Software Engineering for Continuous Delivery of Warfighting Capability

April 2023

Office of the Executive Director
for Systems Engineering and Architecture

Office of the Under Secretary of Defense
for Research and Engineering

Washington, D.C.

Software Engineering for Continuous Delivery of Warfighting Capability

Executive Director for Systems Engineering and Architecture
Office of the Under Secretary of Defense for Research and Engineering
3030 Defense Pentagon
Washington, DC 20301
https://www.cto.mil
osd.r-e.comm@mail.mil | Attention: Software Engineering Team

Approved by

Thomas W. Simms

Acting Principal Deputy Director for Systems Engineering and Architecture

Office of the Under Secretary of Defense for Research and Engineering

Software Engineering for Continuous Delivery of Warfighting Capability
Change Record

Date	Change	Rationale

This page is intentionally blank.

Contents

Figures

Tables

This page is intentionally blank.

1 Introduction

1.1 Purpose

This guide is intended to help Department of Defense (DoD) Program Management Offices (PMOs), defense engineers, software engineers, and acquisition officials plan and execute software development in an environment of changing software technologies, software engineering practices, software requirements, and software acquisition practices. The guide assumes the reader is familiar with traditional DoD systems and software engineering practice. It provides readers with information on recent changes in policy and practice and provides lessons from DoD programs.

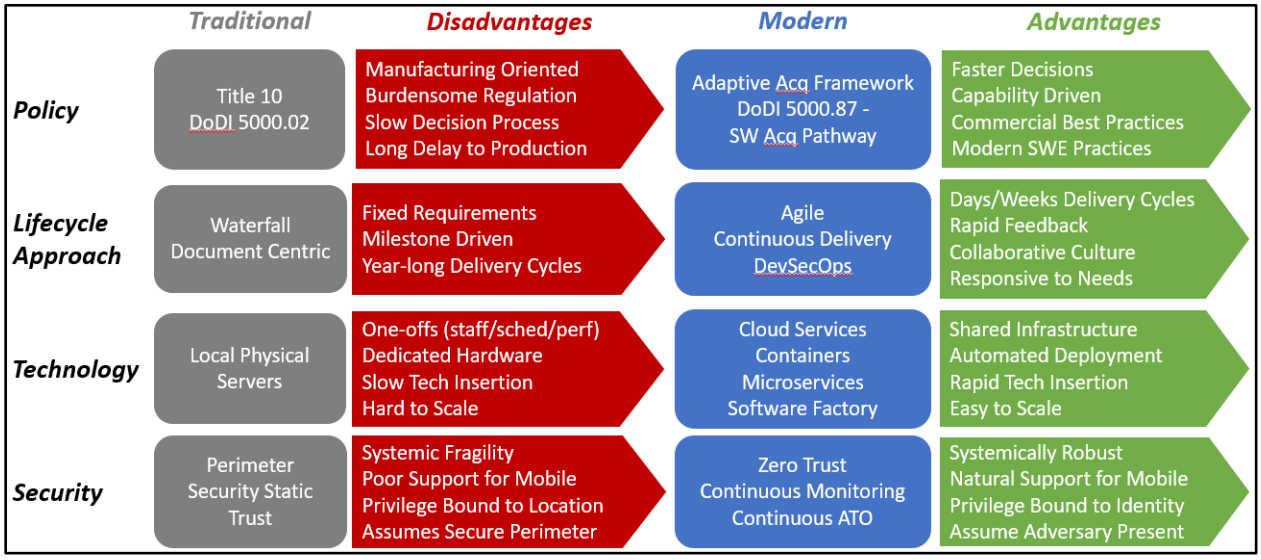
The Office of the Under Secretary of Defense for Research and Engineering (OUSD(R&E)) prepared this guide with coordination from subject matter experts (SMEs) from across the DoD Components and defense industry. The guide draws on experience and best practices from more than a decade of program engagements on software-enabled systems spanning all warfighting domains.

1.2 Overview

This guide is an element of a broader strategy to modernize software engineering and acquisition activities to deliver superior capability to the warfighter quickly, safely, and effectively, in keeping with the National Defense Strategy (NDS 2022) and the DoD mission. The principal challenge in achieving this modernization is finding ways to move from a traditional process characterized by large batches of capability delivered over long periods of time to a more continuous or Agile process characterized by delivering increments of capability over many short cycles while continuously maintaining an acceptable cybersecurity risk posture (DSB 2018) (DIB 2019b). These priorities are reflected in DoD policy and guidance issued between 2018 and early 2022 (see References).

This guide assists programs in adopting modern software development practices that apply more automation and engineering rigor to deliver better software faster. The Department is developing new software workforce competencies to support technology modernization and new ways of working. Software engineering metrics and contracting vehicles are changing how DoD manages software development in acquisition. Software technologies such as artificial intelligence (AI) and machine learning (ML) are influencing how we develop, test, and deploy the next generation of warfighting capability. This guide seeks to reinforce the defense software modernization and help acquisition programs overcome software development challenges.

Figure 1-1 describes traditional and modern software development approaches and their characteristics.



Source: OUSD(R&E) SE&A Software Team

Figure 1-1. Shift from Traditional to Modern Software Engineering Practices

1.3 Guide Organization

The guide is organized as follows:

- Section 1, Introduction, explains the purpose and organization of the guide.
- Section 2, Policy and Guidance, presents major sources that guide DoD software initiatives, including national strategy and DoD issuances.
- Section 3, Technology Modernization, discusses emerging concepts in software engineering and technology that influence DoD software development.
- Section 4, Challenges and Best Practices, discusses challenges observed in DoD software development and acquisition. It suggests how programs can adapt commercial techniques, especially Agile/Development, Security, and Operations (DevSecOps), to military systems, and it discusses challenges ranging from budgeting to security and safety.
- Section 5, Software Metrics Use, discusses Agile/DevSecOps metrics as well as more traditional metrics. The section provides information on who uses what types of metrics, why those metrics are used, and what decisions they inform. The case studies included in this section digest the direct experience of the authors of this guide.
- Section 6, Software Engineering and Workforce Competencies, outlines workforce competencies required to support modern software development and delivery.

- Section 7, Contracting for Software Engineering, offers advice on how to structure agreements for software engineering that can support rapid delivery of warfighting capability and flexibility in managing the effort.
- Section 8, Artificial Intelligence and Machine Learning, offers background on these areas of software of growing importance in warfare, including a taxonomy of AI/ML as a field, and the Department's strategic approach and vision for pushing these technologies out to the edge of our warfighting systems.

1.4 Summary

This guide helps defense acquisition programs:

- Plan and execute software development to deliver capability faster.
- Make that capability more robust and more secure.
- Adopt modern software technologies and best practices.
- Learn from recent software engineering experience on programs.
- Develop new workforce competencies in support of modern software engineering.
- Select appropriate metrics to manage and oversee software development.
- Understand how AI/ML may affect conventional practices.

2 Policy and Guidance

- DoD revised its defense acquisition process to include six pathways that programs may tailor depending on the maturity and urgency of the acquisition.
- Software Acquisition is one of the new pathways, intended to modernize software development and deliver better software faster.
- DoD is implementing the DoD Software Modernization Strategy it released in November 2021.

Between 2018 and 2022, DoD revised several areas of policy and guidance to modernize the way the Department acquires, develops, operates, and manages software across the life cycle. This section reviews significant changes. DoD policy is mandatory direction, whereas guidance provides recommendations, best practices, and lessons learned.

The policy revisions offer new options and remove old constraints, increasing the opportunity for innovation by allowing for more localized decision making by a program. The policy and guidance updates should prepare programs to employ new administrative vehicles, management constructs, and modern software engineering practices to increase resilience and help the Department develop and deliver needed capability faster.

DoD software engineering policy and guidance may be decomposed into three tiers:

- **Overarching strategic policy and guidance**, including the National Defense Strategy and Defense Science Board (DSB) advisory reports.
- **Software engineering and acquisition policy and guidance**, including the National Defense Authorization Act (NDAA) and DoD issuances such as DoD Instruction (DoDI) 5000.02, "Operation of the Adaptive Acquisition Framework (AAF)," which enables adoption of modern software engineering practices through the Software Acquisition pathway.
- **Software engineering and technology modernization policy and guidance**, such as the Federal Cloud Computing and DoD software modernization initiatives. On balance these focus more on technology than on process.

2.1 Overarching Strategic Policy and Guidance

Table 2-1 lists major sources of DoD-wide strategic guidance as of this writing.

Table 2-1. Overarching Strategic Guidance

Title	Office	Year	Guidance
National Defense Strategy	Secretary of Defense	2022	Guidance
DoD Software Modernization Strategy	Department of Defense	2021	Guidance
DoD Software Science and Technology Strategy	Under Secretary of Defense, Research and Engineering	2021	Guidance
Software Acquisition and Practices	Defense Innovation Board	2019	Guidance
Design and Acquisition of Software for Defense Systems	Defense Science Board	2018	Guidance

The National Defense Strategy (NDS Fact Sheet 2022) continues to emphasize modernizing software engineering practices as part of overall "reforms to accelerate force development, getting the technology we need more quickly, and making investments in the extraordinary people of the Department, who remain our most valuable resource."

The DoD Software Modernization Strategy (2021) identifies a vision, along with goals and objectives, with the purpose of delivering better software faster.

The Defense Innovation Board (DIB 2019b) and Defense Science Board (DSB 2018) identified software development challenges facing the Department. Both provided recommendations for ways to modernize DoD software engineering practices to deliver capability more rapidly.

2.2 Software Engineering and Acquisition

Table 2-2 lists congressional, DoD, and industry sources that pertain to software engineering acquisition ordered by year.

Table 2-2. Software Engineering and Acquisition Policy and Guidance

Title	Office	Year	Policy/Guidance
DoDI 5000.02 Operation of the Adaptive Acquisition Framework, Change 1 Effective June 8, 2022	OUSD(A&S)	2022	Policy
DoDI 5000.95 Human Systems Integration in Defense Acquisition	OUSD(R&E)	2022	Policy
Systems Engineering Guidebook	OUSD(R&E)	2022	Guidance
Engineering of Defense Systems Guidebook	OUSD(R&E)	2022	Guidance
DAU Adaptive Acquisition Framework	DAU	2022	Guidance
ISO/IEC/IEEE 14764:2022, Software Engineering – Software Life Cycle Processes – Maintenance	ISO/IEC/IEEE	2022	Guidance

Title	Office	Year	Policy/Guidance
ISO/IEC/IEEE 24641:2021(E) Systems and Software Engineering – Methods and tools for model-based systems and software engineering.	ISO/IEC/IEEE	2021	Guidance
DoD Software Science and Technology Strategy and Report to Congress	OUSD(R&E)	2021	Report
DoDI 5000.87 Operation of the Software Acquisition Pathway	OUSD(A&S)	2020	Policy
DoDI 5000.88 Engineering of Defense Systems	OUSD(R&E)	2020	Policy
DoDI 5000.89, Test and Evaluation	OUSD(R&E)/DOT&E	2020	Policy
DoDI 5000.90, Cybersecurity for Acquisition Decision Authorities and Program Managers	OUSD(A&S)	2020	Policy
Title 10 USC Chapter 327 Subchapter I, "Modular Open Systems Approach in Development of Weapon Systems"	U.S. Congress	2021	Statute
Title 10 USC Chapter 146 Section 2460, “Definition of depot-level maintenance and repair	U.S. Congress	2021	Statute
Modular Open Systems Approach (MOSA) Reference Frameworks in Defense Acquisition Programs	OUSD(R&E)	2020	Guidance
National Defense Authorization Act, 868(c), “Implementation of Recommendations of the Final Report of the Defense Science Board Task Force on the Design and Acquisition of Software for Defense Systems”	U.S. Congress	2019	Statute
DoDI 5010.44 Intellectual Property (IP) Acquisition and Licensing	OUSD(A&S)	2019	Policy
DoD Digital Modernization Strategy	DoD CIO	2019	Guidance
DoD Digital Engineering Strategy	OUSD(R&E)	2018	Guidance
DoD 4151.18, Maintenance of Military Materiel	DoD	2017	Policy
ISO/IEC/IEEE International Standard - Systems and software engineering – Life cycle management – Part 3: Guidelines for the application of ISO/IEC/IEEE 12207 (software life cycle processes)	ISO/IEC/IEEE	2017	Guidance
IEEE Std 1633-2016, IEEE Recommended Practice on Software Reliability	IEEE	2017	Guidance
Federal Information Technology Acquisition Reform Act	U.S. Congress	2014	Statute

The DoD Software Science and Technology Strategy (OUSD(R&E) Software S&T Strategy 2021) describes four strategic goals for delivering resilient software capability. The strategy is intended to "guide strategic thinking within the Department with regard to modern software development approaches and connecting the innovative capabilities developed from S&T investments. The strategy is intended to be thought provoking and to enable a culture focused on modern software development processes and tools on par with the commercial sector which the Department can leverage to insert new and innovating software capabilities quickly into DoD weapon systems" (page 4).

