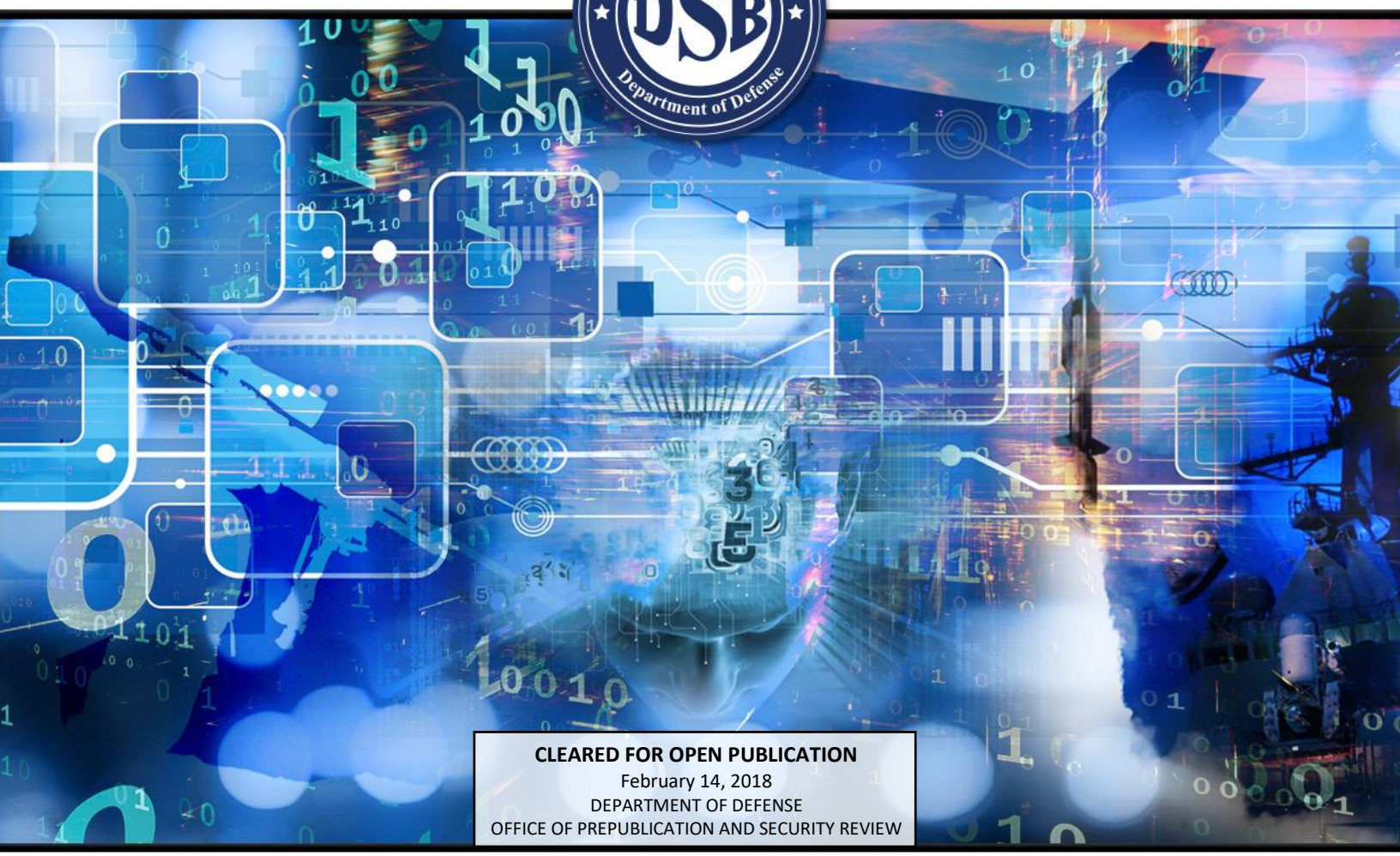
DEPARTMENT OF DEFENSE
DEFENSE SCIENCE BOARD

DESIGN AND ACQUISITION OF SOFTWARE FOR DEFENSE SYSTEMS

February 2018



OFFICE OF THE UNDER SECRETARY OF DEFENSE FOR RESEARCH AND ENGINEERING
WASHINGTON, D.C. 20301-3140

DEFENSE SCIENCE BOARD

OFFICE OF THE SECRETARY OF DEFENSE
3140 DEFENSE PENTAGON
WASHINGTON DC 20301-3140

MEMORANDUM FOR UNDER SECRETARY OF DEFENSE FOR RESEARCH AND ENGINEERING

SUBJECT: Final Report of the Defense Science Board (DSB) Task Force on the Design and Acquisition of Software for Defense Systems

I am pleased to forward the final report of the DSB Task Force on the Design and Acquisition of Software for Defense Systems, chaired by Dr. William LaPlante and Dr. Robert Wisnieff.

The Task Force has made seven recommendations on how to improve software acquisition in defense systems. A base recommendation underlying all others is to emphasize the importance of the software factory and to incorporate the software factory as a key evaluation criterion in the source selection process. Next, the Department of Defense (DoD) and its defense industrial base partners need to adopt continuous iterative development best practices. The study recommends DoD adopt best practices on risk reduction and metrics in formal program acquisition strategies. Software strategies must be better incorporated in current and legacy programs from development, production, and sustainment. The Task Force recommends ways to improve the software and acquisition workforce, in both software development expertise and the broader functional acquisition work force. Next, software is immortal and contracts must be framed to allow for software sustainment. Finally, the Task Force recommends further research into machine learning and the implementation of an independent verification and validation process for machine learning and autonomy in software systems.

Software is a crucial and growing part of weapons systems and the Department needs to be able to sustain immortal software indefinitely. The Task Force concluded that the Department of Defense would benefit from the implementation of continuous iterative development best practices as software becomes an increasingly important part of defense systems.

I concur with the Task Force's conclusions and recommend you forward the report to the Secretary of Defense.

Dr. Craig Fields
Chairman, DSB

THIS PAGE LEFT INTENTIONALLY BLANK

DEFENSE SCIENCE BOARD

OFFICE OF THE SECRETARY OF DEFENSE
3140 DEFENSE PENTAGON
WASHINGTON DC 20301-3140

MEMORANDUM TO THE CHAIRMAN, DEFENSE SCIENCE BOARD

SUBJECT: Final Report of the Defense Science Board Task Force on the Design and Acquisition of Software for Defense Systems

Attached is the final report of the Defense Science Board Task Force on the Design and Acquisition of Software for Defense Systems. The Task Force was formed to determine whether the iterative development practices in the commercial world are applicable to the development and sustainment of software for the Department of Defense (DoD). The study Terms of Reference stipulated the Task Force should:

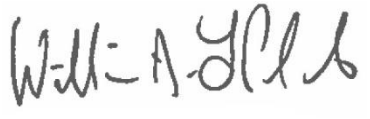
- examine the current state of DoD software acquisition and recommend actions for the DoD and its suppliers;
- consider development, test, and evaluation of learning systems;
- contrast and compare DoD and commercial software development and determine what commercial software development capabilities military systems should embrace;
- identify impediments in DoD requirements, contracting, and program management and how they might be removed;
- determine if "Agile" software techniques are being used effectively and identify impediments;
- determine if the commercial concept of a minimum viable product should be adopted by the DoD;
- determine best management approaches to achieve rapid and effective software upgrades, including an analysis of modular, open architecture;
- look at lessons learned from recent software challenges (e.g., OCX, F-35); and
- provide recommendations to ensure rapid adoption of cognitive capabilities as they mature.

The Task Force assessed best practices from commercial industry as well as successes within the DoD. Commercial embrace of iterative development has benefited bottom lines and cost, schedule, and testing performance, while the Department and its defense industrial base partners are hampered by bureaucratic practices and an existing government-imposed reward system. The Task Force concluded that the Department needs to change its internal practices to encourage and incentivize new practices in its contractor base. The assessment of the Task Force is that the Department can leverage best practices of iterative development even in its mission critical software systems.

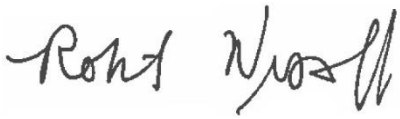
The Task Force made seven recommendations on how to improve software acquisition in defense systems. Our base recommendation, which underlies all other recommendations, is the importance

of the software factory – the efficacy of an offeror's software factory should be a key evaluation criterion in the source selection process for software. Next, the Department and its defense industrial base partners need to adopt continuous iterative development best practices (continuing through sustainment) for software. The Task Force recommends implementing certain best practices on risk reduction and metrics in formal program acquisition strategies as well as incorporating better software strategies in current and legacy programs in development, production, and sustainment. The Task Force further recommends workforce actions, both in software development expertise and in broader functional acquisition. The Task Force acknowledges that software is immortal, and therefore, the Task Force provides recommendations for software sustainment. Finally, the Task Force recommends further study of machine learning, including the implementation of an independent verification and validation process for machine learning and autonomy in software systems.

Software is a crucial and growing part of weapons systems and the national security mission, and the DoD must address its ability to build and sustain software continuously and indefinitely. Overall, the Task Force concludes that the Department can improve its methods of acquiring, building, and incentivizing software in defense systems and will greatly benefit from altering some of its acquisition practices and adopting continuous iterative development best practices.



Dr. William LaPlante
Co-Chair



Dr. Robert Wisnieff
Co-Chair

Table of Contents

List of Figures

List of Boxes

Executive Summary

The goal of the Defense Science Board (DSB) Task Force on Design and Acquisition of Software for Defense Systems was to determine whether iterative software development practices evolved in the commercial world are applicable to the development and sustainment of software for the Department of Defense (DoD).

Software has become one of the most important components of our Nation's weapons systems, and it continues to grow in importance. Software defines the way our systems see, communicate, and operate in combat. Design and acquisition decisions at the beginning of the software development process frequently have far-reaching and long-term effects that impact the weapon system's efficacy on the battlefield and its ability to adapt to changing requirements.

Software development in the commercial world has undergone significant change in the last 15 years, while development of software for defense systems has continued to use techniques developed in the 1970s through the 1990s. Traditional "Waterfall" software development practices (i.e., determining a functional specification, writing the software, and testing the software to the functional specification) have evolved in the commercial world into an iterative process, called "Agile" or "continuous iterative development," where a team develops software in smaller blocks that can be incrementally evaluated by a user community. This incremental approach allows updates and improvements to be rapidly incorporated into the software; in many cases, updates are made every day. The DoD, however, still largely buys and develops software developed using the slower traditional Waterfall approach that was mostly abandoned by commercial companies years ago.

Modern commercial software development best practices use software factories, which are a set of software tools that programmers use to write their code, confirm it meets style and other requirements, collaborate with other members of the programming team, and automatically build, test, and document their progress. This allows teams of programmers to do iterative development with frequent feedback from users. Additionally, a number of new tools and techniques are being utilized by the commercial sector, including:

- automation at scale;
- continuous development throughout the life of the product;
- increased and cheaper computing power;
- static, dynamic, and fuzz testing techniques, which have allowed substantial automated software testing; and
- open source, which leverages a larger community of developers to create reusable components and development tools.

These advances allow software production and sustainment to be done rapidly and continuously, enabling greater flexibility as requirements change. Harnessing these techniques and practices has yielded results in many commercial areas, from mobile and web technologies to banking, finance, and trade.

The DoD can leverage today's commercial development best practices to its advantage, including on its weapons systems. Doing so will enable the DoD to move from a capabilities-based acquisition model to a threat-based acquisition model. Making this transition is necessary if the United States is to maintain its technological superiority and counter rapidly growing adversary capabilities. Our adversaries are acquiring capabilities not previously anticipated and are doing so at a pace that now challenges U.S. technological superiority. The DoD needs to return to a modernized version of threat-based assessments. The United States must have the ability to quickly respond to adversary advancements and update our systems accordingly. Rapid and continuous software development will be essential to achieving this outcome.

The defense contractor base has not adopted many of the proven commercial sector software development techniques due to DoD culture, internal practices, and a government approach to contracting that disincentivizes their adoption. The DoD develops software and associated contracting based on upfront detailed systems requirements and specification for the entire completed system, an approach that is inadequate to meet today's challenges. The Department must change the structure of its contracts to incentivize best practices in its contractor base in order to take advantage of these modern software development practices.

Problems associated with software development continue to plague major DoD acquisition programs. This results in long delays in fielding, significant cost overruns, and, in some cases, program cancellation. The problems appear to be caused by the same software development issues that have occurred in programs over the last two decades. The Task Force strongly believes greater adoption of continuous iterative development and its associated best practices will result in significantly improved acquisition performance. The assessment of the Task Force is that an iterative approach to software development and sustainment is applicable to the DoD and should be adopted as quickly as possible.