2.3 Software Engineering and Technology Modernization

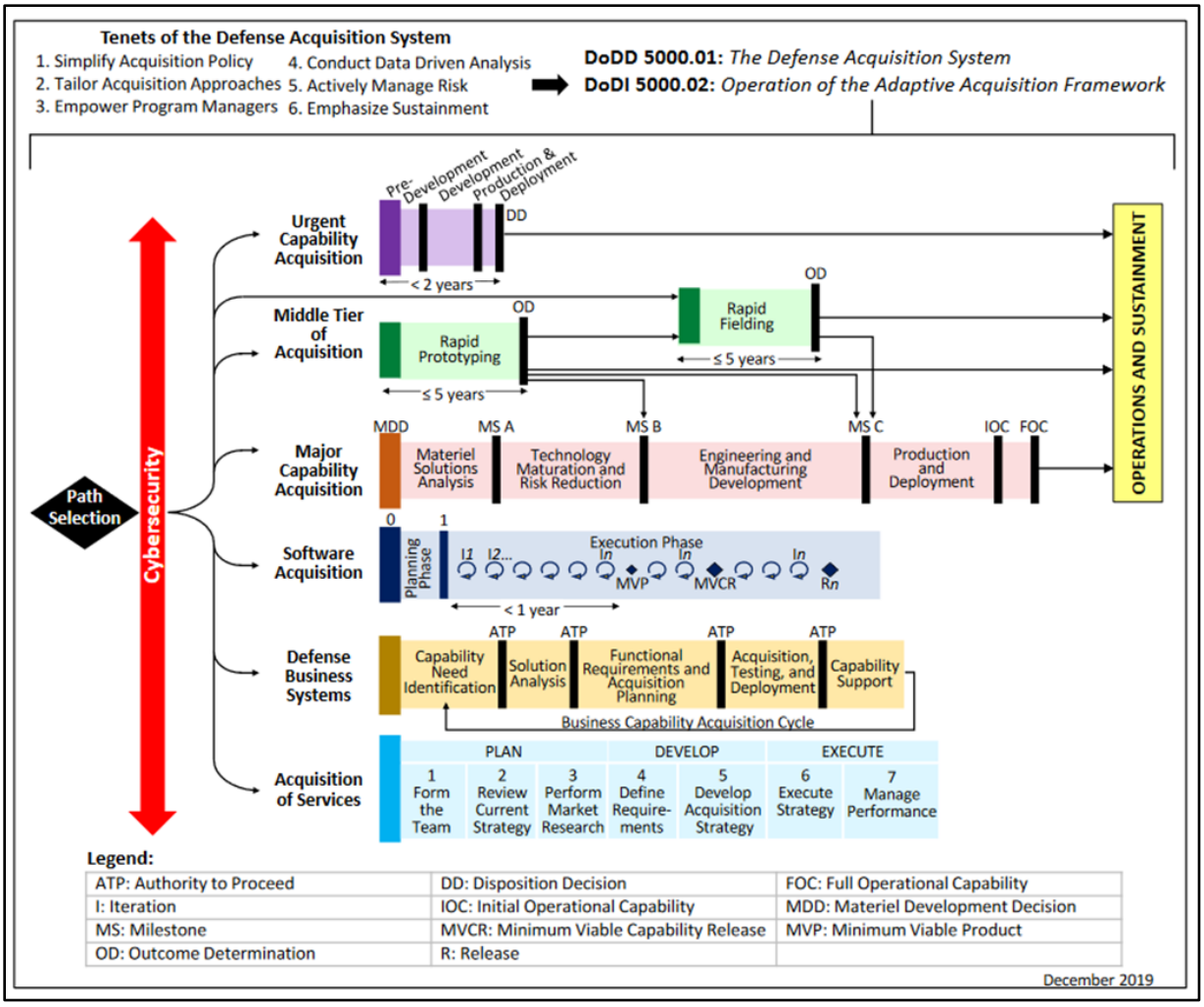
Technology modernization has received much attention in DoD, and several DoD entities have created materials to help support this transition. The materials include publications, online tutorials and videos, training and consulting services, and assistance in setting up automated software development pipelines. Table 2-3 lists relevant sources of policy and guidance.

Table 2-3. DoD Software Engineering and Technology Modernization Policy and Guidance

Title	Office	Year	Policy/Guidance
National Institute of Standards and Technology (NIST) SP 800-207 Zero Trust Architecture	National Institute of Standards and Technology	2020	Article
Mission Engineering Guide	OUSD(R&E)	2020	Guidance
Defense Modeling and Simulation Reference Architecture (DMSRA)	OUSD(R&E)	2020	Guidance
Federal Cloud Computing Strategy	CIO Council	2019	Policy
DevSecOps Academy Video Series (website)	DAU	2019	Guidance
DoD Directive 5000.59 DoD Modeling and Simulation (M&S) Management	OUSD(R&E)	2018	Policy
CJCSI 8510.01C Management of Modeling and Simulation	Chairman of the Joint Chiefs of Staff	2012	Policy
DoD DevSecOps Document Set (website)	DoD CIO	ongoing	Guidance
Joint Federated Assurance Center (website)	OUSD(R&E)	ongoing	Guidance
Modular Open Systems Community of Practice (website)	DAU	ongoing	Guidance

2.4 DoD Instruction 5000.02

DoDI 5000.02, Change 1 (2022) cancelled the 2015 issuance of this instruction that was designated 5000.02T during the transition period to establish a distinction between the two issuances. The issuance presents an approach to tailoring program planning through the six AAF pathways and distinguishes Software Acquisition as one of the pathways (Figure 2-1).



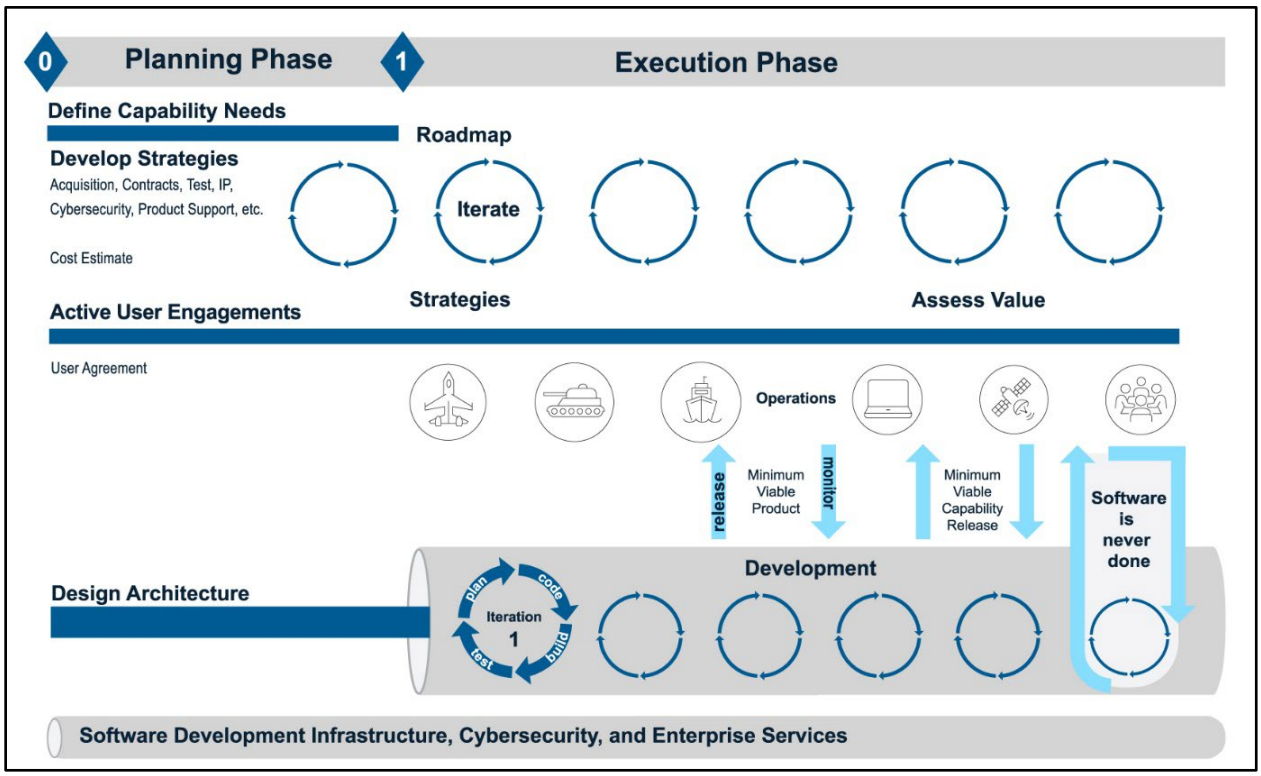
Source: DoDI 5000.02

Figure 2-1. DoD Instruction 5000.02, Adaptive Acquisition Framework (AAF)

Programs may tailor, combine, and transition among acquisition pathways to deliver capability (DoDI 5000.02 2022). The DAU AAF web page "Selecting and Transitioning Pathways" (DAU Selecting Pathways 2022) provides guidance to help programs select a pathway and tailor that pathway to best deliver capability. The Engineering of Defense Systems Guidebook (OUSD(R&E) 2022) also provides guidance for using the pathways.

2.5 Software Acquisition Pathway

The Software Acquisition pathway (Figure 2-2) is designed for software-intensive systems or for software-intensive components or subsystems. The pathway facilitates rapid and continuous delivery of software capability to the warfighter, integrating modern iterative software development practices, such as Agile, Lean, human-centered design, and DevSecOps, to deliver secure and resilient software capability rapidly and iteratively to the end user in the operational environment.



Source: DoDI 5000.87

Figure 2-2. AAF Software Acquisition Pathway

A program selecting the Software Acquisition pathway must commit to delivering a Minimum Viable Capability Release (MVCR*) within one year from the date on which the funds are first obligated.* The pathway recognizes just two phases:

- **Planning Phase.** The planning phase focuses on understanding the users' needs and planning the approach to deliver capabilities to meet those needs.
    - The planning phase is guided by a draft Capability Need Statement (CNS) developed by the operational community, which the sponsor must approve before the execution phase starts.
    - In addition to functional requirements that address user needs, planning addresses non-functional requirements such as security controls (encryption, access controls, Risk Management Framework (RMF) requirements).

- **Execution Phase.** The execution phase focuses on first scoping, developing, and deploying a Minimum Viable Product (MVP) and MVCR to the warfighter/end user as quickly as possible, and iteratively developing and deploying remaining capability thereafter. Ideally, programs will target 3-to-6-month releases, with a preference to the shortest time frame possible.

A program electing the Software Acquisition pathway is required to produce the following artifacts as part of planning and execution:

- Acquisition Strategy
- Capability Needs Statement
- Test Strategy
- User Agreement
- Value Assessment

The DAU AAF Software Acquisition portal (2022) provides links to templates for these documents.

2.6 Human Systems Integration

DoDI 5000.95, Human Systems Integration (HSI) in Defense Acquisition Programs (2022) provides instruction to Program Managers and capability developers about planning, processes, and artifacts required to execute activities to meet HSI requirements. The purpose of HSI is to provide equal consideration of the human element along with the hardware and software processes to engineer a system that optimizes total system performance and minimizes total ownership costs. PMO staff, specifically the Lead Systems Engineer with HSI practitioner support, analyze requirements to optimize total system performance and determine the most effective, efficient, and affordable design. The PMO staff should use the analysis of the HSI domains to help determine and investigate the science and technology gaps to address all aspects of the system (hardware, software, and human).

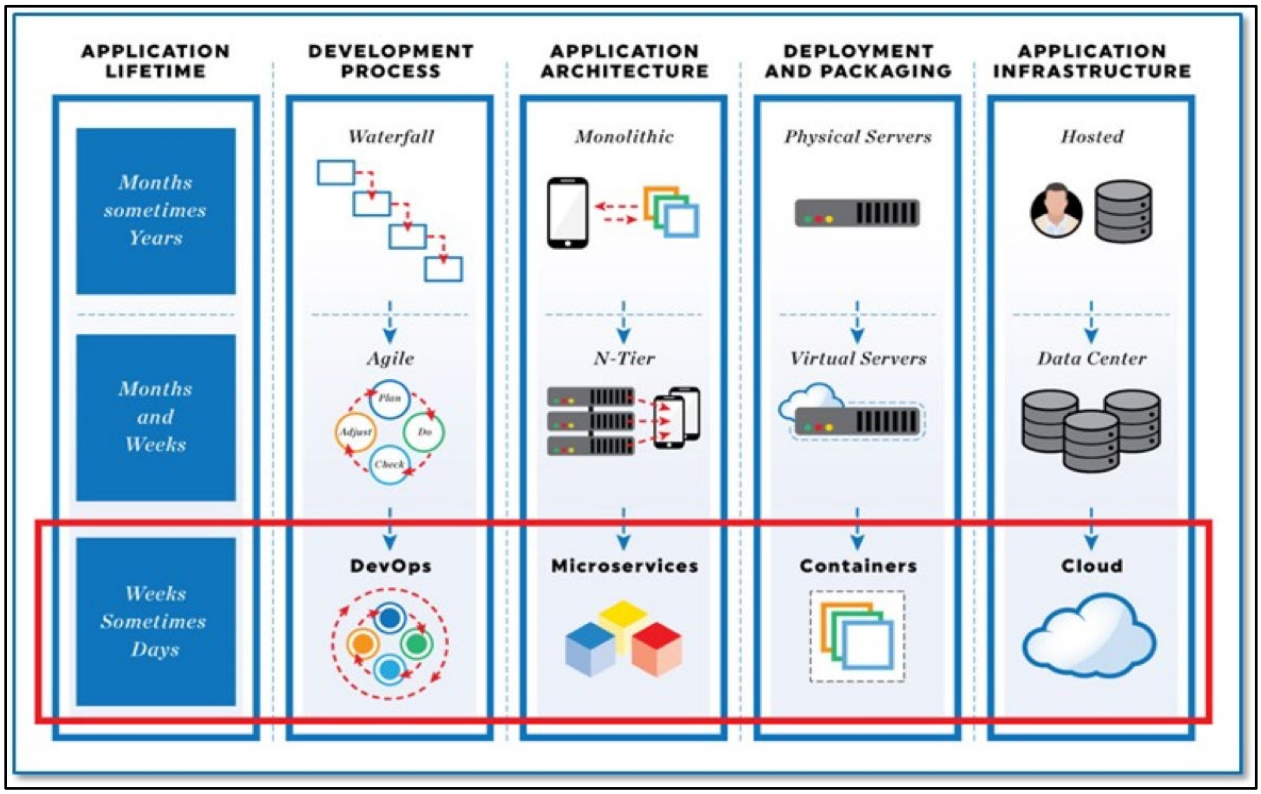
3 Technology Modernization

- Product and technology life cycles are shorter, and the emphasis has shifted from "projects" with a defined start and end to Agile/DevSecOps "products" that are never done and continuously evolve to meet the need of end users.

Recent innovations revolutionize how software is designed, developed, tested, deployed, and operated. Program Managers, software engineers, and other personnel engaged in acquisition and development of software-enabled systems need to be familiar with these concepts. This section discusses significant recent technologies and their implications for how DoD plans, executes, and assesses software-intensive programs.

3.1 Evolution in Software Technology

Software technology is changing in five aspects, as illustrated in Figure 3-1 (Chaillan 2020).



Source: (Chaillan 2020)

Figure 3-1. Five Aspects of Software Technology Changes

Development processes are evolving from traditional document-driven Waterfall processes to the more continuous Agile/DevSecOps processes. DoD Waterfall processes have been structured around "projects" with a defined start and end (typically several years). The processes deliver a

product with a useful life of 5 to 10 years (although sometimes they have been left in place much longer) but replacing or rebuilding the legacy product requires another major project entailing significant effort, cost, and risk.

In newer efforts, the emphasis has shifted from "projects" with a defined start and end date to "products" that are "never done" and continuously evolve to meet the needs of end users. Product and technology life cycles are shorter, and architectures that once were large and monolithic give way to Agile/DevSecOps architectures oriented around many loosely coupled microservices developed independently by small teams.

Rather than relying on manual testing mechanisms at the end of development, testing is automated and occurs on an ongoing basis, at the speed of the development team. Rather than manually installing software on physical servers, developers may bundle applications into pre-integrated containers, which can be dropped onto standard, highly virtualized infrastructure that can adapt to meet the needs and demand of users. Processes for code and infrastructure builds, integration, testing, release, and deployment all can be automated. These innovations enable software engineers to develop and field applications in months or days rather than years.

Agile/DevSecOps offers several advantages:

- Replacing Waterfall with Agile/DevSecOps processes provides the ability to deliver value earlier and to use working product and delivery data to validate future investment (with less risky data-driven micro-investments) rather than relying on the promise of value far into the future validated primarily by documentation (large and long-term investments that are far more risky).

- Replacing Waterfall processes with Agile processes further mitigates risk because requirements are not locked in and are allowed to adapt to changing needs of customers and end users over time. This flexibility coupled with regular releases to the customer or end user provides an opportunity to capture feedback that minimizes investment in unneeded software features and requirements.

- Moving from monolithic to microservice-based architectures helps organizations scale to hundreds or thousands of developers, efficiently, by enabling teams to develop and deploy capability independently.

- Moving from manual deployment and packaging to containers enables a high degree of automation that reduces system administration effort by orders of magnitude.

- Moving from hosted to cloud-based infrastructure helps to contain infrastructure costs by providing increased scalability and flexibility using shared computing and network resources.

Modern software development practices are not necessarily a universal solution to every problem, but in a growing number of contexts modern software development practices offer compelling advantages, reduce risk, and provide sound solutions.

3.2 Software Technologies

Following is a summary of several emerging software engineering technologies applicable to DoD.

Application Programming Interface (API). "A system access point or library function that has a well-defined syntax and is accessible from application programs or user code to provide well-defined functionality" (NIST SP 1800-16 2020). An API lets the software interface with other software. APIs open the door to creating new capability by connecting independently developed systems so they can exchange data, automate repetitive processes, and create and deliver new capabilities rapidly (often in unforeseen ways) in response to changing needs.

Container. "A standard unit of software that packages up code and all its dependencies, down to but not including the operating system (OS). It is a lightweight, stand-alone, executable package of software that includes everything needed to run an application except the OS: code, runtime, system tools, system libraries and settings" (DSOF 2021).

Container Orchestration. A mechanism to scale the deployment, management, networking, and availability of containers through automation. As the number of containers grows, the task of managing them grows more complex. Containers need to be allocated to run, must be started and monitored, and must be restarted when necessary. Container orchestration provides high availability through replication, automation, resilience, load balancing, monitoring, ingress, and more (DAU DevSecOps 2022).

Container Sidecar. A mechanism to monitor what is going on inside a container. The sidecar is ideal for security monitoring and security policy enforcement. The sidecar is itself a container that monitors the communications going into and coming out from another container instance. Because it sits outside the container, a rogue application cannot tamper with the monitoring software (DAU DevSecOps 2022).

The Cloud Native Computer Foundation (CNCF). An organization that defines broadly accepted standards for containers and related technology. Use of CNCF-compliant containers is a constraint found in DoD reference architectures (DoD CIO DSOERDK).

Continuous Integration/Continuous Delivery (CI/CD) Pipeline. An automated software development pipeline that continuously integrates software changes into the system and delivers those changes into the production system (DAU DevSecOps 2022).

Microkernels. A type of kernel that permits the customization of the OS. The microkernel runs in the privileged domain of the hardware and provides low-level address space management as well as interprocess communication. Operating system functions such as the virtual memory manager, file system, and central processing unit (CPU) scheduler may be built on top of the microkernel (see Virtual Machine.) Every service has its own address space to make the services secure, and every application has its own separate address space. This separation protects applications, OS services, and the kernel. Microkernel-based operating systems offer a high level of extensibility, making it possible to customize the operating system's services to meet the needs of the application (Jaiswal n.d.).

Microservices. A distributed systems approach aimed at maximizing scalability by minimizing the human communication needed among development teams. Microservices should be considered when scalability of the development organization is a critical concern, for example, if the organization needs to scale from 20 developers to 2,000 developers. Each microservice is a small, self-contained service designed, built, and maintained by an individual or small team. Microservices are (by definition) independently deployable and independently testable. Those two properties distinguish microservices from traditional service-oriented architectures. Microservices are deployable to production without live testing with the other services in the system, so they must have very stable interfaces. Designers typically associate each microservice with a simple bounded context focused on concepts from the operating (i.e., user) domain. Microservices are not a panacea. The advantages of microservices come at the cost of accepting substantial constraints on the design space. "Microservices is an organizational scaling pattern. That is its advantage. If you don't need to scale up development in your organization, you don't need microservices (although "services" may be a great idea)" (Farley 2022).