1. Introduction

1.1 The Importance of Software in Defense Systems

Software is a crucial and growing part of weapons systems and the national security mission. While recognized as central to enterprise business systems and related information technology (IT) services, the role software plays in enabling and enhancing weapons systems often goes underappreciated.

Today, many of the capabilities provided by our weapons systems are derived from the software of the system, not the hardware. This shift from hardware-enabled capabilities to software-enabled capabilities is increasing quickly. As a 2017 paper published by the Institute for Defense Analyses notes, "The Department of Defense is experiencing an explosive increase in its demand for software-implemented features in weapon systems…in the meantime, defense software productivity and industrial base capacity have not been growing as quickly."[1]

In new weapons systems, software has become a significant part of the development and qualification process. Improving functionality and security can be delayed or even prevented by the inability to do the necessary testing; maintaining the complex testing infrastructure (i.e., both human and computing) is a growing issue.

Software does not only affect new weapons systems under development. Legacy systems, such as Tomahawk, F-16, and F-18, continue beyond design life due largely to improvements via software upgrades. While original development of these legacy systems used traditional software development practices, current upgrades have begun to employ iterative development practices, including for basic sustainment.

Unlike hardware, software never dies. Laying the groundwork to allow software improvement over the life of a weapons system is a strategic imperative. Utilizing development practices that enable continuous upgrade of capability ensures software can be adapted to threats and opportunities unanticipated during the specification of the system. The DoD must lay the groundwork now for software to meet the demands of the future.

DoD Software Growth

One method of estimating the complexity, cost and schedule, and overall centrality of software is to count the source lines of code (SLOC), often used as a basis of cost estimates.[2] This method has limitations – different languages and programming systems result in different SLOC counts, and industry no longer considers this technique credible. Even so, SLOC provides insight into a software system's size and the SLOC, for many weapons systems, has grown dramatically over the

[1] David M. Tate, "Software Productivity Trends and Issues (Conference Paper)," *Institute for Defense Analyses* (March 2017): iii.

[2] This procedural software cost estimation model is referred to as the Constructive Cost Model (COCOMO).

last four decades. **Figure 1** illustrates this trend for avionics software. This growth in SLOC shows how critical software is to the capabilities of advanced weapons systems.

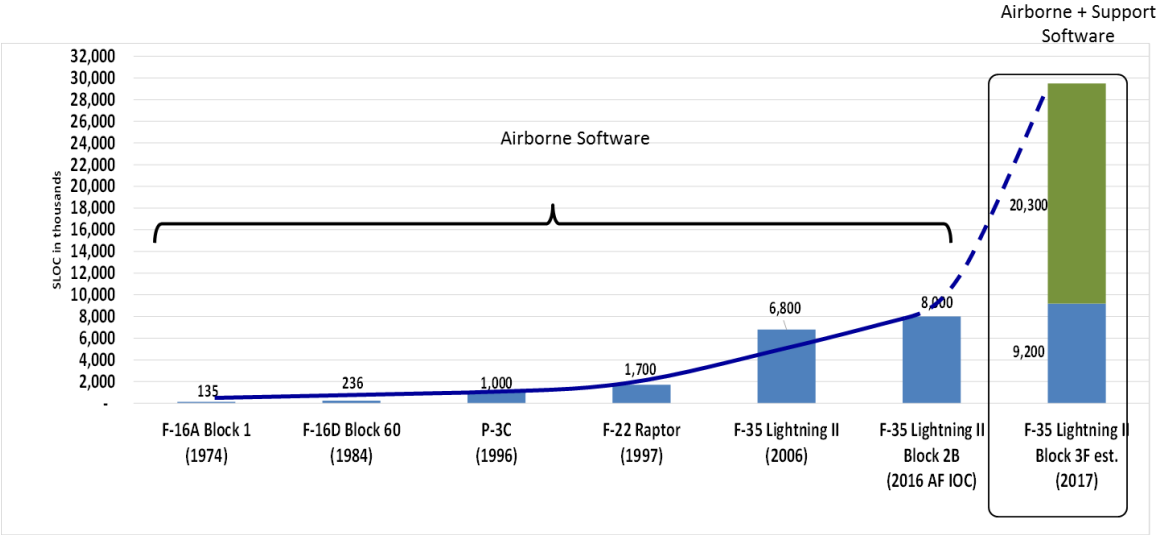


Figure 1. DoD Software Complexity and Growth: Explosive Growth of Source Lines of Code (SLOC) in Avionics Software[3]

DoD Software Risk Assessment

In the acquisition of new systems, software drives program risk for approximately 60 percent of programs (shown in **Figure 2**). Risks come in many forms. When building systems with new capabilities, it is not possible to anticipate all of the challenges until hands-on experience is obtained, not only in terms of basic operations but also for concepts of operation and tactics, techniques, and procedures. Unexpected complications can arise from unanticipated interdependencies within the software itself, often driven by the underlying architecture. A current DoD acquisition best practice is to reduce project risk by specifying the function of the software in detail at the beginning of a program. However, when such a system is tested, additional requirements typically are identified, thus requiring substantial effort to implement.

[3] The information in this chart was compiled from Christian Hagen, Jeff Sorenson, Steven Hurt, and Dan Wall, "Software: The Brains Behind U.S. Defense Systems," A.T. Kearney, 2012, https://www.atkearney.com/documents/10192/247932/SoftwareThe_Brains_Behind_US_Defense_Systems.pdf/69129873-eecc-4ddc-b798-c198a8ff1026. SLOC for F-16 and F-22 are at first operational flight. SLOC for F-35 Block 2B and 3F plus support software provided by the USD(R&E) office.

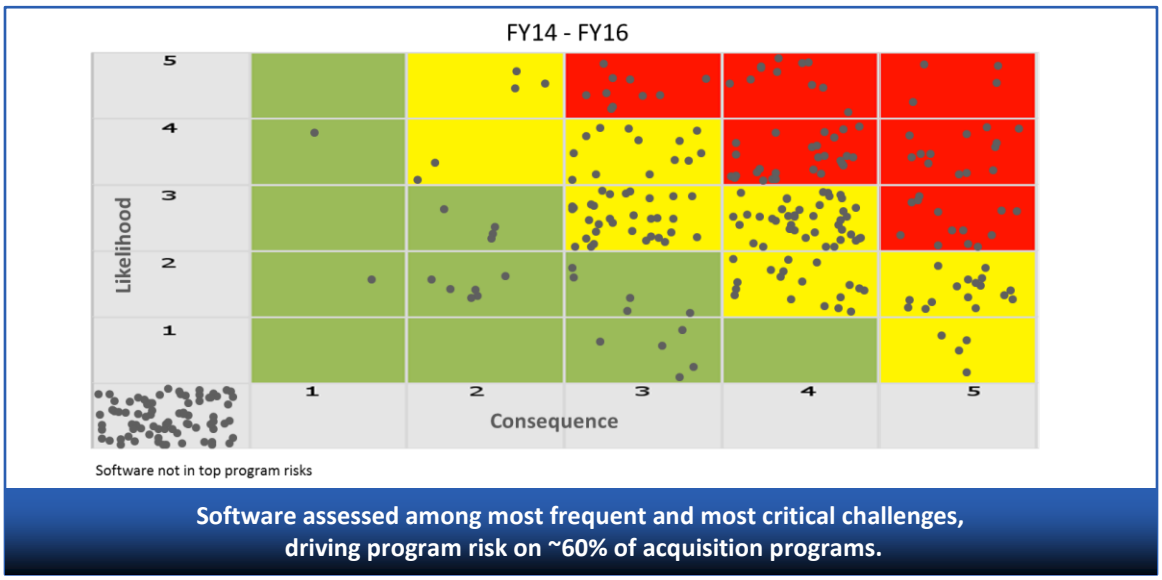


Figure 2. Software Risk Assessed by DoD Program Office

1.2 Silicon Valley Baedeker: Theories of Software Development

A number of software theories have evolved over time regarding software development. Assessing these different theories often leads to heated arguments about the best approach. This report uses the term "continuous iterative development" to characterize the best method for the DoD. Below is a Baedeker, or guide, to the various software approaches.

For more definitions of software terms, please see the glossary in *Appendix G*.

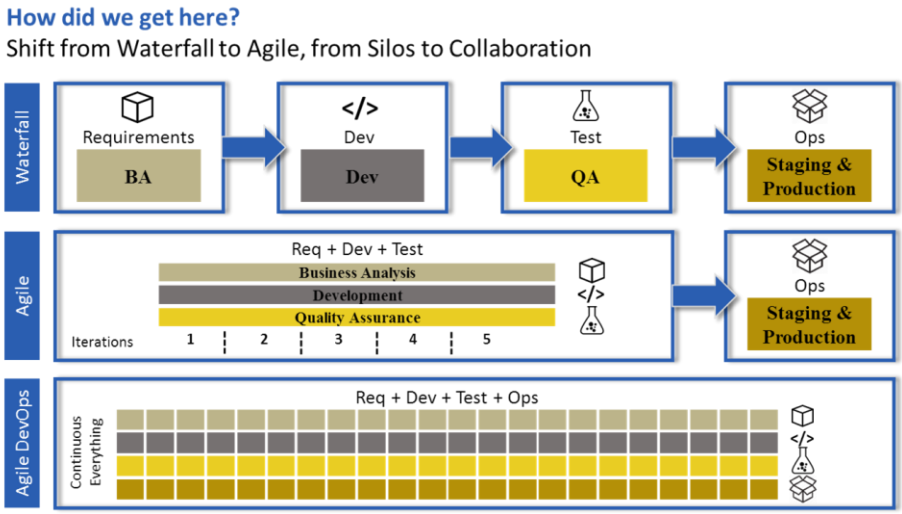


Figure 3. Theories of Software Development[4]

[4] Graphic adapted from Tim Dioquino, "DevOps: Transforming Military Application Delivery Lifecycles," *Hewlett Packard Enterprise, FedInsider, Intel,* 14.

Waterfall Development

The traditional approach to software development is Waterfall development. Waterfall development begins with writing down the full function specification. It is used to write the program as well as the tests. When the software passes all of the tests, it is considered finished and ready for delivery to the user.

Agile Development

Agile development, also called "iterative" development, begins with the creation of a software factory. Development and testing sprints – a set period of time during which specific work is completed – allow a team to do rapid iterations of development, obtain user feedback, and adjust goals for the next increment. This framework allows for continuous development throughout the life of the product.

Agile DevOps

DevOps entails running multiple Agile projects simultaneously to develop the next increment of an application. DevOps requires careful architectural design to avoid unintended complications by concurrent efforts. In general, this requires carefully defining the module and subsystem interfaces; thorough testing of interfaces is mandatory.

Iterative Development: Agile, Spins, and Spirals

Iterative development is the ineluctable process imposed by use of a product – especially a software product – that reveals a shortcoming or suggests an improvement. What distinguishes traditional iterative development from Agile approaches to software design and development is the velocity and granularity of the iterations. In venerable software production methodology (Waterfall development), an iteration commences after field deployment and use. New development approaches (i.e., Agile, spin, spiral) uncover and deal with flaws and opportunities much earlier in the process, leading to rapid development of a more robust product delivered to the field.

2. Finding: Continuous Iterative Development for the Department of Defense

2.1 DoD Software Processes

The standard software development process in the DoD follows the linear path illustrated in **Figure 4**: requirements are finalized and documented, schedule and cost is set at the beginning of the program (often using legacy SLOC-based models), and a preliminary design review is performed leading up to the release of the development request for proposal (RFP). After Milestone B and contract award, software is developed using resources determined by estimating the SLOC of each section of software. Finally, the system is tested prior to release. This approach, referred to as "Waterfall development," dominated all of commercial and defense software development until the early 2000s.

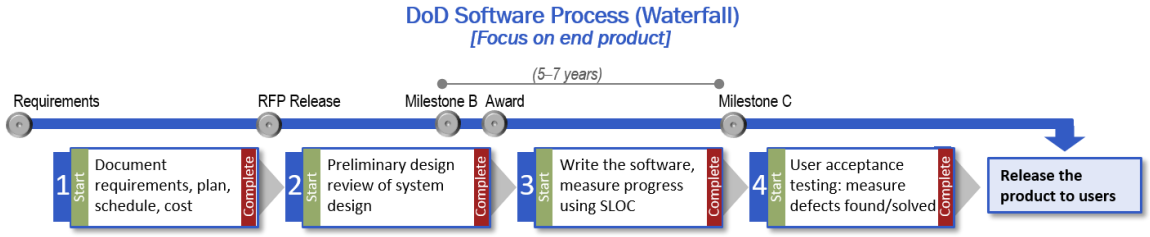


Figure 4. DoD Software Process (Waterfall)

2.2 Commercial Software Processes

The growth of mobile computing in the 2000s forced commercial organizations to look for ways to write software without knowing all of the requirements ahead of time while anticipating future security and testing concerns. To wait for certainty about requirements meant companies losing their markets. The goal was to find ways to iteratively develop software, extending capability incrementally over time.