Service Mesh. A tool to manage the complex web of interconnections among containers. As the number of containers grows, so does complexity of their communication. As container orchestration automates the management of containers, the service mesh manages the connections among them. Containers do not send messages directly but identify intended recipients through the service mesh, which routes them to their destination. The service mesh can re-route communications to bypass failures, avoid bottlenecks, monitor, and adjust network resources to fluctuations in load and availability. In the absence of a service mesh, the code for managing and rerouting connections would reside in the applications themselves, adding significant complexity (DAU DevSecOps 2022).

Virtual Machine. A technology for virtualizing system resources such as the CPU, memory, devices, and services such as network interfaces and file systems. An OS will not be able to tell that a virtual machine is present between the OS and those resources. The OS will appear to have the machine to itself, as most instructions run directly on the hardware and performance is comparable to that of bare metal. Virtual machines let the user run multiple operating system instances on the same hardware and manages swapping between them transparently. Virtual

machines are a key technology enabling the transition from physical servers to virtual servers in the Deployment and Packaging column of Figure 3-1. Virtual machines also can play a role in embedded software by facilitating deployment, continuous monitoring, secure boot, and cyber defense (Humble and Farley 2011).

Zero Trust (ZT). A "term for an evolving set of cybersecurity paradigms that move defenses from static, network-based perimeters to focus on users, assets, and resources" (NIST SP 800-207 2020). ZT is not so much a technology as an approach to security. This approach stands in contrast to perimeter-based security built around firewalls and enclaves. Under the perimeter-based approach, anyone outside the perimeter was considered untrusted, but once inside they were considered trusted. Under ZT, everyone is suspect whether inside or outside. The system architect assumes the presence of bad actors inside the network and makes architectural decisions accordingly. All interactions between systems are authenticated so both endpoints positively identify one another. All communications are encrypted between systems (NIST SP 800-207 2020).

3.3 Model-Based Systems Engineering

DoD is adopting more model-based systems engineering (MBSE) and model-based software engineering methods to address changes in the information technology (IT) industry. A Practical Software and Systems Management (PSM 2022) working group composed of DoD and defense industry experts observed IT is "undergoing profound changes from traditional engineering requirements, design, development, integration, and verification methods based on documents and artifacts to a future based on digital models and cross-functional digital representations of system designs and end-to-end solutions."

The DoD Digital Engineering Strategy (2018) outlined five goals and provided a foundation for DoD enterprise stakeholders across Government, industry, and academia to use in developing digital transformation initiatives. The defense digital engineering community is now working to implement the five goals:

1. Formalize the development, integration, and use of models to inform enterprise and program decision making.
2. Provide an enduring, authoritative source of truth.
3. Incorporate technological innovation to improve the engineering practice.
4. Establish a supporting infrastructure and environment to perform activities, collaborate, and communicate across stakeholders.
5. Transform the culture and workforce to adopt and support digital engineering across the life cycle.

The PSM digital engineering measurement framework is intended to "help projects and enterprises establish an initial path toward a measurably effective transition and implementation of digital engineering processes, tools, methods and measures" (PSM 2022). The ISO/IEC/IEEE DIS 24641:2021(E) (2021) standard addresses digital and model-based aspects of the engineering process from the initial release of formalized applications of modeling to support systems and software engineering. These two efforts are representative of increasing support for digital engineering in both DoD and industry.

3.4 Technology Modernization Resources

This section has highlighted a few recent technical innovations that have changed DoD software engineering and software development, but DoD is continually employing new technologies. Table 3-1 lists additional resources of information on DoD reference architectures and enterprise resources.

Table 3-1. Technology Modernization Resources

Resource	Description
DoD Software Modernization Strategy (2021) and DevSecOps Document Set https://dodcio.defense.gov/libarary	• Provides the approach for achieving faster delivery of software capabilities in support of Department priorities. • Informs DoD Component execution of DoD Chief Information Officer (CIO) Capability Programming Guidance in support of DoD CIO budget certification for Cloud DevSecOps investments. • Specifies enterprise-wide implementation of innovative acquisition authorities and policies, to include DoD Instruction 5000.87, Operation of the Software Acquisition Pathway . • Promotes increased DoD Component utilization of software factories and secure continuous integration/continuous delivery (CI/CD) pipelines.
DAU DevSecOps Academy Video Series https://media.dau.edu	• Provides capsule introductions to DevSecOps, Containers, Container Orchestration, Infrastructure as Code, Zero Trust (ZT) Model, Chaos Engineering, Telemetry, CI/CD pipelines, and related topics.
Joint Federated Assurance Center https://rt.cto.mil/stpp/syssec/jfac/ https://jfac.navy.mil	• Promotes and enables software assurance and hardware assurance. • Supports program offices by identifying and facilitating access to DoD software assurance and hardware assurance expertise and capabilities to reduce vulnerabilities in fielded DoD systems.

Resource	Description
DoD Zero Trust Reference Architecture https://dodcio.defense.gov/library/	• Documents the Department's approach to cybersecurity. • Shows how ZT supports the 2018 DoD Cyber Strategy, the 2019 DoD Digital Modernization Strategy, the 2021 Executive Order (EO) on Improving the Nation's Cybersecurity, and the DoD CIO vision for: ○ Creating a "more secure, coordinated, seamless, transparent, and cost-effective architecture that transforms data into actionable information and ensures dependable mission execution in the face of a persistent cyber threat." • Specifies that ZT should be used to re-prioritize and integrate existing DoD capabilities and resources, while maintaining availability and minimizing temporal delays in authentication mechanisms, to address the DoD CIO's vision.

Zero Trust Architecture, NIST SP 800-207
 	
https://pages.nist.gov/NIST-Tech-Pubs/SP800.html
	• Provides a canonical set of concepts, definitions, and models for implementing secure systems applying ZT principles and approach.
	  ○ 7 tenets of ZT
	  ○ Logical components of ZT architecture
	  ○ Deployed variations of the ZT architecture
	  ○ Deployment scenarios and use cases
	  ○ Threats associated with ZT
	  ○ Possible interactions with existing Federal guidance
	  ○ Migrating to ZT

Secure Software Development (SSDF) Framework, Version 1.1, NISP SP 800-218
 	
https://pages.nist.gov/NIST-Tech-Pubs/SP800.html
	• Provides recommendations for mitigating risk of software vulnerabilities.
	  ○ Secure software development best practices
	  ○ Build in cybersecurity at onset of software development
	  ○ Open Web Application Security Project (OWASP)
	  ○ Software Assurance Forum for Excellence (SAFE) Code

Test Resource Management Center (TRMC) Repository
 	
https://www.trmc.osd.mil
	• Provides technical guidance on setting up platforms for DevSecOps orchestration and distribution services.
	  ○ Confluence
	  ○ Jira
	  ○ Artifactory
	  ○ Jenkins
	  ○ and others

4 Challenges and Best Practices

- Whereas Waterfall methods manage software development through documents and milestone reviews, Agile/DevSecOps approaches manage development continuously through automated development, testing, and deployment pipelines.
- Testing should guide development, measure progress, and ensure product quality while also moving at the speed of the Agile system.
- A recurring challenge is adapting to change, and a recurring best practice is continuous feedback to advance progress.

This section discusses how to anticipate and overcome challenges associated with software engineering, especially in planning, execution, and assessment. It provides checklists and questions to aid in managing these areas.

Whereas Waterfall methods manage software development through documents and milestone reviews, Agile/DevSecOps approaches manage development continuously via automated development, testing, and deployment pipelines (DIB 2019b). This section discusses software engineering challenges and best practices in terms of the following development stages of the Software Development Life Cycle (SDLC): Requirements, Architecture, Design, Coding, Testing, System Integration and Test, and Operations.

As software development moves through the SDLC, some activities may be in Requirements while others are in Coding and others in System Integration and Test. Programs should seek to automate as much of the SDLC as possible.

Table 4-1 summarizes several challenges and best practices associated with each area. The succeeding sections provide further discussion and recommendations to consider.

Table 4-1. Challenges and Best Practices by Development Stage

Challenge	Best Practices
Requirements (Section 4.1)	
• Understanding what is needed. • Understanding how a potential capability addresses that need. • Adapting as needs change in unpredictable ways. • Incorporating technical innovations. • Managing volatile requirements. • Ensuring efficient human system interactions. • Tracing requirements from high-level Capability Needs Statement to user stories.	• Employ rapid prototyping and early engagement with the stakeholders, primarily focusing on customers/end users. • Structure work to deliver stand-alone value along pre-determined timeboxes (e.g., release cycles, sprints) without disrupting those cadences. • Refine requirements based on iterative feedback, usability assessments, and human capabilities and limitations. • Carefully manage scope by managing requirements backlog with continuous planning and prioritization. • Deliver Minimum Viable Product to users as soon as possible (e.g., 3-6 months or sooner). • Account for Human Computer Interaction in requirements and telemetry planning. • Plan human systems integration activities by tailoring SAE6906. • Ensure early involvement by testers to provide input to inform test-driven development, assess testability of requirements, automate testing to the greatest extent possible, and use automated tools for traceability.
Software Architecture (Section 4.2)	
• Adapting legacy architectures and systems constrained evolution (closed and proprietary architectures). • Finding sufficient software expertise in new paradigms. • Understanding Cloud services. • Setting up continuous integration/continuous delivery (CI/CD) pipelines. • Transitioning to a Zero Trust architecture model. • Speeding up product deployment. • Embedded systems, cyber-contested environments, disconnected operation. • Scaling size of development organization to hundreds or thousands of developers. • Determining desired modularity and open architecture. Consider Modular Open Systems Approach design when practicable.	• Follow DoD Enterprise and DevSecOps model. • Employ vetted standard CI/CD pipelines. • Employ microservice architectures in the context of distributed systems when organization needs to scale personnel. • Use vetted containers with a security monitoring sidecar. • Employ cyber resiliency countermeasures such as continuous monitoring, sandboxing, authentication services, opportunistic connectivity. • Follow strangler pattern with legacy application modernization applications. Incrementally replace legacy code with new services employing a facade in front of the refactored software to prevent disruption to dependent external systems.

Challenge	Best Practices
Design (Section 4.3)	
• Adapting to changing technology trends. • Integrating with future warfighting systems in unforeseen ways. • Ensuring software can be tested efficiently. • Designing Application Programming Interfaces (APIs) that are both easy to use and easy to test. • Capturing experience in the field to understand operational needs. • Avoiding design mistakes that open the door to cyber-attack. • Ensuring suitable and effective human system interaction. • Maintaining control over technical debt.	• Identify potential high-value touch points and define APIs to enable semantic interoperability. • Identify interfaces to external systems and work with external system POCs. • Employ Test-Driven Development (TDD) by developing executable specifications as test cases prior to implementing APIs. • Require a Software Bill of Materials. • Modularize early Minimum Viable Product/Minimum Viable Capability Release delivery. • Design, plan for evolution, iterate over time. • Design Interfaces and instrumentation to aid automated testing and gathering of telemetry data. • Conduct table-top cyber exercises for early design feedback. • Incorporate human-centered design such as described by ISO 9241-210. • Design well enough to avoid technical debt and future rework. • Simulate the operational environment in software.

Challenge	Best Practices
Coding (Section 4.4)	
• Detecting coding errors early before they lead to vulnerabilities. • Obtaining high productivity without sacrificing security, quality, and reliability. • Choosing the right programming language and tool stack for development. • Guarding against supply chain attacks. • Recording software design decisions so can be automatically verified for consistency and completeness. • Communicating assumptions clearly to programmers who will need to understand the code over its lifetime. • Creating inline and technical design documentation to support cross-training and supportability.	• Ensure the team is dedicated (fully allocated). • Ensure the team is not disrupted within sprints • Plan for at least one development team to become the product team (software is never done, eliminate the shift to operations and maintenance). • Adopt secure coding standards and train programmers to apply them. • Code APIs to the executable specifications developed in the TDD process. • Enforce coding standards through review aided by static and dynamic analysis tools in the CI/CD pipeline. • Use the most type-safe, abstract language feasible (aids automated checking and human understanding). • Employ provenance scanning tools. • Retain code in a Government-controlled repository. • Employ ubiquitous automated unit testing during coding. • Release code frequently through CI/CD to testing, integration, and production • Ensure Definition of Done includes building the code necessary to automate testing and validate all acceptance criteria.

Challenge	Best Practices
Testing (Section 4.5)	
• Moving at the speed of the development team to ensure testing does not inhibit value delivery • Cultural challenges that drive manual testing requirements • Measuring delivery of new capability in the software base. • Measuring the quality of regression tests. • Preventing errors introduced as new capabilities are added. • Testing for safety, reliability, security. • Confidence in the coverage of the test suite. How much is enough? • Finding computing resources and time to execute the tests. • Availability of hardware, personnel, and equipment. • Transparency of testing to support Government test roles, including evaluation of effectiveness, suitability, survivability, and Responsible Artificial Intelligence. • Perform Workload Analysis to determine potential impacts to safe and effective performance as capabilities are introduced. • Availability of user personnel to support mission-oriented developmental test. Best Practice: Identify user requirement in User Agreement or similar documents.	• Map all test procedures to requirements. • Ensure all stories have testable acceptance criteria. • Employ code coverage metrics to assess test quality. • Build automated regression and other tests into the CI/CD pipeline. • Employ the executable specifications developed in the TDD process as automated regression tests. • Employ scalable Cloud resources to execute many tests in parallel. • Provide a range of simulation and test environments to augment testing on hardware with lower cost alternatives. • Synchronize development priorities with hardware availability. • Use Cybersecurity T&E Guidebook to develop security testing (addresses testing for security challenge). • Use Scientific Test and Analysis Techniques (STAT) to measure test coverage.
System Integration and Test (Section 4.6)	
• Integrating components and subsystems produced by different development teams. • Complexity of interfaces and potential for misunderstanding, ambiguity, and miscommunication. • Time to deal with unforeseen interactions that surface when combining components into a system for the first time. • Minimizing rework that discovery of integration issues may entail. • Potential to exceed specified human limits (e.g., workload, situational awareness, etc.) leading to increased potential for human errors.	• Shift integration left as much as possible. • Plan and design for integration to be performed incrementally as capabilities evolve. • Integrate to mature, standardized APIs where feasible. • Integrate to stubbed and simulated component interfaces in advance. • Ensure components developed by different teams are loosely coupled (i.e., interact through simple interfaces characterized by a small number of well-understood, clearly documented assumptions).

Challenge	Best Practices
Operations (Section 4.7)	
• Deployment of the software into the operational hardware and environment. • Training of the operational staff in the use monitoring and administration of the deployed system. • Managing the evolving security threats present in the operating environment. • Assessing the availability, reliability, and usability of the software under operating conditions. • Usability under denied, disrupted, intermittent, limited (D-DIL) connectivity. • Ensuring that new capabilities align with operational needs in a timely manner.	• Deploy software through Cloud-based services • Customer and end user engagement in sprints (design and demonstrations) • Provide continuous feedback to the development team. • Collect and transmit telemetry to inform development. • Employ a continuous Authorization to Operate (cATO) Layered Approach; swappable approved layers; continuous monitoring. • Zero Trust approach, authenticate everything, encrypt all communication.

4.1 Requirements Best Practices

Requirements should be unambiguous, testable, consistent, and precise so the program will be able to demonstrate whether a system meets the requirements. In an Agile/DevSecOps environment, requirements are typically expressed as capabilities, epics, features, and stories.

Modern practice recognizes there is a point of diminishing returns when refining requirements in advance of system development. This guide advocates initially developing a core set of requirements that define the MVP or MVCR, then routinely structuring time-boxed and cadenced release cycles to continuously develop the product. The MVP/MVCR reflects the core set of mandatory features the software must have to deliver value to operational users so they can give feedback to help evolve the product. Features of the MVP/MVCR must be balanced against the complexity or effort to be exerted by the team (i.e., the product must be both minimum and viable). Feedback from user experience with the software guides future product direction, informs adaptation, and helps refine requirements to maximize the value of the product. Once the MVP is defined, the program can begin to deliver working software to the users.

Following are emerging best practices for developing software requirements applicable to DoD:

- Participate in the development of the Operational Sponsor's CNS or the Software Initial Capabilities Document (SW-ICD) to get an early understanding of the user's needs.

- Derive requirements from the CNS or SW-ICD describing the capability and benefit of having the capability, written from the stakeholder, and in particular the end user's, perspective.

- Communicate with stakeholders (primarily customers and end users), building user stories to capture user input on problems the system is expected to solve and the value provided by solving those problems.

- Engage developmental test and evaluation (DT&E) and operational test and evaluation (OT&E) organizations early in the process of developing requirements, and in all subsequent phases of development.

- Prepare a Product Vision that defines the long-term vision for the system and the goals and objectives to achieve.

- Develop a high-level, adaptable Product Roadmap that illustrates how the product team(s) will deliver on that vision.
    - The Product Roadmap typically covers a 1-3 year period and focuses on capability delivery. Epics and features may be defined for the MVP or upcoming release only.
    - Stories, tasks, and activities should not be included in a Product Roadmap (this is not a project schedule or an Integrated Master Schedule).
- Map capabilities, epics, and features on the roadmap back to mission needs.

- Employ automated tools (e.g., Jira, AzureDevOps, ServiceNow) to automate the Roadmap and Product Backlog, and to maintain traceability among the capabilities, epics, features, and stories.

- Hold a 1-2 day program increment or release planning event with leadership, user representatives and product teams to break down the work in the Product Roadmap targeted for the MVP or upcoming release into epics, features, and/or stories that will comprise the Product Backlog. This planning event helps ensure the product teams understand and communicate the needs and requirements.

- Involve users from the beginning while defining the Product Roadmap and Product Backlog; consider user experience from the beginning, especially in new product or new capability development.

- Create a clear, concise, traceable, and effective CNS or SW-ICD, along with a Product Roadmap and Product Backlog.

- Establish a product owner responsible for being the “voice of the customer” with the authority to prioritize specific capabilities, epics, and features (requirements) targeted needed for an MVP or MVCR (in accordance with DoDI 5000.87 (2020)).

- Do not wait for full agreement on all requirements before proceeding; begin with a stable subset of requirements that together provide a useful core capability.

- Create a prototype and conduct demonstrations to confirm or refine agreed-upon requirements.
- Have developers interact directly with users to better understand their needs to guide the many micro-decisions that developers make in refining the requirements into design and code.
- Capture requirements as test cases incorporated into automated test suites as part of a test-driven development approach.
- Address functional requirements (functions the system performs for the user) and non-functional requirements (security, reliability, maintainability).
- Track and prioritize requirements (Product Backlog) using automated tools that integrate with the development pipeline to provide traceability and prioritization and to aid testing.
- Be prepared to adjust requirements and roadmaps as the program gains new insights regarding feasibility, cost, and value to users.
- Allow early (and continuous) access to features on the operational system so users can provide valuable feedback to modify and re-prioritize the requirements in the Product Roadmap and Backlog.
- Understand and begin implementation of the Risk Management Framework (RMF) workflow. NIST SP 800-53 (2020) defines the RMF as the process for identifying, implementing, assessing, and managing cybersecurity capabilities and services, expressed as security controls, and authorizing the operation of information systems (IS) and platform information technology (PIT) systems. The RMF brings a risk-based approach to the implementation of cybersecurity, supports cybersecurity integration early and throughout the system life cycle, promotes reciprocity to the maximum extent possible, and stresses continuous monitoring.
- Understand the Authorization to Operate (ATO) workflow. The Committee on National Security Systems (CNSS) (CNSSI 4009 2015) defines an ATO as the official management decision issued by a designated accrediting authority or principal accrediting authority to authorize operation of an information system and to explicitly accept the residual risk to agency operations (including mission, functions, image, or reputation), agency assets, or individuals.

4.2 Software Architecture Best Practices

It is difficult to make blanket statements about architecture that apply to all types of software. The Defense Innovation Board study (DIB 2019b) divided software into three broad operational categories: enterprise systems, business systems, and combat systems. It further subdivided combat systems into logistics systems, mission systems, and weapon systems.

The report also identified several types of computing platforms on which these operational functions might be implemented: Cloud computing, client/server computing, desktop/laptop computing, mobile computing, and embedded computing. Each of these platforms may be employed alone or in combination as part of any of the operational categories above.

Following are some best practices for tailoring architectural decisions to program needs:

- Use digital models in architecture development to illustrate different models and views. Models may illustrate software edges, interactions, and operations. Ensure each software architecture can be implemented in a physical architecture.
- See vetted DoD Enterprise DevSecOps software factories and platforms. Information on vetted software factories and platforms is available at (DoD Enterprise DevSecOps Portal 2023)(Common Access Card required). As of early 2023 this site maintains a spreadsheet describing over fifty DevSecOps platforms and software factories listing attributes such as Name, Location, Software Factory and Platform Mission, and Point of Contact.
- Adopt a Modular Open Systems Approach (MOSA) employing a design strategy that consists of a technical architecture that uses system interfaces compliant with widely supported and consensus-based standards (if available and suitable) and supports modules that are highly cohesive internally but loosely coupled to one another at the system level.
- A program building a distributed system that requires maximum scalability should consider using microservices. By definition, microservices are independently deployable and testable without reliance on testing with any other microservices (see also Section 3, Technology Modernization).
- Consider using enterprise Cloud infrastructure with templates for integrated toolchain to accelerate setting up automated development pipelines.
- Take advantage of software container technology to automate deployment of software and minimize manual system administration effort.
- Reuse architectural components such as a service mesh and container sidecar when developing container-based solutions.

- Consider that modern infrastructure may be relevant even for systems that are not a web service. For example, embedded systems might take advantage of Cloud-based infrastructure for simulation and testing.
- Make specialized hardware available remotely as a sharable resource incorporated into dynamically configured Cloud-based test and development platforms.
- Use an architecture that divides the work among teams who can work independently. An ideal separation minimizes the need for communication among teams, which speeds development and leads to more robust systems.
    - Stable interfaces and simple well-documented APIs are characteristic of architectures that minimize communication needs.
    - Unstable interfaces with complex and poorly documented APIs create confusion, and resolving the issues requires a great amount of communication among teams.
- Consider Software as a Service (SaaS) offerings to replace the functions of legacy systems that are in sunset or no longer supported.
- Upgrade the security architecture to account for modern embedded software that has evolved from simple embedded controllers to complex network entities with operating systems, network stacks, and programmability – and the vulnerabilities and exposures that come with that added complexity.
- Apply a ZT approach to architecture and design in accordance with NIST 800-207.
- Use the ecosystem of high-quality open source software (OSS) to save time and development effort. See answers to frequently asked questions in the "DoD Open Source Software FAQ" (DoD CIO 2021).
- Maintain awareness of supply chain risks and adopt countermeasures to mitigate those risks to acceptable levels.