Figure 5 illustrates the cyclical process of continuous iterative development commonly employed in the commercial sector. Goals and features are identified at the beginning, but requirements are not strictly set as in the usual DoD process. User feedback is used to establish goals of each iteration (called a "sprint") and to establish the definition and expectations of the minimum viable product (MVP). The software team writes the software using a highly automated tool chain that rebuilds the system and tests the resulting changes every night. If issues are found, the developers make the necessary changes the next day. The continuous development process, which lasts weeks, delivers an MVP to the user at the end of each iteration. Within the loop, there are nightly builds and tests, including durable, automated granular, performance, security, and capability tests that facilitate confidence when changes are subsequently introduced. Problems can be identified daily. The goal of this process is delivering a series of products that provide enhanced functionality, facilitating ongoing safe modification, and enabling users to evaluate performance that drives the next iteration.

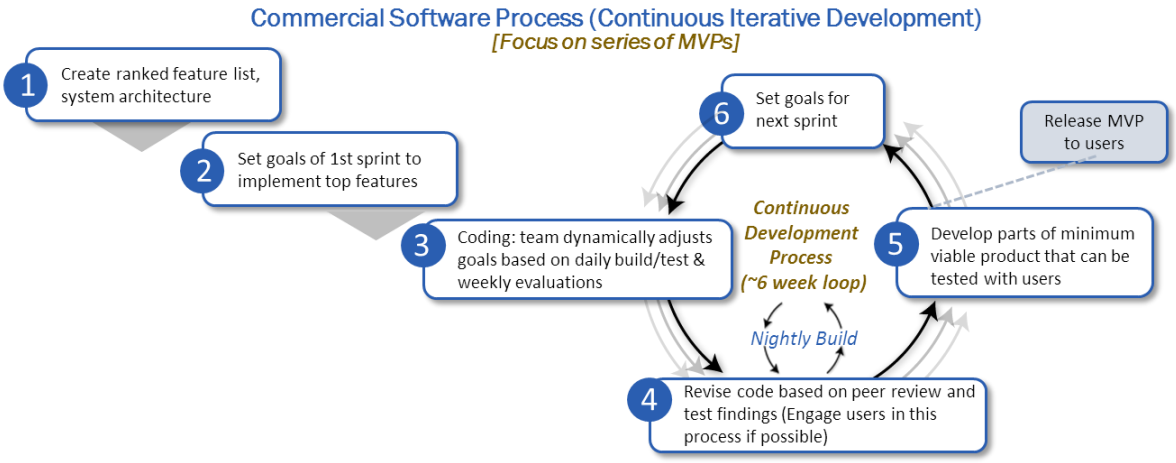


Figure 5. Commercial Software Process (Continuous Iterative Development)

The decrease in the cost of computing made this iterative development approach cost-effective. Previously, it was too expensive to run a computing infrastructure that could build and test the entire project every night. Large projects would compile the entire software system every six to nine months, making it more difficult for a programmer to see dependencies or other problems with the system. This iterative and more automated approach initially was embraced in the mobile space; its success led to widespread adoption across most areas of the commercial world.

Going from one MVP to another enables spiral development. The lessons learned during an iteration cycle are used to set key features and changes for the next iteration. Software architecture is key to enabling this approach and must be designed to allow and account for changes. Therefore, function must be assigned to modules to enable likely extensions and evolution. Successful developments become visible in the product while unsuccessful ones are discarded. Companies often ameliorate the risk of unsuccessful architectures by starting multiple groups with different architectures and down-selecting when the best architecture is determined, which is not an easy task.

2.3 Software Factory

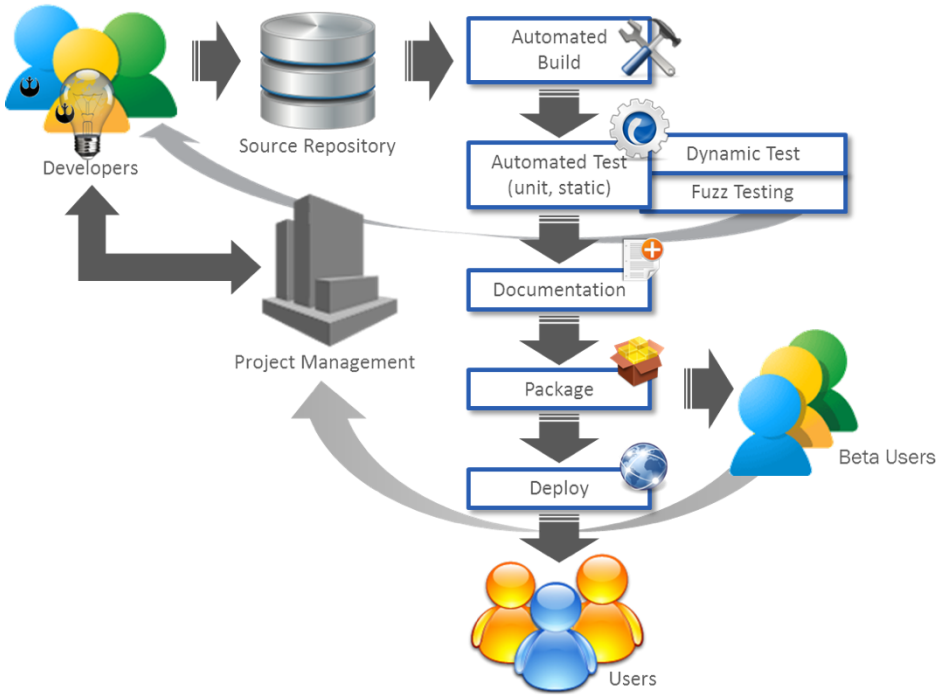


Figure 6. Software Factory

Underlying iterative development is the software factory, illustrated in **Figure 6**. Low-cost, cloud-based computing is used to assemble a set of tools (see *Appendix E* for an example list of applications) that enable the developers, users, and management to work together on a daily tempo. The goal is to ensure the code meets requirements by building and testing the application automatically every day and feeding back any issues to the developer responsible for the code. A source code repository archives current and past versions of the application while each developer works on a local copy of the code. After attaining a stable version, it is uploaded to the repository along with extensive tests and test data, and documentation listing the added features and resolved issues. In most organizations, code is peer reviewed prior to the upload. Peer review is especially useful for new members of the team, allowing them to learn the nuances of the software system conventions.

Once the code is uploaded, a style checker ensures there are no violations of coding conventions and then the software system is built. For interpreted languages such as Python or Swift, the build process involves static testing (i.e., no undeclared variable, no variables being called after the variable has been discarded) and syntax checking. For compiled languages, such as C, a compilation of the source code to executable code is involved. Individual modules then go through unit testing, which validates resolution of previously identified issues as well as compatibility with required functionality. In a new project, the first software written is often the unit tests and, in fact, comprehensive unit tests can offer the best insight into function. The full build is dynamically tested by executing use-scenarios identified as edge cases. Fuzz testing is also used — giving random inputs of all allowed values — to look for instances where unexpected

behavior is displayed. Any issues identified during the build and test process are communicated to the programmer and errors receive attention quickly.

The build process also generates documentation. Typical tool chains allow the programmer to incorporate documentation directly into source code. The documentation can then be extracted and the documents assembled. Next, the full system is packaged into a container allowing rapid, reliable deployment to users. Users and automated monitoring provide feedback to the development team through the project management software, providing a channel to communicate desired feature additions and modifications as well as prioritized bug reports.

2.4 Addressing Cyber

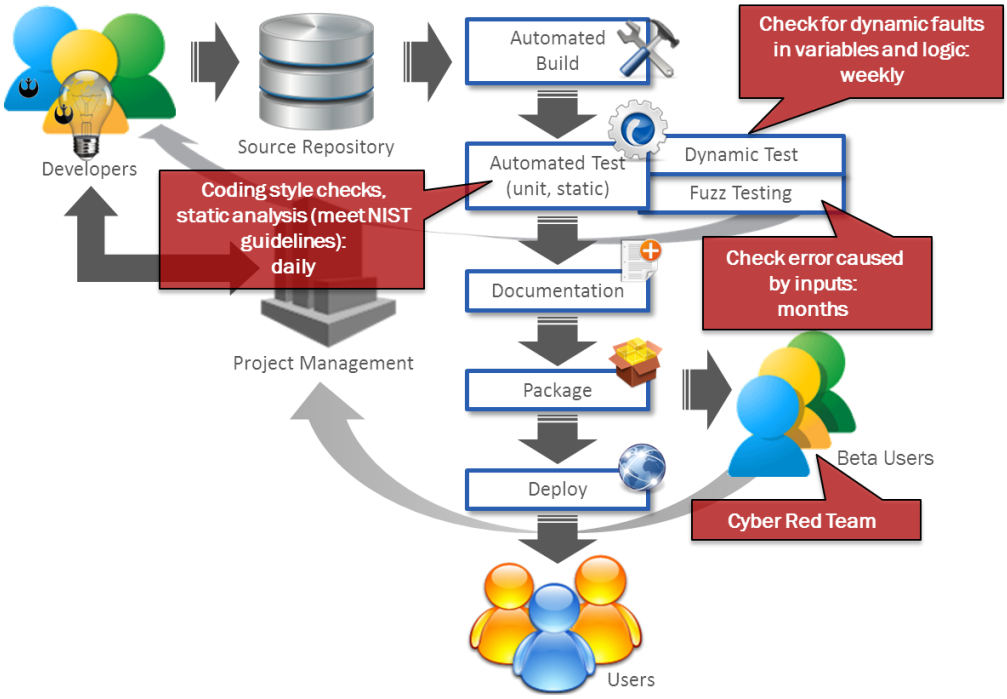


Figure 7. Addressing Cyber in the Software Factory

A tool chain for iterative development enables code to be developed that meets a set of cyber rules (shown in **Figure 7**), which prohibit constructions likely to become vulnerabilities. The cyber rules are formulated as style sheet checks. Code that violates the rules (e.g., not checking for overflow) is highlighted to the programmer. The National Institute of Standards and Technology (NIST) guidelines may be used as a starting point for this list of cyber rules. Checking a software system's code base daily keeps manageable the number of changes required to comply with a large base of cyber rules. When a new vulnerability is discovered, additional rules are formulated to detect similar errors in code. Dynamic testing helps identify logic errors and fuzz testing checks for vulnerabilities to user input-induced faults. Red teams are also periodically employed to evaluate the packaged code for faults. Issues identified by red teams are fed back along with automated tests preventing future similar issues.

2.5 Importance of Architecture

Given today's dynamic security environment, it is impossible to formulate a complete set of software requirements ahead of time. Without a robust underlying architecture, someone working on a low-level function will be unable to understand all the end applications in which a function might be used. Therefore, the architect must try to define modules in a way that avoids cross-couplings, whereby changes in one module impact and require changes to other modules. A goal of iterative development is to have many programmers concurrently working on different aspects of a shared code base; however, this parallel effort is possible only with a suitable architecture. Avoiding conflict between the different concurrent efforts requires an understanding of which aspects of the program will require de-conflicting and ensuring that only one programmer has responsibility for implementation.

While full specifications should be eschewed, emphasis must be placed early on in a project to develop clear, complete, and easily communicated principles of operation. Initial builds with alternate architectures may help to gain sufficient understanding to make an informed choice of final architecture. The Task Force found commercial practice starts with several competing architectures and winnows down to the one that experience suggests can handle iterative development requirements. While this practice at first seems inefficient, the long-term gain of an architecture that permits iterative development justifies the investment. In addition, well-architected, well-documented components accompanied by automated tests can often be reused.

2.6 The Right Conditions for Iterative Development in Defense Systems

After assessing the different software development methods and examining the benefits derived from employing iterative development practices in the commercial sector, the Task Force believes there are many circumstances where adoption of continuous iterative development would greatly benefit the DoD and its defense contractors.

The main benefit of iterative development — the ability to catch errors quickly and continuously, integrate new code with ease, and obtain user feedback throughout the development of the application — will help the DoD to operate in today's dynamic security environment, where threats are changing faster than Waterfall development can handle.