Software architecture lays the foundation for system and product development; therefore, the role of architect is a key one. To make correct architectural decisions and execute software development successfully, the architect must be able to staff the effort with qualified people in key positions in a timely manner. The organization will be able to recruit, hire, and establish clearances more successfully if it can attract the best talent and streamline hiring and onboarding processes.

4.3 Design Best Practices

Where software architecture activity deals with overarching decisions that apply to the system as a whole, design activity addresses further refinement of the system components and relationships among them. Following are examples of design best practices:

- Maintain awareness of and control over dependencies among software components or modules. Ensure module interfaces are available to the Government.

- Make explicit in the design which modules require the presence of other modules in order to provide a function. Awareness of these dependencies allows the software engineer to infer (1) constraints on the order in which capabilities can be delivered, (2) usable subsets of the system that may be composed from those modules, and (3) what modules can be reused independently in other contexts.

- Require a Software Bill of Materials (SBOM), a formal record containing the details and supply chain relationships of various components used in building software. Executive Order (EO) 14028, "Improving the Nation's Cybersecurity" (2021), defines an SBOM and identifies the value proposition in paragraph 10(j). In addition, paragraph 4(f) defines the minimum elements of the SBOM and paragraph 4(e)(vii) provides guidance on providing a purchaser SBOM for each product. The National Telecommunications and Information Administration (NTIA), Department of Commerce, provides a comprehensive collection of materials on the SBOM on its website (https://ntia.gov/page/software-bill-materials) (NTIA SBOM 2021).

- Employ automated tools using formalisms such as the Unified Modeling Language (UML) or System Modeling Language (SysML) to capture, analyze, and communicate how software modules interact.

The design points stated above concern the relationships between modules at a high level. A finer level of detail would include specifications for all the individual API calls. Such specifications can easily range into the hundreds, even thousands, of elements considering the specification of a function's arguments, return values, and all the potential errors and exceptions that may be raised and under what conditions.

4.4 Coding Best Practices

The chief challenge of the coding phase is to produce working code within constraints of the architecture, software design, and user requirements in a timely manner.

Designs evolve in response to what developers learn by refining them into code. They learn even more when the code is released into production. This feedback and iteration among design, coding, and production is inevitable. Modern approaches recognize this reality and help software developers balance the upfront effort devoted to design specifications with the need to generate working code and to get feedback.

Following are examples of coding best practices:

- Proceed with coding after availability of a *sufficiently* detailed design. Do not wait for a *fully* detailed design. Further develop and refine design artifacts as coding and learning proceed. A sufficiently detailed design is defined as one that supports: (1) the design, code, test, and delivery of an MVP or MVCR to proceed collaboratively without confusion and conflict; (2) a shared understanding of responsibilities and authorities for further refinement to the design; and (3) a clarity regarding which decisions are fixed and which are flexible.

- Generate API specifications for specific software components and sub-components as part of the coding and design process. Many (if not all) modern languages have corresponding utilities for extracting API documentation from comments embedded in the code. These utilities generate API documentation as hyperlinked, cross-referenced, fully searchable websites; e.g., Javadoc, Doxygen, Hdoc, mkdocs, jsDoc, YARD, sandcastle.

- Annotate code with concise natural language comments that can be extracted by automated tools to generate web pages, diagrams, and searchable databases documenting those interfaces. Provide inline documentation for program logic, variable names, etc., to make the software easier to maintain.

- Have the development team review the generated documentation, especially those who rely on those program interfaces to build their own components.

- Employ digital models and automated tools to update interface documentation, maintaining constancy as the design and code evolve.

Coding is also a critical touchpoint for building in security assurances by adopting standards, training programmers, and employing automated analysis tools.

- Adopt secure coding standards and provide a mechanism to enforce them. Intellipedia (Intellipedia Secure Coding Guidelines 2022) has compiled references for secure coding standards (DoD CAC Required). A more broadly accessible set of language specific coding standards is available from SEI (SEI CERT Coding Standards 2020).

- Train programmers in secure coding practices.

- Employ static analysis tools and dynamic analysis tools into the CI/CD pipeline as a means of enforcing secure coding standards and training programmers to write more secure code.

Finally, developers and infrastructure support groups should configure and use automated tools to do version control, gain immediate feedback from testing and quality gates, and perform rollbacks quickly and efficiently when needed.

- Conduct integrated testing (contractor and Government development test (DT) operational test (OT)) at the speed of the development team (as early as possible and in an ongoing rhythm) to collect data and avoid large tests at the end of development.
- Retain code in a U.S. Government-managed repository system.
- Avoid branching the development tree for prolonged periods as this compounds integration costs. Strive to merge changes into the main development trunk daily.
- Ensure the code repository commit triggers automated security scans and regression tests, and provides timely feedback to the developer on the results.

4.5 Code Development and Test Best Practices

Director, Operational Test and Evaluation (DOT&E) and OUSD(R&E) Developmental Test, Evaluation, and Assessments (DTE&A) are responsible for test and evaluation (T&E) policy and guidance and are in the process of publishing the following T&E guidance documents:

- DoDI 5000.89 Test and Evaluation, Section 4.5 T&E for Software Acquisition Pathway
- T&E Enterprise Guidebook, Chapter 5 Software Acquisition
- T&E Enterprise Guidebook – Software T&E DoD Manual (forthcoming)
- T&E Enterprise Guidebook – Cyber DoD Manual (forthcoming)
- Software T&E Companion Guide – Agile/DevSecOps (forthcoming)
- Cyber T&E Companion Guide (forthcoming)

These documents detail additional expectations with respect to DTE&A and DOT&E organizations.

The following paragraphs discuss incremental testing by the software development team, supported by the software test team. Software engineers should work with software testers and refer to the above documents for additional testing guidance.

Testing should guide development, measure progress, and ensure product quality while also moving at the speed of the Agile system. Following are examples of code development and test best practices:

- Capture testing requirements upfront and build them into Acceptance Criteria (which applies to a specific story) and Definition of Done (which applies to all stories).
- Invite independent Government testers to participate or observe integration testing for early evaluation, as appropriate.
- Begin testing activities early and run them continuously, ideally with the help of DT and OT stakeholders so they can help shape these practices to be maximally effective and avoid downstream effort and calendar time as much as possible.
- Measure the growth of a new capability in the software base in terms of prioritized features traceable to the requirements agreed upon for the system.
- Use Test-Driven Development (TDD) and Behavior-Driven Development (BDD). TDD and BDD are two recognized Agile techniques teams can use to write executable specifications as test cases or automated test script prior to coding, and that developers can use to write the code to satisfy the test cases.
- Employ automated testing for both functional testing (linked to the user stories and functional requirements) and for non-functional testing (linked to cybersecurity software assurance, safety, and resilience).
- Curate and maintain a suite of regression tests, which evolve over the lifetime of the system. Regression tests enable incremental development by ensuring that developers do not introduce new errors as they add new capabilities to the code.
- Testing should cover both:
    - Functional Testing. Testing of those features that correspond to the capabilities the users want, and
    - Non-functional Testing. Testing aimed at measuring global characteristics: safety, security, reliability, maintainability, performance, and resilience.
- Manage, track, and prioritize software defects until they are resolved, tested, and released.
- Integrate automated software analysis tools into the CI/CD pipeline to assist testing for safety, reliability, and security.

- Collect quantitative metrics to assess the test suite quality and completeness of coverage using Scientific Test and Analysis Techniques (STAT).

- Use the instrumentation available (e.g., code coverage tools, test case to requirement tracking) in modern development environments to collect metrics to assess the quality of test suites.

- Employ a range of test platforms to provide the flexibility to trade off fidelity for lower cost, faster cycle times, and greater scale. For example, in avionics systems the highest fidelity/most expensive platform is flight testing on the physical aircraft. Less fidelity, but at lower cost, would be testing against a software simulation of the aircraft's physical responses and simulated interfaces to the controls and sensors on the aircraft. Having a range of testing platforms helps the program optimize the overall testing effort.

- Programs should seek to automate testing to the greatest extent possible. This will require developers to build automated tests during coding, but it also will require test engineers who are well trained on automated testing practices; including how to interpret results of automated testing, how to identify automated testing gaps, and avoiding manual testing that duplicates automated test results/outcomes.

- Analyze test data and generate metrics to inform decisions that keep the program on track. Take corrective action in response to undesired trends and variance from control thresholds.

- Synchronize development and test priorities with hardware availability. In many DoD projects, software development testing will depend on the timely delivery of hardware that is still in the process of design and development. Consider the risks inherent in timing of hardware availability and consider mitigations to minimize impact on software development and testing.

4.6 System Integration and Test Best Practices

This section focuses on integration and component testing by the software development team, supported by the key stakeholders/users. Integration testing also takes place at higher levels of subsystem, system, and system of systems, which the software engineers should track. Testing at these levels may be performed by independent Government testers.

System integration and testing should happen continuously through the SDLC as software is made available. See the DOT&E and OUSD(R&E) DTE&A T&E policy and guidance documents outlined in section 4.5 for additional testing guidance. Following is a partial list of system integration and test best practices:

- Integrate new components into the system in small increments as they become available.
- Integrate to a surrogate simulation or stub out the component if a dependency is unavailable.
- Prototype integration of new components to identify issues and resolve them before they become critical.
- Maintain open channels of communication between development teams working in different locations.
- Avoid trying to integrate too many components at once, especially on compressed schedules.
- Take advantage of regression testing at the functional level to shake out integration issues. Functional level testing focuses on the end user interaction and interactions with external systems.

4.7 Operations Best Practices

4.7.1 Release on Demand

Once a solution for a capability has been developed and has passed its testing, validation, and applicable certification gates, it can be made available to the user; however, these new releases should include technical measures to mitigate risks from potential defects since the project may have minimal experience running the new release in the user's context.

- Dark Launches provide the ability to deploy to a production environment without releasing the functionality to the users.
- Feature Toggles facilitate Dark Launches by implementing toggles in the code that enable switching between old and new functionality.
- Canary Releases provide a mechanism for releasing the solution to a specific user segment to enable measurement of results before expanding and releasing to more users.
- Decoupled Release Elements allow releases of some capabilities to be delivered on a different schedule or frequency from others, depending on need. For example, releases that fix security flaws may be quite urgent, while routine changes resulting from evolving office functions may be released less frequently.

It is important for new Agile/DevSecOps teams to release in short, routine cadences that allow them to practice and improve the flow of value delivery. It is also important that operations do

not unnecessarily inhibit the delivery of value or delay the learning/feedback loops necessary to guide future development effort. Therefore, it is critical that the program not unintentionally translate "release on demand" to releasing over long-term (Waterfall) release cycles. If concerns or challenges inhibit providing new value to users, the team should identify and eliminate them to the greatest extent possible. Independent release cycles allow developers to achieve separate flows or "value streamlets," delivering at their own needs and pace. Identifying streamlets is a critical aspect of release on demand.