Systems such as platform mission software, electronic warfare (EW), communications, radar, and launch systems could benefit from continuous improvements and extensions to system capabilities, crucial for creating tactical advantage and coping with strategic surprise. The Task Force found the EW domain especially could benefit from modern software practices.[5] In EW, rapid software changes allow both new modes to be deployed and new adversary capabilities to be detected on an operationally useful timeframe.

[5] For more information on electronic warfare, see the Defense Science Board's report, "21st Century Military Operations in a Complex Electromagnetic Environment," *U.S. Department of Defense,* July 2015, http://www.dtic.mil/dtic/tr/fulltext/u2/1001629.pdf.

However, not all projects are well-suited to iterative development approaches. Examples of applications unlikely to benefit from iterative development include digital engine control systems, low-level mission critical flight control systems, and legacy systems at end-of-life. The first two examples are systems that control a platform (e.g., jet engine or airframe) that seldom change over the life of the application; these applications are specified during development to keep the platform within acceptable operational limits that will not change. Still, even these systems benefit by the automated modelling and testing modern software factories encourage. End-of-life systems are no longer undergoing application development.

Ground control systems and enterprise logistics support systems do not require changes as frequently as EW systems, but they do require changes more frequently than the low-level mission critical flight control systems. Ground control systems and enterprise logistics support systems need to be able to innovate quickly to respond to the loss of some of their system elements. This is where the architecture again comes into play — a good architecture can enable iterative approaches in these systems while a poor architecture can impede iterative approaches. **Figure 8** provides an overview of favorable and unfavorable conditions for iterative development. **Figures 9** and **10** illustrate how some capabilities are well-suited for iterative development while other capabilities are not, even on the same weapons platform.

The Right Conditions for Agile

Conditions	Favorable	Unfavorable
Market environment	▪ Customer preferences and solution options change frequently.	▪ Market conditions are stable and predictable.
Customer Involvement	▪ Close collaboration and rapid feedback are feasible. ▪ Customers know better what they want as the process progresses.	▪ Requirements are clear at the outset and will remain stable. ▪ Customers are unavailable for constant collaboration.
Innovation Type	▪ Problems are complex, solutions are unknown, and the scope isn't clearly defined. ▪ Product specifications may change. ▪ Creative breakthroughs and time to market are important. ▪ Cross-functional collaboration is vital.	▪ Similar work has been done before, and innovators believe the solutions are clear. ▪ Detailed specifications and work plans can be forecast with confidence and should be adhered to. ▪ Problems can be solved sequentially in functional silos.
Modularity of Work	▪ Incremental developments have value, and customers can use them. ▪ Work can be broken into parts and conducted in rapid, iterative cycles. ▪ Late changes are manageable.	▪ Customers cannot start testing parts of the product until everything is complete. ▪ Late changes are expensive or impossible.
Impact of Interim Mistakes	▪ They provide valuable learning.	▪ They may be catastrophic.

Figure 8. Harvard Business Review: Embracing Agile[6]

[6] Darrell K. Rigby, Jeff Sutherland, and Hirotaka Takeuchi, "Embracing Agile," *Harvard Business Review* (May 2016), https://hbr.org/2016/05/embracing-agile.

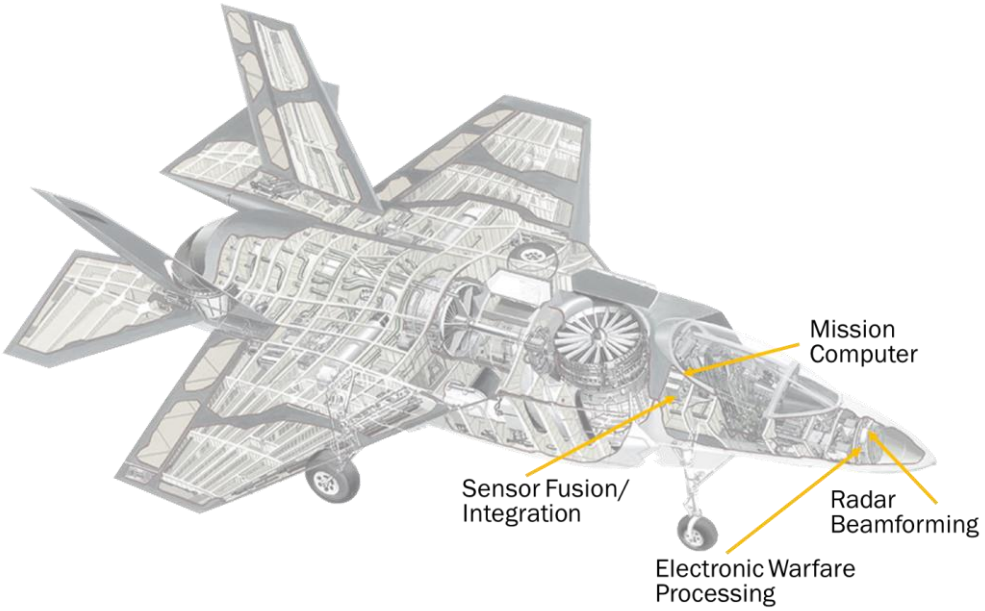


Figure 9. Favorable Conditions for Iterative Development on the F-35

[Highlighted functions will change often with new sensors and algorithm development. Changes are possible even mission to mission, and must be rapidly upgradable to protect the viability of the platform.]

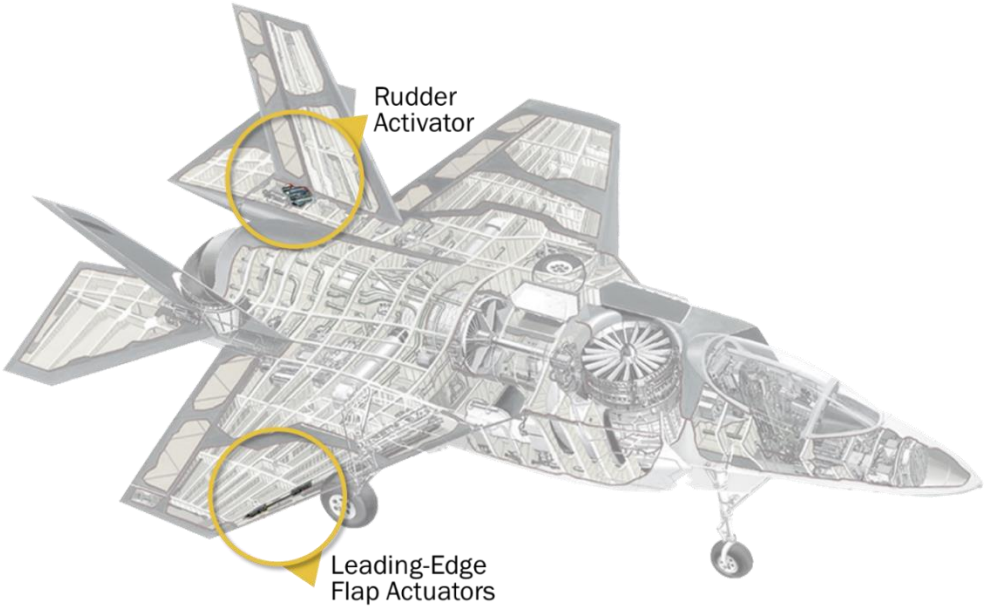


Figure 10. Unfavorable Conditions for Iterative Development on the F-35

[Highlighted functions impact flight safety, require rigorous acceptance testing and are not expected to regularly change throughout the platform life.]

In addition to iterative development producing the best results for projects ideally suited to benefit from its process, many of its techniques and best practices can produce benefits even when applied to a Waterfall development project. For example, the requirements for Boeing's fixed-price KC-46A tanker were set for over seven years with little chance of changing, but by using a software factory for development, programmatic error risks were reduced. Therefore, even under conditions that would suggest a Waterfall approach, programmers can still improve their processes and end products by adopting some iterative techniques.

2.7 The Case For and Against Iterative Development for DoD Systems

There is considerable anecdotal evidence to support the Task Force's belief that iterative development techniques are attractive for the DoD and its contractors. Companies that have embraced Agile approaches speak highly of the payoffs, and this family of approaches have become the standard in commercial software development, from Microsoft to Amazon to Google to Facebook. IBM is now making the transition to this approach as well. Moreover, there are no reports of companies transitioning from Agile to Waterfall software development approaches.

However, there are no widely cited or authoritative empirical studies to support the thesis that Agile development practices are superior to Waterfall approaches. Even if there were such studies, they would likely be focused on commercial software and, thus, one might question whether those results would translate to the kinds of software systems that the DoD builds, which are often characterized by a real-time control requirement and a high-end security threat.

Ideally, empirical studies would account for various kinds of Agile development methods, system architectures, programming languages, and systems (i.e., enterprise data processing vs. real-time control vs. signal processing). Yet, to build a substantially sized system even twice — once using Waterfall and once using Agile — would be costly and time consuming. Therefore, generalizations must be made from empirical studies of relatively small systems.

There are also many dimensions for comparison. A useful study might hope to understand the impact of iterative development on direct measures such as quality (i.e., measured in terms of reported bugs or exploitable vulnerabilities), system size (i.e., measured as SLOC), and development effort (i.e., elapsed time or total labor hours). Results also need to be calibrated based on the expertise and experience of the development team with the nature of the problem, the programming language, and the tool chains, among others.

The Task Force expects that, over time, considerable literature will be produced documenting experiences with iterative development methods. Using that literature, a better understanding of the benefits will emerge. For now, the Task Force found only two studies that surveyed the use of iterative development providing empirical comparisons. One contained a survey of 36 empirical studies prior to 2005.[7] Of the studies surveyed there, four gave empirical data for productivity

[7] Tore Dyba and Torgeir Dingsoyr, "Empirical studies of agile software development: A systemic review," *Information and Software Technology* 50, no. 9-10 (2008), https://doi.org/10.1016/j.infsof.2008.01.006.

comparison of the "extreme programming" (XP) version of iterative versus traditional development. **Figure 11** summarizes these results using lines of code (LOC) as the measurement.

Study	Traditional Productivity	Agile Productivity	Productivity Gain
S7	3 LOC/hr	13.1 LOC/hr	337%
S10	3.8 LOC/hr	5.4 LOC/hr	42%
S14	300 LOC/month	440 LOC/month	46%
S32	157 LOC/engr	88 LOC/engr	-44%

Figure 11. Dyba and Dingsoyr Meta-survey[8]

Some of the other relevant findings include the following:

- S10[9] finds that 13% fewer defects were reported by the customer as compared with a non-Agile project.
- S14[10] found a 65% improvement in pre-release quality and a 35% improvement in post-release quality for an Agile-developed project.
- S15[11] compared XP with Waterfall and found no difference in observed quality when comparing the work of 10 XP teams and 10 traditional teams.
- S28[12] compared Agile and document-driven approaches in managing uncertainty in software development, finding companies that use Agile methods are more customer-centric and flexible than document-driven ones, and companies that use Agile methods seem to have a more satisfactory relationship with the customer.

[8] S7 involved 15 teams and used four different approaches. This study showed the greatest difference between traditional and Agile. The Agile team delivered far more code, but achieved the same functionality as the traditional team. Regarding S14, the Agile team had more experience with languages and management. S32 was a study concerning student programmers.

[9] Sylvia Ilieva, Penko Ivanov, and Eliza Stefanova, "Analyses of an agile methodology implementation," *Proceedings of the 30th EUROMICRO Conference,* 2004, http://ieeexplore.ieee.org/document/1333387/.

[10] Lucas Layman, Laurie Williams, and Lynn Cunningham, "Exploring Extreme Programming in Context: An Industrial Case Study," *Proceedings of the Agile Development Conference,* 2004, http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1359793.

[11] Francisco Macias, Mike Holcombe, and Marian Gheorghe, "A Formal Experiment Comparing Extreme Programming with Traditional Software Construction," *Proceedings of the Fourth Mexican International Conference on Computer Science,* 2003, http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1232877.

[12] Alberto Sillitti, Martina Ceschi, Barbara Russo, and Giancarlo Succi, "Managing Uncertainty in Requirements: a Survey in Documentation-driven and Agile Companies," *11th IEEE International Software Metrics Symposium,* 2005, http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=1509295.

- S18[13] asked subjects whether the team's productivity had increased significantly as a result of the development process that was used. On a scale from one (strongly disagree) to six (strongly agree), the mean for the non-XP developers was 3.78, while the mean for the XP developers was 4.75.

- S32[14] compared plan-driven and Agile development to team cohesion and product quality. The XP team's code scored consistently better on the quality metrics used than the traditional team. In addition, the quality of the code delivered by the XP team was found to be significantly greater than that delivered by the traditional team. However, both teams agreed the traditional team had developed a better and much more consistent user interface.