4.7.2 Automated Deployment

The operational environment is where the program is first able to validate that the delivered product provides valuable capability by fulfilling the operational need of the users. Those planning and assessing deployment into operations should consider:

- Automating deployment of the software and systems into the operational hardware and environment.
- Making canary releases (see 4.7.1) of new capability that can be rolled back easily if flawed or unsatisfactory.
- Training the operational staff in the use of software tools to handle monitoring and administration of the deployed system; including the software install and rollback.
- Giving attention to managing the evolving security threats present in the operating environment. These may differ from threats present in the development environment.
- Collecting metrics to help assess the availability, reliability, survivability/resiliency, and usability of the system under operating conditions.
- Assessing usability under disconnected and intermittent connectivity situations that can occur in operations.
- Deployment to pre-production or staging environment to support Government DT and early OT.

4.8 Agile Development Maturity

Organizations vary widely in their ability to execute an Agile/DevSecOps approach to software development. Those that have developed disciplined processes with a high degree of test automation, quality checks, and trained personnel, and that have refined a continuous delivery pipeline over several years on multiple projects, will have a far easier time than organizations just starting this transition.

Rather than developing a maturity model, DoD CIO recommends developing a set of capabilities. They have prepared a playbook that elaborates on specific capabilities needed to put in place effective Agile/DevSecOps processes (DOD DSO Playbook 2021).

4.9 Summary

DoD established traditional software development practices to deliver systems that meet warfighter needs and are safe, secure, and reliable. The Department's evolution to more Agile, continuous, and iterative processes helps achieve those benefits and maximize the value delivered by readily adapting to new learning, evolving needs, and adversary actions. The development team should manage requirements, architecture decisions, design coding, documentation, testing, and deployment through systematic processes that they refine continually to achieve speed of delivery and high quality of result.

5 Software Metrics Use and Lessons Learned

- Programs that adequately track and monitor software development with metrics have been more successful at lowering program risk.
- Objective metrics provide quantitative information teams can use to identify and understand trends, promoting continuous improvement.
- Without objective metrics, assessing an effort or development may be open to biased interpretation, which could make the assessment inaccurate.
- Programs should establish consistent metrics for the life of the project.

Metrics can yield useful insight regarding program health and can prompt decision makers at various levels to take corrective action as needed. This section discusses how to use metrics to guide decision making by grounding those decisions in objective data.

This section discusses metrics as they apply to the Software Acquisition pathway outlined in DoDI 5000.87, Operation of the Software Acquisition Pathway (2020); Agile/DevSecOps processes; and software factories. It suggests metrics to inform decision makers at the team, product, and enterprise levels. It includes questions to consider when selecting metrics and presents case studies from OUSD(R&E) program assessments to illustrate the use of metrics.

5.1 Distinction between Waterfall and Agile/DevSecOps Metrics

In traditional Waterfall approaches, requirements are fixed and used to estimate time and cost. Waterfall emphasizes metrics that validate these estimates.

Agile/DevSecOps approaches attempt to fix time and cost rather than requirements. Software teams fix the cost and releases or iterations along "timeboxes" (windows of time) and then estimate the amount of value (requirements or stories) they can deliver within those timeboxes. Agile/DevSecOps metrics focus on team performance, the efficiency of the team's value delivery, and quality. Therefore, the Agile/DevSecOps approach results in a different set of metrics from Waterfall because the metrics are used for a different purpose. The metrics must support decision makers and help the team to improve.

OUSD(R&E) recommends Agile/DevSecOps metrics using as much automation as possible. The DAU website on Metrics and Reporting provides DoD guidance including the following:

> Each program will develop and track a set of metrics to assess and manage the performance, progress, speed, cybersecurity, and quality of the software development, its development teams, and ability to meet users' needs. Metrics

> collection will leverage automated tools to the maximum extent practicable. The program will continue to update its cost estimates and cost and software data reporting from the planning phase throughout the execution phase. (DAU Metrics and Reporting 2022)

5.2 Metrics Inform Decisions

Metrics should inform decisions on many levels and throughout the life of the project:

> The goal of generating metrics is to provide leadership, the Product Owner, team members, and other key stakeholders information and insights into the development effort to guide technical/programmatic decision-making, continuous improvement efforts, and remediation of blockers/impediments. Software teams should regularly review metrics as part of their sprint/release retrospectives and leverage metrics both for continuous improvement and to plan future iterations. (DAU Metrics and Reporting 2022)

In its *Agile Assessment Guide: Best Practices for Agile Adoption and Implementation*, the Government Accountability Office (GAO) emphasized the importance of metrics for routine management as well as enterprise-level decisions:

> Project and technical managers need objective information to make day-to-day decisions, identify project issues, correct existing problems, and manage prospective risks. This same information must also provide a basis for evaluating organizational and enterprise-level performance and assessing the impact of policy and investment decisions. (GAO 2020)

5.2.1 Decision Makers

Decisions occur at the team, product, and enterprise level. Each level has its own role, accountability, and authority and is responsible for a range of decisions. The varying scope requires varying metrics that result in the appropriate data to inform each type of decision.

- Team (Technical Focus). The Chief Engineer, system architect, software engineers, and supporting roles (e.g., information technology, configuration management, test) work with the software environment daily to make many detailed low-level decisions at a rapid pace. They coordinate among themselves to allocate design, development, and testing tasks with an immediate impact measured in hours and days.
- Product (Tactical Focus). The product owner acts as the voice of the customer to prioritize work for the team and guide product direction. The product owner owns the Product Vision, Roadmap, and Backlog. The product owner ensures that the requirements reflect the needs and priorities of the user community and align to the mission objectives.

The Program Manager manages contracts, provides servant leadership to ensure the appropriate Key System Attributes (KSAs) and tools for the team, and tracks execution and delivery of product.

- Enterprise (Strategic Focus). Executive leaders (e.g., Milestone Decision Authority (MDA) and Program Sponsor) and those with program oversight functions must track the performance either individually or in the larger context of a portfolio of programs. They make investment decisions that direct large blocks of resources over single- or multi-year horizons. They require accurate technical assessments of program risk and deployment value. They act as Servant Leaders that help the team remove blockers that inhibit success in a timely manner.

5.2.2 Performance Insight

Agile/DevSecOps teams routinely use performance data to guide continuous improvement efforts. Since most teams generate working product (value) routinely over short release cycles (e.g., 3-6 months), they generate data to assess performance. These objective metrics provide insight into the current state of a program's performance and can be extrapolated to future performance (e.g., burndown charts).

For example, metrics data collected over time can show how efficiently the team is building (cycle time) and how fast they are delivering value to the customer (lead time for change).

Claim. Programs that track and continuously monitor software metrics are typically the most successful in delivering timely value to the warfighter.

5.2.3 Technical Debt

Technical debt, an important metric in the execution and life cycle of a program, is additional work that needs to be completed arising from decisions made during development and sustainment. Technical debt may accumulate because of a combination of architecture definition/modification, software design definition/modification, and/or a growing backlog of software defects and/or requested features. If left unchecked, mounting technical debt can overwhelm a program with unplanned work to address a variety of issues, e.g., poor system performance, stability, and maintainability.

Claim. Addressing an increasing technical debt workload can have major impacts on productivity and overall team velocity, potentially leading to cost and schedule impacts.

During sprint planning sessions, reserve buffer in a sprint backlog; plan fewer items than the average sprint velocity, and absorb new user stories addressing recidivism or technical debt. As executed on several Major Defense Acquisition Programs (MDAPs) and consistent with industry best practice, this buffer should not be more than 10 to 20 percent of the team's velocity.

5.3 Identifying and Selecting Software Metrics

Waterfall development tends to fix requirements (scope) and then estimate time and cost, resulting in metrics and reporting to validate time and cost; e.g., earned value management. The Agile approach flips the so-called Iron Triangle by imposing fixed costs (based on the cost structure of dedicated teams) and time (through regular release cycles), then estimates scope (the amount of value the team can deliver). This approach shifts the focus of metrics and reporting to value delivery and team performance; therefore, Agile programs will leverage different metrics than Waterfall programs. Leadership and teams must use and understand these new metrics in addition to the old, to continuously improve program outcomes and results.

A software metric can be defined as a standard of measure to which a software system or process possesses an attribute. Even if a metric is not a measurement (metrics are functions, while measurements are the value obtained by the application of metrics), often the two terms are used as synonyms.

Program leaders and teams should use metrics to guide continuous improvement activities, discussed during sprint retrospectives and program reviews.

5.3.1 Metrics to Support Programmatic Decisions

Metrics are useful because they inform program decisions such as the following:

- Should the program be continued or canceled to free up resources for other initiatives?
- Does the value delivered satisfy the stakeholders' needs?
- Is new capability being delivered fast enough for programs to validate its value to the stakeholders?
- Is the system proving to be resilient and robust in practice?
- Are new capabilities flowing through the pipeline at an acceptable rate?
- Are requested capabilities delivered into production within acceptable time frames?
- Is the system meeting its cyber resilience requirements?
- Is the system ready for deployment to operational users?
- Is the system effective, suitable, and survivable in supporting the operational mission?
- What capabilities should we prioritize for delivery and deployment?

5.3.2 Questions to Consider in Selecting Metrics

Questions to consider when identifying and selecting metrics include:

- Who are the decision makers, what specific decisions do they need to make, and what level of detail and data do they need to inform their decisions?
- What are the program goals? How do they influence the metrics needed; e.g., transition to DevSecOps, migration to a Cloud-native application architecture, or reduction in fielded technical debt? What software metrics will provide the insight needed to manage to these goals?
- What are the program deliverables? What components of the deliverables are the stakeholders vested or interested in; e.g., capability, feature, function, or bug correction? What data and metrics are needed to track progress against development of deliverables and at what level?
- What software development risks, watch items, or concerns has the program identified that need to be supported by data collection and metrics; e.g., transition to DevSecOps, ability to meet planned staffing ramp rate, or dependency (assumption) on high velocity or productivity?

5.3.3 Qualities of a Useful Software Metric

To be useful, a software metric must be consistent, actionable[1], discoverable, consequential, and repeatable. Metrics will help to gauge progress toward a goal but should not be confused with the actual goal (e.g., a SMART goal – specific, measurable, achievable, relevant, and time-bound).

- Consistent and Actionable. The metric should be clearly defined, including what pipeline process it addresses and how to calculate it.
    - OUSD(R&E) Lesson Learned. If a metric triggers debate about what it means or what action to take with subsequent analyses, then that metric is of questionable value.
- Discoverable. The metric should be easily produced from naturally occurring data, i.e., a by-product of engineering or management activities.
    - OUSD(R&E) Lesson Learned. The metric should not require hours of work or a team of individuals to compute reliably.
- Consequential. The metric must connect to a program, project, or software development outcome.
- Repeatable. The metric can be produced over the software development life cycle and aggregated across projects to support analyses like benchmarking.

[1] An example of an inactionable metric could be the number of defects. While it is important to track the number of defects, this metric alone does not provide any actionable insights. To make this metric actionable, it would be necessary to analyze the characteristics of the defect, such as severity, when the defect was discovered, when it was fixed, where it was found, and its root cause. This information can then be used to assess system health, forecast maturity, target training, and improve the development pipeline.

Teams should address foundational questions such as the following when assessing the quality and source of software metrics:

- What is the program's software development engagement strategy? How involved will the stakeholders, including the Program Executive Officer or System Program Office, be with the software development contractor? How much control and involvement in the software development decision process will there be by the stakeholders? Who is in the role of the Product Owner, and is this a joint role? Who owns and is responsible for the technical baseline?
- Who owns and is responsible for the software factory? (DSB 2018) Is the software development contractor or staff using a software factory or tooling that enables automatic capture and generation of dashboards and reports, or does the software factory provide the possibility of self-service or direct access to such data?
- Does the software contractor(s) or subcontractor(s) have experience and demonstrated success with the software development methodology, software factory tool sets, architecture, and technologies that will be used?