A second meta-survey[15] of 29 studies (of 300 articles analyzed) found that Agile development yielded the following return on investment (ROI):

- 29% lower cost
- 91% better schedule
- 50% better quality
- 400% better job satisfaction

The Task Force concluded there are too many variables to generalize from this past work to quantify the benefits to the DoD by switching to iterative methods. Much more experience is needed before authoritative empirical results about the benefits of iterative development can provide insight to the DoD and its contractors. However, the principles behind Agile development can be evaluated on their own merits by people who understand the software development enterprise. The Task Force found these principles to be sound. Moreover, they address problems the DoD has been experiencing with Waterfall approaches. Finally, even without careful experimental results, the widespread adoption and endorsement of iterative techniques by the commercial sector supports the view that DoD contractors will benefit from making the transition.

---

[13] Katiuscia Mannaro, Marco Melis, and Michele Marchesi, "Empirical Analysis on the Satisfaction of IT Employees Comparing XP Practices with Other Software Development Methodologies," in *Extreme Programming and Agile Processes in Software Engineering,* eds. Jutta Eckstein and Hubert Baumeister (New York: Springer-Verlag, 2004), 166-174, https://link.springer.com/chapter/10.1007/978-3-540-24853-8_19.

[14] Carol A. Wellington, Thomas Briggs, and C. Dudley Girard, "Comparison of Student Experiences with Plan-Driven and Agile Methodologies," *35th ASEE/IEEE Frontiers in Education Conference,* 2005, http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1611951.

[15] David F. Rico, "What is the ROI of Agile vs. Traditional Methods," http://w.davidfrico.com/rico08b.pdf.

3. Finding: Commercial, the DoD, and Its Partners: Case Studies

3.1 Differences and Similarities of DoD and Commercial Software Development

Commercial development practice has rapidly evolved over the last 15 years. The need to adapt to changes while maintaining a consistent user experience forced companies to use large teams of development engineers located at different sites but working together. Keeping the teams synchronized meant everyone had to work from a common code base, everyone used the same programming style and tools, and everyone received rapid feedback about the impact of changes. Cloud-based development environments made daily compilation, linking, and testing of the entire codebase possible. Weekly updates evolved into nearly continuous events. The different parts of the tool chain also evolved, many of them open source. This is in contrast with much of the software development that is done for the DoD. The Task Force met with several commercial industry representatives, including from Facebook and Google. **Box 1** explains some commercial best practices for software development.

1 Box 1: Facebook and Google Best Practices

FACEBOOK

Facebook was one of the first companies to embrace iterative development to meet demand and deliver a superior user experience in the mobile marketplace. Facebook delivers a continuous stream of improvements on hardware (e.g., cell phones, tablets) over which they have no control, and deals with attempts to curtail its access to certain geographies on a daily basis. Currently, Facebook operates with the DevOps mode with many teams developing improvements simultaneously and releasing these improvements continually. All of the aspects of a software factory are used along with peer review, enabling new members to be brought up-to-speed on coding practice. Dynamic and fuzz testing are used to catch errors.

Programmers are required to be online during the launch of their code. Facebook programmers have developed a strategy of incremental release that limits the downside potential of an unintended consequence.

GOOGLE

Google is also at the leading edge of high-speed iterative development, mostly using small teams of people working on projects that feed into a platform. They perform more than 150 million test cases per day. Like Facebook, they have developed a release strategy (called “canaries”) that limits downside risk.

3.2 Defense Prime Contractors State of Play

Most software developed by the major defense prime contractors follows the traditional Waterfall process, albeit with some exceptions where large programs are broken into blocks, which may themselves last a year or more. These contractors are familiar to some degree with iterative development—some more so than others. Most have used it on selected small programs or portions of larger DoD programs. Some seem eager to pursue iterative development as their primary methodology. DoD contractors understand they are 15 years behind best commercial practices and would benefit from closer and more frequent customer feedback that iterative development enables. Nevertheless, the defense prime contractors' perception is that they are unable to adopt iterative development methods because DoD contracts require documentation, progress reviews, and incentives based on a Waterfall model.

Others see iterative development as something useful for web applications, but not appropriate for most defense systems due to their complexity and importance. These DoD contractors do not seem inclined to change their current approach to developing software given the current incentive structure.

Still, others are already trying to adopt portions of the iterative process into their development processes when it does not conflict with contractual language. They cite cost and schedule benefits in doing so. There are cases of iterative processes used for large-scale, fixed-price development programs whose requirements have been unchanged for many years (e.g., KC-46A Tanker). Three examples of iterative development implementation among defense prime contractors follow in **Boxes 2, 3,** and **4**

2 Box 2: Iterative Development with Fixed Price: KC-46A Tanker

The KC-46A Pegasus is the aerial refueling and strategic military transport aircraft developed by Boeing based upon its design of the 767 jet airliner. The tanker has been under development since 2011 and will replace the older KC-135 Stratotankers. The first 18 KC-46A are scheduled to be delivered to the U.S. Air Force in 2018. The development phase of the KC-46A is being conducted under a fixed-price incentive-fee contract, where the U.S. Government's liability is capped at the fixed-price ceiling and any cost overaged is the responsibility of the industry partner. As with other fixed-price development programs, there is an emphasis by the U.S. Government to keep requirements firm to meet the terms of the fixed-price contract. Accordingly, there have been no requirements changes to the program since 2011. However, within this overall fixed requirement structure, Boeing has segmented the software development into small pieces and is using modern iterative software development practices. This is in essence a hybrid approach between Waterfall and modern iterative software development.

3 Box 3: Iterative Development for the National Security Mission: SpaceX

SpaceX appears to be an "existence proof" that modern DevOps commercial practices can be used effectively for rapidly changing systems that are mission critical for national security, in this case the Air Force Space Launch.

SpaceX uses Agile scrum – a framework for managing Agile software development – for the project management of critical internal systems (Enterprise Resource Planning, Space Operations, Finance, and Human Resources). A continuous deployment pipeline for updating critical internal information multiple times per day is utilized. Furthermore, the internal application infrastructure is managed entirely in code for both Windows and Linux hosts and utilizes automated testing of software artifacts.

As an acquisition model, the U.S. Government competes the launch as a service. When SpaceX wins an award, they have the freedom to develop hardware and software organically as they see fit; however, it must remain launch certified. SpaceX has been using iterative software since 2010. Requirements changes come from both the customer (e.g., specifics to each launch mission) and from SpaceX internally (e.g., improvements to capability).

4 Box 4: National Security Agency Has Successfully Moved to Agile …With Limitations

The National Security Agency (NSA) successfully has moved to an Agile-iterative model for much of its software development since 2012. Tools and in-house expertise have been built, allowing defense contractors to contribute and bring mission experience. However, the NSA essentially owns the software factory and buys software development by the hour from the contractors.

Using modern commercial tools combined with NSA-approved encryption and security measures, teams of multiple contractors at multiple locations can collaborate simultaneously.

This model has been successful but with some limitations:

- The model is typically used for systems with stable hardware processing environments only.
- The NSA defines and manages the development process. While contractors apply specific local expertise and write most of the code, the NSA tightly manages the process and metrics. This requires highly trained and capable program managers on the part of the customers who are experts in the Agile process.
- Since the NSA manages the process and buys software development by the hour, contractors do not develop intellectual property and, therefore, do not have business-case incentives for large investments to advance the relevant technologies. This lack of investment is often disappointing to U.S. Government customers who seek more industry investments in these areas.
- The NSA is intimately involved in the daily development of the software by the contractor.

4. Finding: Acquisition Strategies and Contracting Approaches

4.1 Software Acquisition Misalignment

For almost a decade, experts — including the DSB[16] — have noted the linear defense acquisition process is ill-suited to accommodate how modern IT is produced, adapted commercially, and employed. Today, modern software development only makes this misalignment more glaringly apparent. The fundamental mechanisms used by the DoD and the defense industrial base for achieving a fielded capability — acquisition strategies, RFPs and source selection, and contracting — are not aligned with the realities of current continuous software development and deployment as practiced commercially. In the rare cases where modern software approaches were truly used (not just claimed to be used) in national security mission systems, nonstandard acquisition and contracting approaches were used to work around the standard system. This is unacceptable and must be fixed.

In the development of complex weapons systems, the largely accepted contracting approaches tend to use cost reimbursable type contracts, with specifics detailed in the categories and variance in the incentives (i.e., incentive fee or award fee). Once in production, the DoD often switches to fixed-price contracts, again using award fee or incentive approaches. In sustainment, the DoD often goes to services contracts — the exemplar being the best practice of performance-based logistics service contracts.

The Task Force found contracting approaches and incentive structures for software intensive systems need to be updated to enable and encourage the DoD and its contractors to begin using continuous iterative development when applicable. The speed of modern software iterations (i.e., sprints) and the agility required to change specifications quickly necessitates a new approach.

Without prescribing the exact answer, the Task Force found best practices for a new approach to developing and acquiring software should include:

- contracting software development as a service (e.g., the United States paying for contracted software development as a service);
- paying for the overall outcome as a service (e.g., paying SpaceX for space launches or problem-based learning for sustainment programs); and
- fixed-price development programs where hybrid iterative approaches have been adopted by industry to control costs.

The classic acquisition metrics include cost, schedule, and performance. The classic phases of acquisition include development, production, and sustainment. However, modern software is in

[16] Defense Science Board, "Defense Science Board Task Force on Department of Defense Policies and Procedures for the Acquisition of Information Technology," *U.S. Department of Defense,* March 2009, https://www.acq.osd.mil/dsb/reports/2000s/ADA498375.pdf.

continuous development. This creates a misalignment between the DoD's processes and the reality of contemporary best practices.

Average Acquisition Category (ACAT) I development programs develop schedules for five years, lasting from Milestones B to C. The initial development actually takes closer to seven years, with a follow-on capability provided every two years.

Money allocation (i.e., colors of money) and funding distribution phases are not well-aligned with how software is developed today. The closest DoD analogies are Planning, Performance, Process & Innovative Solutions, Inc. (P3I), smaller ACAT programs, life-extension, routine sustainment, or the U.S. Special Operations Command (USSOCOM) Major Force Program 11 (MFP-11). These DoD acquisition approaches feature upgrades to existing systems handled within the acquisition system. However, the approaches are all aimed at different purposes and were designed in an era where continuous iterative development was not available and widely employed.

As an example of the mismatch between traditional acquisition metrics and modern software development, it is useful to consider the independent U.S. watchdog, the U.S. Government Accountability Office (GAO). The GAO's annual report to Congress on the performance of the defense acquisition system compares total cost and schedule across all colors of money per program values of the current year to previous years. It also compares changes from the original estimate from previous years. This leads to conclusions by the GAO that are often misleading. The misperceptions created by many GAO reports are difficult to rectify because the reports attract headlines in the press that already fit into preconceived notions about government waste, as demonstrated in the quote below:

> Over the past year, the total acquisition cost for the 79 programs in the 2015 portfolio decreased by $2.5 billion and the average schedule delay in achieving initial capability increased by 2.4 months. When assessed against first full estimates, total costs have increased by $469 billion, over 48 percent, most of which occurred over five years ago. The average delay in delivering initial capabilities has increased to almost 30 months.[17]

The National Reconnaissance Office (NRO) has developed a useful method for cost estimation, which is explained in **Box 5**.

[17] Michael J. Sullivan, "Weapons Acquisition Program Outcomes and Efforts to Reform DOD'S Acquisition Process," *U.S. Government Accountability Office*, 2016, http://www.dtic.mil/dtic/tr/fulltext/u2/1016830.pdf.

5 Box 5: National Reconnaissance Office Best Practice: Database of Historic Cost Actuals for Software Development – Waterfall or Agile

Cost estimation at the start of software intensive DoD programs is difficult. Most independent cost estimates (i.e., the Independent Cost Estimate (ICE) performed by the Cost Assessment and Program Evaluation (CAPE) or Service Cost Estimators) use outdated SLOC-based cost models. CAPE and Service Cost Estimators' historic cost data appear sparse. SLOC-based assumptions are compared to historical "comparables" with mixed results in matching program actuals.

- The NRO established a contractual relationship with all of their major prime contractors to provide internal cost data software by the contractor.

4.2 Defense Acquisition Could Use Continuous Iterative Development in Many Types of Programs

A key finding of the Task Force is defense acquisition and weapons systems can exploit modern continuous development techniques, including for mission critical systems and subsystems. Ongoing acquisition programs – whether in development, production, or sustainment – should be tailored to their acquisition strategy, systems architecture, and maturity.

Ongoing Small-scale Major Development Programs (Hybrid Model)

Ongoing small-scale major development programs (e.g., KC-46A fixed-price development, see **Box 2**) may still be done using iterative development at a small scale provided the end product remains unchanged (i.e., meets specifications of the contract). Typically, overall technical specifications are derived from the requirements — a perfected statement of the need as expressed by the user. Precise specifications are typically enshrined in the contract, which is awarded at the beginning of development (e.g., Milestone B) in traditional defense acquisition programs.

Ongoing Large-scale Major Development Programs

In the course of incremental developments, the designer and/or the customer may find original specifications incomplete, overly ambitious, too conservative, or otherwise undesirable due to technology change or warfighter need. If software is incrementally built and tested — and the user is exposed to interim products — alternatives may become apparent. Thus, iterative development opportunities will emerge for large-scale major development programs (e.g., F-35). However, to change a requirement in an ongoing program, the law requires a Configuration Steering Board (CSB) review the issue, a formal process that may require a lengthy staffing and high-level approval process – the opposite of Agile.

New Programs

New programs provide a clean slate opportunity for iterative development from the beginning. An alternative acquisition approach could be to compete software development as a service

where source selection is chosen for its "best value" for mission success. In this case, multiple vendors should be considered and the deliverable considered a service rather than a product.

Legacy Programs

Even in cases where development is complete, there is still an opportunity to utilize the benefits of iterative development, demonstrated in **Box 6**.

6 Box 6: Example of Legacy Program Moving to Iterative Development: Tomahawk

Tomahawk is currently executing a streamlined, hybrid-Agile approach, with good results. The development approach for Tomahawk add-on, however, is still Waterfall. The program is conducting two-week long sprints over a defined period of time (i.e., the Waterfall spiral time) with the goal of discovering defects earlier, not necessarily shortening the time to completion. The benefit of this process is that shorter sprints allow for periodic deliveries for early integration and testing, as well as cyber scans. This approach will be implemented in full in the next baseline *(Tactical Tomahawk Weapons Control System v5.6.1).*

5. Recommendations

Recommendation 1: Software Factory

A key evaluation criterion in the source selection process should be the efficacy of the offeror's software factory.

The **Under Secretary of Defense for Research and Engineering (USD(R&E))** should immediately task the **Defense Digital Service (DDS)**, the **U.S. Air Force Life Cycle Management Center (LCMC)**, the **Software Engineering Institute (SEI) Federally Funded Research and Development Center (FFRDC)**, the **U.S. Naval Air Systems Command (NAVAIR),** and the **Army Materiel Command (AMC)** to establish a common list of source selection criteria for evaluating software factories for use throughout the Department (see *Appendix E* for suggested draft criteria). To be considered minimally viable for a proposal, competing contractors should have to demonstrate *at least* a pass-fail ability to construct a software factory. The criteria should be reviewed and updated every five years.

The DoD has limited iterative development expertise. Focusing this expertise during source selection uses this limited talent in the most efficient way.

Recommendation 2: Continuous Iterative Development

The DoD and its defense industrial base partners should adopt continuous iterative development best practices for software, including through sustainment.

The **Service Acquisition Executives (SAE)**, with the **program executive officers (PEOs)**, the **program managers (PMs)**, and the **Joint Staff/J-8**, should, over the next year, identify minimum viable product (MVP) approaches and delegate acquisition authority to the PM (cascade approach), providing motivation to do MVP and work with the users to:

- deliver a series of viable products (starting with MVP) followed by successive next viable products (NVPs);
- establish MVP and the equivalent of a product manager for each program in its formal acquisition strategy, and arrange for the warfighter to adopt the initial operational capability (IOC) as an MVP for evaluation and feedback; and
- engage Congress to change statutes to transition Configuration Steering Boards (CSB) to support rapid iterative approaches (Fiscal Year (FY) 2009 National Defense Authorization Act (NDAA), Section 814).

The **Defense Acquisition Executive (DAE)** and the **SAE** or the **Milestone Decision Authority (MDA)** (i.e., PEO or PM) should require all programs entering Milestone B to implement these iterative processes for Acquisition Category (ACAT) I, II, and III programs. The goal is not to be overly prescriptive, and the details should be tailored to each program. Progress should be made on this action by summer 2018.

The **SAE** should identify best practices and decide how to incorporate these practices into regular program reviews (e.g., the Defense Acquisition Boards (DABs), the Internal Program Reviews (IPRs), and the Service Review Boards), and waivers should be done only by exception.

Recommendation 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.

The **MDA** (with the **DAE**, the **SAE**, the **PEO**, and the **PM**) should allow multiple vendors to begin work. A down-select should happen after at least one vendor has proven they can do the work, and should retain several vendors through development to reduce risk, as feasible.

The **MDA** with the **Cost Assessment and Program Evaluation office (CAPE)**, the **USD(R&E)**, the **Service Cost Estimators**, and others should modernize cost and schedule estimates and measurements. They should evolve from a pure SLOC approach to historical comparables as a measurement, and should adopt the National Reconnaissance Office (NRO) approach (demonstrated in **Box 5**) of contracting with the defense industrial base for work breakdown schedule data to include, among others, staff, cost, and productivity.

The **MDA** should immediately require the **PM** to build a program-appropriate framework for status estimation. Example metrics include:[18]

- Sprint Burndown: tracks the completion of work throughout the sprint.
- Epic and Release Burndown: tracks the progress of development over a larger body of work than a sprint.
- Velocity: the average amount of work a team completes during a sprint.
- Control Chart: focus on the cycle time of individual issues—the total time from "in progress" to "complete."
- Cumulative Flow Diagram: shows whether the flow of work across the team is consistent; visually points out shortages and bottlenecks.

There may be short-term costs in transitioning to iterative development (e.g., software factory, training). However, based on the experience of the commercial sector, net costs can be expected to decrease after adopting iterative development.

Recommendations 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

[18] Such metrics should also be used by the DoD, the GAO, and Congress. For more information on Agile contracting approaches and metrics, see the U.S. Digital Services TechFAR Handbook at https://techfarhub.cio.gov/handbook/.

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.

For legacy programs where development is complete, the **USD(A&S)** should immediately task the **PMs** with the **PEOs** to make the business case for whether to transition the program.

Over the next year, the **USD(A&S)** should task the **PMs** of programs that have transitioned successfully to modern software development practices to brief best practices and lessons learned across the Services.

Recommendation 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.

Over the next two years, the **service acquisition commands** (e.g., the **LCMC**, the **NAVAIR**, the **U.S. Naval Sea Systems Command (NAVSEA)**, and the **AMC**) need to develop workforce competency and a deep familiarity of current software development techniques. To do so, they should acquire or access a small cadre of software systems architects with a deep understanding of iterative development. Services acquisition commands should use this cadre early in the acquisition process to formulate acquisition strategy, develop source selection criteria, and evaluate progress. The goal is to ensure software development expertise is established as core to the program and to ensure the mission is done in smaller pieces with functionality at each step.

Beyond development of coders and developers, there is a need for software-informed PMs, sustainers and software acquisition specialists. In 2018, the **Service Acquisition Career Managers** should develop a training curriculum to create and train this cadre. The **SAE** and the **PEO** should ensure the PMs of software-intensive programs are knowledgeable about software and with software acquisition training. The **USD(A&S)** and the **USD(R&E)** should direct the **Defense Acquisition University (DAU)** to establish curricula addressing modern software practices leveraging expertise from the DDS, the FFRDCs, and the University Affiliated Research Centers (UARCs).

Defense prime contractors must build internal competencies in modern software methodologies. Starting immediately, the **chief executive officers (CEOs) of DoD prime contractors** should brief the **USD(A&S)** at least annually to demonstrate progress on adapting modern software practices, including their corporations' proficiencies in establishing effective software factories.

Working with **Congress** in 2018, the **DoD career functional Integrated Product Team (IPT) leads** should immediately establish a special software acquisition workforce fund modeled after the Defense Acquisition Workforce Development Fund (DAWDF), the purpose of which is to hire and

train a cadre of modern software acquisition experts across the Services. The objective is to have 500 or more software acquisition experts per year starting in FY2019.

Within FY2019, the **PMs** should create an iterative development IPT with associated training. The **Service Chiefs** should delegate the role of Product Manager to these IPTs.

Recommendation 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.

Availability, cost, compatibility, and licensing restrictions of such framework elements to the U.S. Government and its contractors should also be part of the selection criteria for contract award.

During the RFP-phase, proposers may designate pre-existing components not developed under the proposal but used or delivered as part of the project. However, limitations related to use or access to underlying design information (including components designed using the software factory approach) may also be a selection criteria.

Except for such pre-existing components, all documentation, test files, coding, application programming interfaces (APIs), design documents, results of fault, performance tests conducted using the framework, and tools developed during the development, as well as the software factory framework, should be:

- delivered to the U.S. Government at each production milestone; or
- escrowed and delivered at such times specified by the U.S. Government (i.e., end of production, contract reward).

Selection preference should be granted based on the ability of the United States to reconstitute the software framework and rebuild binaries, re-run tests, procedures, and tools against delivered software, and documentation. These requirements should flow down to subcontractors and suppliers subject to reasonable restrictions affecting use, duplication, and disclosure of material not originally created as part of the development agreement.

Recommendation 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.

The Department must invest to build a better posture in this critical technology. Under the leadership and immediate direction of the **USD(R&E)**, the **Defense Advanced Research Projects Agency (DARPA),** the **SEI FFRDC,** and the **DoD laboratories** should establish research and

experimentation programs around the practical use of machine learning in defense systems with efficient testing, independent verification and validation (IVV), and cybersecurity resiliency and hardening as the primary focus points. They should establish a machine learning and autonomy data repository and exchange along the lines of the U.S. Computer Emergency Readiness Team (US-CERT) to collect and share necessary data from and for the deployment of machine learning and autonomy. They should create and promulgate a methodology and best practices for the construction, validation, and deployment of machine learning systems, including architectures and test harnesses.

Appendix A: Task Force Terms of Reference

THE UNDER SECRETARY OF DEFENSE
3010 DEFENSE PENTAGON
WASHINGTON, DC 20301-3010

ACQUISITION,
TECHNOLOGY,
AND LOGISTICS

AUG 03 2016

MEMORANDUM FOR CHAIRMAN, DEFENSE SCIENCE BOARD

SUBJECT: Terms of Reference – Defense Science Board Task Force on the Design and Acquisition of Software for Defense Systems

The Department has always been challenged to develop software intensive products within cost and schedule and with the desired performance. Cyber security, steadily increased functionality (including artificial intelligence and autonomy features), and a growing desire for tightly networked systems all lead to more complex software intensive programs.. This is true for weapons systems platforms—such as F-35 and Aegis—as well as for logistics and other systems that support the Warfighter.

The software challenge becomes more important in light of the emerging threats and promising capabilities that could greatly affect future military systems. Building and upgrading systems to be resilient to cyber vulnerabilities and threats has never been more important. Software for systems that must function in a complex and constantly changing environment must be developed using techniques appropriate to those demands. When systems attributes include adaptive cognitive capabilities, the challenges are only greater. Software that "learns" greatly complicates software testing insofar as it changes itself every time the system is used. Commercial software development is also constantly and quickly evolving in response to both technological opportunity and competitive pressures; the Department and the Defense industrial base need to capitalize on the opportunities provided by commercial sector improvements in software development techniques and practices.

For these reasons, this Task Force is established to examine the current state of DoD software acquisition and recommend practical actions to improve performance by the DoD and its suppliers.. The Task Force will consider development, test and evaluation of learning systems. Since the DSB examined software acquisition in 2007, there has been considerable attention on improving acquisition outcomes; the Weapons System Acquisition Reform Act in 2009 and the Better Buying Power initiatives from 2010 to 2016 are just two notable examples. Given these changes, has the acquisition of software intensive systems by DoD improved or is it lagging behind? Topics to be considered include but are not limited to the following:

- Contrast and compare DOD and commercial software development. How current are traditional defense contractors and DOD program offices on software development capabilities? What emerging and state-of-the-art commercial software development capabilities should military systems embrace?

- What impediments exist in the DOD requirements, contracting and program management practices to the use of more advanced software development processes and how can they be removed?

- Do we need, for example, expansion of the use of special authorities like Federal Acquisition Regulation Part 12 (Other Transactional Authorities)? What about "plug fest" type approaches? For agile approaches, how is the requirements process best addressed?

- Do "agile" software approaches live up to their promise? Have DoD and DOD suppliers applied "agile" techniques effectively? What are the impediments to more extensive use of "agile" approaches in the Department?

- Should the DoD adopt the commercial concept of a minimum viable product?

- What are best management approaches to achieving rapid and effective software upgrades in military systems? Are modular open architectures (hardware and software) being used effectively? If so, how does one perform required functions like testing and information assurance assessments in a timely manner?

- What lessons can be learned from those DoD programs with recent major software challenges, such as F-35 mission software builds (e.g., Blocks 2B, 3i, 3F) and ALIS, and/or next generation GPS Ground Control System?

- What specific recommendations would be made to ensure rapid adoption of cognitive capabilities as they mature?

I will sponsor the study. The Honorable William LaPlante and Mr. Robert Wisnieff will serve as co-chairmen of the study. Mr. James J. Thompson, Office of the Under Secretary of Defense for Acquisition, Technology, and Logistics, will serve as Executive Secretary for the study. Lt. Col Victor Osweiler, U.S. Air Force, will serve as the Defense Science Board Secretariat Representative.

The task force members are granted access to those Department of Defense officials and data necessary for the appropriate conduct of their study. The Under Secretary of Defense for Acquisition, Technology, and Logistics will serve as the DoD decision-maker for the matter under consideration and will coordinate decision-making as appropriate with other stakeholders identified by the study's findings and recommendations. The nominal start date of the study period will be within 3 months of signing this Terms of Reference and the study period will be between 9 to 12 months. The final report will be completed within 6 months from the end of the study period. Extensions for unforeseen circumstances will be handled accordingly.

The study will operate in accordance with the provisions of Public Law 92-463, the "Federal Advisory Committee Act," and DoD Directive 5105.04, "Department of Defense Federal Advisory Committee Management Program." It is not anticipated that this study will need to go into any "particular matters" within the meaning of title 18, United States Code, section 208, nor will it cause any member to be placed in the position of action as a procurement official.

Frank Kendall

Appendix B: Task Force Membership

Chairs

Dr. William LaPlante
MITRE

Dr. Robert Wisnieff
IBM

Members

Dr. Victoria Coleman
Wikimedia

Dr. Paul Nielsen
Carnegie Mellon University

Mr. Christopher Lynch
Defense Digital Services

Dr. Fred Schneider
Cornell University

Dr. Joe Markowitz
Unaffiliated

Mr. Lou Von Thaer
Batelle

Mr. Robert Nesbit
Unaffiliated

Mr. Alfonso Velosa
Gartner, Inc.

Government Advisors

Ms. Cynthia Schurr
U.S. Air Force (SAF/AQ)

Mr. Joseph Heil
Naval Surface Warfare Center, Dahlgren Division

Executive Secretary

Mr. James Thompson
Office of the Deputy Assistant Secretary of Defense for Systems Engineering

Defense Science Board Secretariat

Edward C. Gliot
Executive Director, Acting
(beginning August 2017)

Lt Col Victor Osweiler, USAF
Deputy for Operations
(October 2016-August 2017)

Karen D. H. Saunders
Executive Director
(October 2016-August 2017)

Lt Col Milo Hyde, USAF
Deputy for Operations
(beginning October 2017)

Study Support

Ms. Sarah Gamberini
SAIC

Ms. Brenda Poole
SAIC

Ms. Ashley Conner
SAIC

Mr. Ari Kattan
SAIC

Appendix C: Recommendations

Recommendation 1: Software Factory

A key evaluation criterion in the source selection process should be the efficacy of the offeror's software factory.

The **Under Secretary of Defense for Research and Engineering (USD(R&E))** should immediately task the **Defense Digital Service (DDS)**, the **U.S. Air Force Life Cycle Management Center (LCMC)**, the **Software Engineering Institute (SEI) Federally Funded Research and Development Center (FFRDC)**, the **U.S. Naval Air Systems Command (NAVAIR),** and the **Army Materiel Command (AMC)** to establish a common list of source selection criteria for evaluating software factories for use throughout the Department (see *Appendix E* for suggested draft criteria). To be considered minimally viable for a proposal, competing contractors should have to demonstrate *at least* a pass-fail ability to construct a software factory. The criteria should be reviewed and updated every five years.

The DoD has limited iterative development expertise. Focusing this expertise during source selection uses this limited talent in the most efficient way.

Recommendation 2: Continuous Iterative Development

The DoD and its defense industrial base partners should adopt continuous iterative development best practices for software, including through sustainment.

The **Service Acquisition Executives (SAE)**, with the **program executive officers (PEOs)**, the **program managers (PMs)**, and the **Joint Staff/J-8**, should, over the next year, identify minimum viable product (MVP) approaches and delegate acquisition authority to the PM (cascade approach), providing motivation to do MVP and work with the users to:

- deliver a series of viable products (starting with MVP) followed by successive next viable products (NVPs);
- establish MVP and the equivalent of a product manager for each program in its formal acquisition strategy, and arrange for the warfighter to adopt the initial operational capability (IOC) as an MVP for evaluation and feedback; and
- engage Congress to change statutes to transition Configuration Steering Boards (CSB) to support rapid iterative approaches (Fiscal Year (FY) 2009 National Defense Authorization Act (NDAA), Section 814).

The **Defense Acquisition Executive (DAE)** and the **SAE** or the **Milestone Decision Authority (MDA)** (i.e., PEO or PM) should require all programs entering Milestone B to implement these iterative processes for Acquisition Category (ACAT) I, II, and III programs. The goal is not to be overly prescriptive, and the details should be tailored to each program. Progress should be made on this action by summer 2018.

The **SAE** should identify best practices and decide how to incorporate these practices into regular program reviews (e.g., the Defense Acquisition Boards (DABs), the Internal Program Reviews (IPRs), and the Service Review Boards), and waivers should be done only by exception.

Recommendation 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.

The **MDA** (with the **DAE**, the **SAE**, the **PEO**, and the **PM**) should allow multiple vendors to begin work. A down-select should happen after at least one vendor has proven they can do the work, and should retain several vendors through development to reduce risk, as feasible.

The **MDA** with the **Cost Assessment and Program Evaluation office (CAPE)**, the **USD(R&E)**, the **Service Cost Estimators**, and others should modernize cost and schedule estimates and measurements. They should evolve from a pure SLOC approach to historical comparables as a measurement, and should adopt the National Reconnaissance Office (NRO) approach (demonstrated in **Box 5**) of contracting with the defense industrial base for work breakdown schedule data to include, among others, staff, cost, and productivity.

The **MDA** should immediately require the **PM** to build a program-appropriate framework for status estimation. Example metrics include:[19]

- Sprint Burndown: tracks the completion of work throughout the sprint.
- Epic and Release Burndown: tracks the progress of development over a larger body of work than a sprint.
- Velocity: the average amount of work a team completes during a sprint.
- Control Chart: focus on the cycle time of individual issues—the total time from "in progress" to "complete."
- Cumulative Flow Diagram: shows whether the flow of work across the team is consistent; visually points out shortages and bottlenecks.

There may be short-term costs in transitioning to iterative development (e.g., software factory, training). However, based on the experience of the commercial sector, net costs can be expected to decrease after adopting iterative development.

Recommendation 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

[19] Such metrics should also be used by the DoD, the GAO, and Congress. For more information on Agile contracting approaches and metrics, see the U.S. Digital Services TechFAR Handbook at https://techfarhub.cio.gov/handbook/.

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.

For legacy programs where development is complete, the **USD(A&S)** should immediately task the **PMs** with the **PEOs** to make the business case for whether to transition the program.

Over the next year, the **USD(A&S)** should task the **PMs** of programs that have transitioned successfully to modern software development practices to brief best practices and lessons learned across the Services.

Recommendation 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.

Over the next two years, the **service acquisition commands** (e.g., the **LCMC**, the **NAVAIR**, the **U.S. Naval Sea Systems Command (NAVSEA)**, and the **AMC**) need to develop workforce competency and a deep familiarity of current software development techniques. To do so, they should acquire or access a small cadre of software systems architects with a deep understanding of iterative development. Services acquisition commands should use this cadre early in the acquisition process to formulate acquisition strategy, develop source selection criteria, and evaluate progress. The goal is to ensure software development expertise is established as core to the program and to ensure the mission is done in smaller pieces with functionality at each step.

Beyond development of coders and developers, there is a need for software-informed PMs, sustainers and software acquisition specialists. In 2018, the **Service Acquisition Career Managers** should develop a training curriculum to create and train this cadre. The **SAE** and the **PEO** should ensure the PMs of software-intensive programs are knowledgeable about software and with software acquisition training. The **USD(A&S)** and the **USD(R&E)** should direct the **Defense Acquisition University (DAU)** to establish curricula addressing modern software practices leveraging expertise from the DDS, the FFRDCs, and the University Affiliated Research Centers (UARCs).

Defense prime contractors must build internal competencies in modern software methodologies. Starting immediately, the **chief executive officers (CEOs) of DoD prime contractors** should brief the **USD(A&S)** at least annually to demonstrate progress on adapting modern software practices, including their corporations' proficiencies in establishing effective software factories.

Working with **Congress** in 2018, the **DoD career functional Integrated Product Team (IPT) leads** should immediately establish a special software acquisition workforce fund modeled after the

Defense Acquisition Workforce Development Fund (DAWDF), the purpose of which is to hire and train a cadre of modern software acquisition experts across the Services. The objective is to have 500 or more software acquisition experts per year starting in FY2019.

Within FY2019, the **PMs** should create an iterative development IPT with associated training. The **Service Chiefs** should delegate the role of Product Manager to these IPTs.

Recommendation 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.

Availability, cost, compatibility, and licensing restrictions of such framework elements to the U.S. Government and its contractors should also be part of the selection criteria for contract award.

During the RFP-phase, proposers may designate pre-existing components not developed under the proposal but used or delivered as part of the project. However, limitations related to use or access to underlying design information (including components designed using the software factory approach) may also be a selection criteria.

Except for such pre-existing components, all documentation, test files, coding, application programming interfaces (APIs), design documents, results of fault, performance tests conducted using the framework, and tools developed during the development, as well as the software factory framework, should be:

- delivered to the U.S. Government at each production milestone; or
- escrowed and delivered at such times specified by the U.S. Government (i.e., end of production, contract reward).

Selection preference should be granted based on the ability of the United States to reconstitute the software framework and rebuild binaries, re-run tests, procedures, and tools against delivered software, and documentation. These requirements should flow down to subcontractors and suppliers subject to reasonable restrictions affecting use, duplication, and disclosure of material not originally created as part of the development agreement.

Recommendation 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.

The Department must invest to build a better posture in this critical technology. Under the leadership and immediate direction of the **USD(R&E)**, the **Defense Advanced Research Projects Agency (DARPA),** the **SEI FFRDC,** and the **DoD laboratories** should establish research and experimentation programs around the practical use of machine learning in defense systems with efficient testing, independent verification and validation (IVV), and cybersecurity resiliency and hardening as the primary focus points. They should establish a machine learning and autonomy data repository and exchange along the lines of the U.S. Computer Emergency Readiness Team (US-CERT) to collect and share necessary data from and for the deployment of machine learning and autonomy. They should create and promulgate a methodology and best practices for the construction, validation, and deployment of machine learning systems, including architectures and test harnesses.

Appendix D: Briefings Received

18 October 2016 Meeting

Summary of Past Studies
Defense Science Board

OCX GPS
Former Commander, Space, and Missile Systems Center and Program Executive Officer for Space

Joint Strike Fighter F-35 Program Office
Program Executive Officer for the F-35 Lightning II Joint Program Office

DoD Software Challenges: Acquisition Program Performance, with Additional Considerations
Office of the Deputy Assistant Secretary of Defense for Systems Engineering

28-29 November 2016 Meeting

Cost Assessment
CAPE

Intelligence Software Acquisition
Director National Intelligence Division, OUSD(AT&L) SSI; ODNI/SRA, Director Cost Analysis; NRO Director, Mission Processing Systems Program Office; NGA Program Manager; NSA Program Manager

Contracting: Performance Incentives
Defense Procurement and Acquisition Policy

Software Sustainment
Renaissance Strategic Advisors Managing Partner, Enlightenment Capital

Open Architecture
USAF Rapid Capabilities Office

Improving Software Acquisition for Aviation
U.S. Army RDECOM

21 December 2016 Meeting

Test and Evaluation
DASD(Developmental Test and Evaluation), Director, Test Resource Management Center

Defense Digital Service Overview
Defense Digital Service

Why Contractors Think the Way They Do
Defense Science Board

Software Challenges and Best Practices
Deputy Chief Engineer for the Naval Surface Warfare Center Dahlgren Division

Security and Software
Defense Science Board

7-8 February 2017 Meeting

Google's Practices for Developing and Testing Memory-safe C++ Code
Software Engineer, Google

IBM Agile
Vice President, Agile, Talent, and Business Management, IBM

IBM Blockchain
IBM

Large-scale Systems at Facebook
Engineering Director, Release Engineering, Facebook

Kaggle: The Home of Data Science
CEO, Kaggle

Software Security
Qualcomm

Brave Software
CEO and Founder, Brave Software Inc.

Commercial vs. Government Software Development
Director Advanced Technology and Projects, Google

7-8 March 2017 Meeting

Raytheon
CEO, Raytheon Company

18F
Innovation Specialist, General Services Administration-18F

Acquisition Reform
House Armed Services Committee

4 April 2017 Meeting

Defense Digital Service Program Reports Review
Air Force Digital Service and Defense Digital Service

Air Force Expeditionary Combat Support System (ECSS)
Cyber/Netcentric Directorate Deputy Director

Recent Advances in Deep Learning
Carnegie Mellon University

Intellectual Property
Former Director of Defense Procurement and Acquisition Policy

2-3 May 2017 Meeting

Lockheed Martin
Executive Vice President of Lockheed Martin Aeronautics

Air Operations Center (AOC) 10.2
Program Executive Office for Battle Management, Air Force Life Cycle Management Center

5-6 June 2017 Meeting

Software Development for Command, Control, and Communications (C3) Cyber, Business Systems (C3CB)
DASD C3CB

Boeing
Senior Technical Fellow, Software and Systems, Boeing

Code for America
Co-founder United States Digital Service

SpaceX Teleconference
SpaceX Development Team

Appendix E: Software Factory Source Selection Criteria Suggestions

- Configuration management software (e.g., Puppet, Chef, Ansible)
- Continuous integration (build and test) systems (e.g., Travis CI for hosted service, Jenkins for open source application)
- Scripts and code used to release software (e.g., Python scripts)
- Servers, network, or other infrastructure that support release tools
- Software and tools to support developer self-service operations (New Relic for application performance over time, diagnostic tools, monitoring)
- External test frameworks (e.g., Jersey Test Framework, TestPlant/eggPlant)
- External operational monitoring and log mining tools (e.g., Splunk, Elasticsearch + Logstash + Kibana (ELK) Stack)
- Source code repositories (e.g., Github for hosted service, GitLab for open source application)
- Issue tracking systems (e.g., JIRA, Trello, GitHub)
- Container driven tools (e.g., Docker, Elastic Container Service (Amazon Web Services (AWS)), Kubernetes)
- Requirements management (e.g., DOORS (Dynamic Object Oriented Requirements System), Blueprint)
- Infrastructure and cloud providers (e.g., AWS, Rackspace, Azure, Red Hat OpenShift, Pivotal Cloud Foundry)
- Integrated development environment (IDE) DevOps process

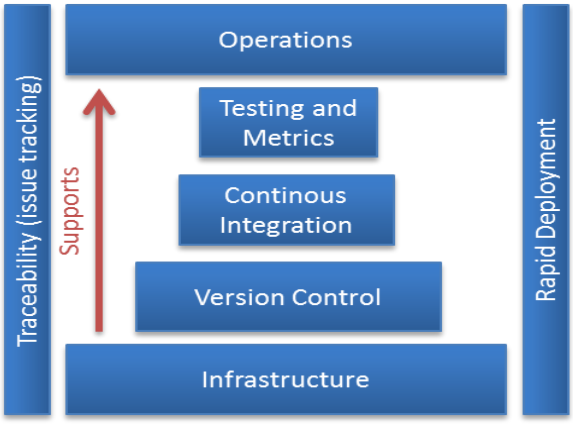


Figure E-1. Software Factory in Source Selection

Appendix F: Acronyms and Abbreviated Terms

ACAT	Acquisition Category
AMC	Army Materiel Command
AOC	Air Operations Center
API	application programing interfaces
ASEE	American Society for Electrical Engineering
AWS	Amazon Web Services
C3CB	Command, Control, and Communications, Cyber, Business Systems
CAPE	Cost Assessment and Program Evaluation office
CEO	Chief Executive Officer
COCOMO	Constructive Cost Model
CSB	Configuration Steering Board
DAB	Defense Acquisition Board
DAE	Defense Acquisition Executive
DARPA	Defense Advanced Research Projects Agency
DASD(C3CB)	Deputy Assistant Secretary of Defense for C3CB
DASD(DT&E)	Deputy Assistant Secretary of Defense for Development, Testing, and Evaluation
DASD(R&E)	Deputy Assistant Secretary of Defense for Research and Engineering
DAU	Defense Acquisition University
DAWDF	Defense Acquisition Workforce Development Fund
DDS	Defense Digital Service
DNI	Director of National Intelligence
DoD	Department of Defense
DOORS	Dynamic Object Oriented Requirements System
DSB	Defense Science Board
ECSS	Expeditionary Combat Support System
ELK	Elasticsearch + Logstash + Kibana
EW	electronic warfare
FFRDC	Federally Funded Research and Development Center
FY	fiscal year
GAO	Government Accountability Office

GPS	global positioning system
ICE	independent cost estimate
IDE	integrated development environment
IEEE	Institute of Electrical and Electronics Engineers
IOC	initial operational capability
IPR	Internal Program Review
IPT	Integrated Product Team
IT	information technology
IVV	independent verification and validation
LCMC	Air Force Life Cycle Management Center
LOC	Lines of Code
MDA	Milestone Decision Authority
MFP	Major Force Program
MVP	minimum viable product
NAVAIR	U.S. Naval Air Systems Command
NAVSEA	U.S. Naval Sea Systems Command
NDAA	National Defense Authorization Act
NIST	National Institute of Standards and Technology
NRO	National Reconnaissance Office
NSA	National Security Agency
NSWC	U.S. Naval Surface Warfare Center
NVP	next viable product
OCX	Raytheon GPS operational control system
ODASD(SE)	Office of the Deputy Assistant Secretary of Defense for Systems Engineering
ODNI	Office of the Director of National Intelligence
OUSD(AT&L)	Office of the Under Secretary of Defense for Acquisition, Technology, and Logistics
P3I	Planning, Performance, Process and Innovative Solutions
PEO	program executive officer
PM	program manager
RDECOM	U.S. Army Research, Development, and Engineering Command

ROI	return on investment
RFP	request for proposal
SAE	Service Acquisition Executive
SAF/AQ	Assistant Secretary of the Air Force for Acquisition
SEI	Software Engineering Institute
SLOC	source lines of code
SRA	Systems and Resource Analyses
SSI	Space, Strategic, and Intelligence Systems
UARC	University Affiliated Research Center
US-CERT	U.S. Computer Emergency Readiness Team
USAF	U.S. Air Force
USD(A&S)	Under Secretary of Defense for Acquisition and Sustainment
USD(R&E)	Under Secretary of Defense for Research and Engineering
USSOCOM	United States Special Operations Command
XP	extreme programming

Appendix G: Glossary

Agile
Agile or continuous iterative development, where a team develops software in smaller blocks that can be incrementally evaluated by a user community

Amazon Web Services
Subsidiary of Amazon that provides on-demand cloud computing platforms on paid subscription basis

Ansible
Software that automates software provisioning, configuration management, and application deployment

architecture
Depiction of the system that aids in the understanding of how the system will behave

automated build
Process of automating the creation of a software build and the associated processes

automated test
Use of special software to control the execution of tests and comparison of actual outcomes with predicted outcomes

Azure
Microsoft cloud computing service for building, testing, deploying, and managing applications and services through a global network of Microsoft-managed data centers

beta user
In beta testing, the second phase of software testing in which a sampling of the intended audience tries the product

blueprint
Final product of a software blueprinting process

C
General purpose, imperative computer programming language

check-in notes
Used for software version control

Chef
Configuration management tool written in Ruby and Erlang, using pure-Ruby, domain-specific language for writing system configuration "recipes"

code provenance
Determining originator of the code for legal and auditing purposes

collaboration
Application software designed to help people involved in a common task to achieve goals

continuous integration
Practice of merging all developer working copies to a shared mainline several times a day

control chart
Statistical process tool to determine if manufacturing or business process is in a state of control

cumulative flow diagram
Tool used in queuing theory

cross-coupling
Interdependence between software modules

cyber red team
Group of white-hat hackers that attack organization's digital infrastructure as an attacker would in order to test organization's defenses

developer
Person concerned with facets of software development process, including research, design, programming, and testing computer software

docker
Software technology providing containers, promoted by Docker, Inc.

documentation
Written text or illustration that accompanies computer software or is embedded in the source code

DOORS
Rational Dynamic Object Oriented Requirements System (DOORS) is a requirement management tool

dynamic test
Examination of the physical response from the system to variables that are not constant and change with time

ELK Stack
Program that consists of Elasticsearch, Logstash, and Kibana programs

executable code
Coding causes a computer to perform indicated tasks according to encoded instructions

features list
Set of related requirements that allow the user to satisfy a business objective or need

fuzz testing
Automated software testing technique that involves providing invalid, unexpected, or random data as inputs to a computer program

GitHub
Web-based version control repository hosting service

GitLab
Web-based repository manager with wiki and issue tracking features, using an open source license

gtest
Google Test, a unit testing library for the C++ programming language, based on the xUnit architecture

iteration
Single development cycle, usually measured in one or two weeks

iterative development
Way of breaking down the software development of a large application into smaller chunks

Jenkins
Open source automation server written in Java

Jersey Test Framework
An external test framework

JIRA
Proprietary issues tracking product providing bug tracking, issue tracking, and project management functions, developed by Atlassian

Kubernetes
Open-source system for automating deployment, scaling, and management of containerized applications, originally designed by Google and donated to the Cloud Native Computing Foundation

minimum viable product
Development technique in which a new product or website is developed with sufficient features to satisfy early adopters

New Relic
Company that provides digital intelligence platforms and delivers application performance monitoring as a purely software as a service product

nightly build
Daily practice of executing a software build of the latest version of a program to ensure all required dependencies are present

NIST Guidelines
National Institute of Standards and Technology, a measurement standards laboratory and a non-regulatory agency of the U.S. Department of Commerce that promotes innovation and industrial competitiveness

open source
Denoting software for which the original source code is made freely available and may be redistributed and modified

overflow
A situation that occurs when more information is being transmitted than the hardware can handle

package
Application program developed for sale to the general public

peer review
Type of software review in which a product is examined by its author or one or more colleagues to evaluate its technical content and quality

Pivotal Cloud Foundry
Open-source, multi-cloud application platform for continuous delivery in supporting the full application development lifecycle

product manager
Administrator of product responsible for the strategy, roadmap, and feature definition for said product or product line

Puppet
Open-source software configuration management tool

Python
Widely used high-level programming language for general-purpose programming

Rackspace
Managed cloud computing company

Red Hat Openshift
Computer software product for container-based software deployment and management

release
Distribution of final version of an application

scripts
Program or sequence of instructions interpreted or carried out by another program

Scrum
a framework for managing Agile software development

silos
Isolated point in a system where data is kept and segregated from other parts of the architecture

software factory
low-cost, cloud-based computing used to assemble a set of tools enabling developers, users, and management to work together on a daily tempo

source code
Text listing of commands to be compiled or assembled into an executable computer program

source lines of code
Software metric used to measure size of computer program by counting the number of lines in the text of the program's source code

source repository
Central file storage location

spin
Verification system for models of distributed software systems

spirals
Model that is similar to incremental development for a system, with more emphasis placed on risk analysis

Splunk
Company that produces software for searching, monitoring, and analyzing machine-generated big data, via web-style interface

sprint
A set period of time during which specific work is to be completed and made ready for review

static testing
Dry run testing, a form of software testing where the actual program or application is not used

style checker
Computer application that identifies possible usage errors or stylistic infelicities in text; also known as grammar checker

swift
Programming language for macOS, iOS, and tvOS

system architecture
High-level structures of software system, the discipline of creating such structures, and the documentation of said structures

test harnesses
Collection of software and test data configured to test a program unit by running it under varying conditions and monitoring its behavior and outputs

TestPlant/eggPlant
Black-box graphical user interface test automation tool

tool chain
Set of programming tools used to perform complex software development task or to create a software product

Travis CI
Hosted, distributed continuous integration service used to build and test software projects hosted at GitHub

Trello
Web-based project management application

unit testing
Software testing method where individual units of source code, sets of one or more computer program modules together with associated control data, usage procedures, and operating procedures, are tested to determine whether they are fit for use

user
Person that a software program or hardware device is designed for

variable
Value that can change depending on conditions or on information passed to the program

velocity
Metric used for planning sprints and measuring team performance in software development

Waterfall
Progress flows in largely one direction through the phases of conception, initiation, analysis, design, construction, testing, deployment, and maintenance

For Agile specific terms, recommend the following source:
https://confluence.atlassian.com/agile/glossary.

Appendix H: Index

THIS PAGE LEFT INTENTIONALLY BLANK


Defense Science Board
DSB
Department of Defense