5.4 Software Metrics and Reporting

An enterprise or program will tailor the measures it needs to implement and collect based on its information needs and objectives. Programs should expand on the minimum set of metrics as needed, considering metrics to measure progress.

Table 5-1 compares metrics covered by this OUSD(R&E) guide and other DoD and industry metrics guides ((OUSD(A&S) 2020); (PSM 2022); (DORA 2022)). The sections following the table further elaborate on the OUSD(R&E) metrics.

Table 5-1. Sample Metrics Mapped to Purpose

OUSD(A&S) AAF Defined Metric Purpose	OUSD(A&S) AAF Program Management Metrics	OUSD(R&E) SWE Guide	PSM CID Measurement Framework	DORA
Process Efficiency				
Process Efficiency	Story Points			
Process Efficiency	Velocity	Team Velocity	Team Velocity	
Process Efficiency	Velocity Variance			
Process Efficiency	Story Throughput			
Process Efficiency	Cycle (Resolution) Time	Cycle Time	Cycle Time / Lead Time	
Process Efficiency	Cumulative Flow Diagram	Cumulative Flow	Cumulative Flow	
Process Efficiency	Story Completion Rate			
Process Efficiency	Sprint Burndown Chart	Sprint or Release Burndown	Burndown	
Process Efficiency	Sprint Goal Success Rate			
Process Efficiency	Release Burnup			
Process Efficiency	Number / Percent of Stories Blocked			
Process Efficiency	Time Blocked and Time Blocked per Story			
Process Efficiency	Lead Time	Lead Time	Cycle Time / Lead Time	
Process Efficiency	Lead Time for Change			Lead Time to Change (LTR)
Process Efficiency			Staff Experience	
Process Efficiency			Team Turnover Rates Program Turnover Rates	
Process Efficiency			Reuse of Artifacts	
Process Efficiency			Backlog Readiness	
Process Efficiency			Defect Resolution	
Software Quality				
Software Quality	Acceptance Rate			
Software Quality	Recidivism Rate	Recidivism	Rework Defects Rework Hours Rework Stories	

OUSD(A&S) AAF Defined Metric Purpose	OUSD(A&S) AAF Program Management Metrics	OUSD(R&E) SWE Guide	PSM CID Measurement Framework	DORA
Software Quality	Defect Count by Story	Defect Trends	Defect Detection	
Software Quality	Change Fail Rate		Change Failure Rate	Change Failure Rate (CFR)
Software Quality	Mean Time to Recover/Restore (MTTR)		Mean Time to Restore (MTTR) / Mean Time to Detect (MTTD)	Mean Time to Recovery (MTTR)
Software Quality	Escaped Defects			
Software Quality	Code Coverage Rate			
Software Quality	Automated Test Coverage	Test Coverage	Automated Test Coverage	
Software Quality			Percentage of Code Base Available for Screening Percentage of Code Requiring Binary Analysis (no source code available)	
Software Quality			Percentage of Code Base Screened for Vulnerabilities	
Software Quality			Percentage of Code Requiring Binary Analysis (no source code available)	
Software Quality	Release/Deployment Failure Rate			
Software Quality		Cyclomatic Complexity		
Software Development Progress				
Software Development Progress	Release/Deployment Frequency		Release (or Deployment) Frequency	Deployment Frequency
Software Development Progress	Time Between Releases / Mean Time Between Releases			

OUSD(A&S) AAF Defined Metric Purpose	OUSD(A&S) AAF Program Management Metrics	OUSD(R&E) SWE Guide	PSM CID Measurement Framework	DORA
Software Development Progress	Progress against Roadmap			
Software Development Progress	Achievement Date of MVP / MVCR, Future Release Cadence			
Software Development Progress			Feature or Capability Backlog Burndown of Technical Debt Backlog Items	
Cybersecurity				
Cybersecurity	Intrusion Attempts			
Cybersecurity	Security Incident Rate			
Cybersecurity	Mean Time to Detect (MTTD)		Mean Time to Restore (MTTR) / Mean Time to Detect (MTTD)	
Cybersecurity	Mean Time to Remediate (MTTR)			
Cybersecurity			Common Vulnerabilities Enumeration (CVEs) Common Weaknesses and Exposures (CWEs) CVEs/CWEs Detected/Resolved Size of Attack Surface	
Benchmarking and Parametric Analysis				
Benchmarking and Parametric Analysis		Size, Schedule, Staffing, Effort, Defects	Committed vs Completed, Defect Resolution	

PSM (2022) provides additional definition and application of the defined metrics and measures:

> This guidance is intended to be used by team, program, and enterprise personnel who are implementing CID [continuous iterative development] approaches, as a reference for common, practical measures that can be used. The measures a program or enterprise chooses to implement and collect will be tailored based on alignment with its information needs and objectives, so they may differ from those described here. The measures presented are intended to be tailored and adapted to the development approach and environment. (PSM 2022)

A team at Google conducted research addressing the idea of optimizing software delivery performance. The results of this research are the DevOps Research and Assessment (DORA) metrics. DORA limited the metrics list to the four metrics identified by their research that are key DevSecOps indicators of software development lifecycle performance. The Office of the Under Secretary of Defense for Acquisition and Sustainment (OUSD(A&S)) is taking advantage of this research to assess AAF effectivity.

Sections 5.5–5.11 present commonly used metrics by management activity (Process Efficiency; Technical Performance and Mission; Software Quality; Software Productivity; Continuous Integration, Test and Release, and Operations; and Benchmarking and Parametric Analysis).

Programs are encouraged to draw not only on this guide but on referenced sources and their own experience to select the most effective metrics to help them deliver software-enabled warfighting capability. The DAU AAF website provides additional definitions and application of the AAF defined metrics and measures. As introduced,

> "The goal of generating metrics is to provide leadership, the product owner, team members, and other key stakeholders information and insights into the development effort to guide technical/programmatic decision-making, continuous improvement efforts, and remediation of blockers/impediments. Software teams should regularly review metrics as part of their sprint/release retrospectives and leverage metrics both for continuous improvement and to plan future iterations. Programs should have the ability to expand on the minimum set of metrics as needed, considering metrics to measure progress …" (DAU Metrics and Reporting 2022)

5.5 Process Efficiency Metrics

5.5.1 Team Velocity (Team Measure)

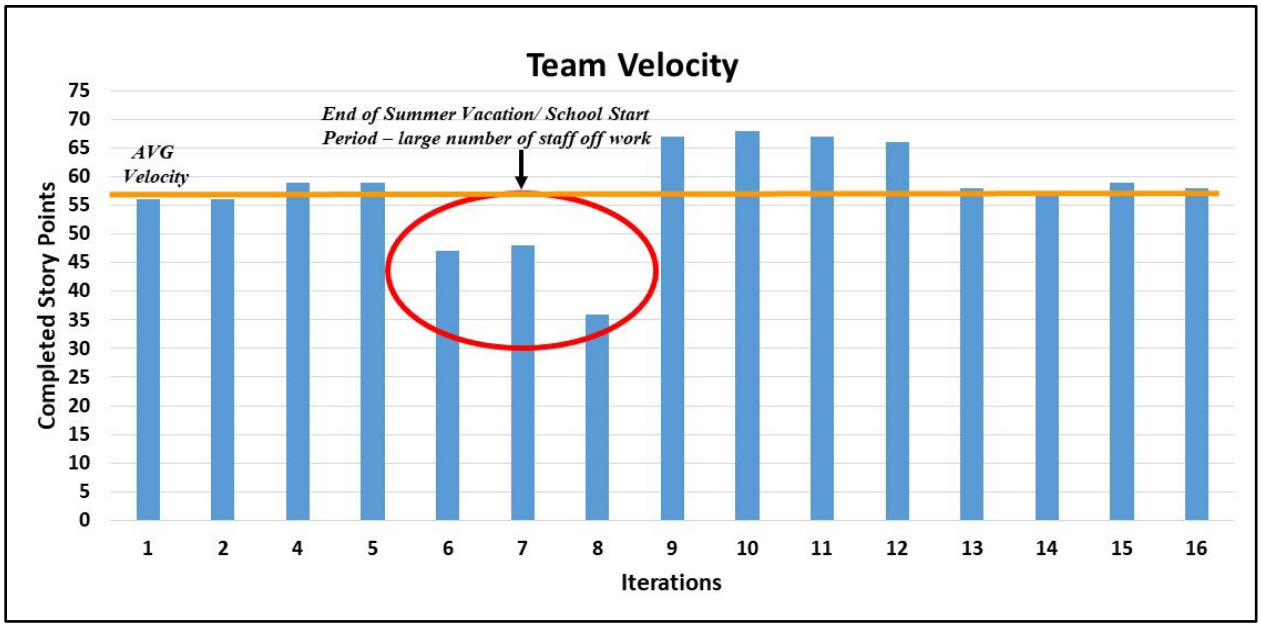
Team Velocity is a measure of team performance and the amount of work completed in an iteration, typically a count of completed story points (measures of complexity for a story) or

equivalent. Velocity calculations can be used to estimate the amount of work that a team will be able to accomplish in future iterations and to estimate when they will complete planned deliveries.

The Team Velocity metric can help answer the following questions:

- Is the team performing as expected?
- Does the team consistently meet the anticipated velocity?
- How much work can be accomplished by the team in a future iteration?

OUSD(R&E) Case Study 1. In an actual MDAP, the Team Velocity (Figure 5-1) remained relatively consistent through the period measured. Velocity dropped during Iterations 6-8 because of team members taking vacation just before, during, and just after school start. Iterations 9-12 saw increased story points completed related to increased defect resolution activity that occurred from systems testing activities post iteration 8.



Source: OUSD(R&E) Software Team

Figure 5-1. Team Velocity

By measuring Team Velocity, the program can:

- Set better delivery expectations and realistic sprint forecasts.
- Understand if the team is blocked (noted by falling velocity).
- Spot unforeseen challenges that were not accounted for during sprint planning.
- Investigate a process change result (noted by decreased, stable, or increased velocity).

Volatility in Team Velocity could indicate a process or processes are not working and need to be investigated.

Each team's Velocity metric is unique and should not be used to compare Team A with Team B in terms of performance or productivity. Each team has a specific estimation culture; for example, they may interpret story points differently. The goal is to optimize each team's work processes to ensure consistent performance over time.

5.5.2 Lead Time, Cycle Time, and Lead Time for Change (Team, Product, or Enterprise Measure)

Lead Time, Cycle Time, and Lead Time for Change are all used to evaluate efficiency in delivering value to the user and as predictors for estimating future work. Cycle Time and Lead Time for Change are both components of Lead Time. The differences are in when start times are measured (PSM 2022).

Lead Time measures the time from when a customer makes a request to when the team delivers the working product to the user. Lead Time includes up-front activities such as identifying backlog, prioritizing, planning, analyzing requirements, and design. Lead Time can be heavily influenced by the value and complexity of the work. For example, the product owner may deprioritize a user request because it is low value and high complexity (hard to deliver).

Lead Time can help answer the following question:

- How long does it take to deploy an identified feature/capability once a request is submitted?

Cycle Time and Lead Time for Change both look at different aspects of delivery.

Cycle Time is the elapsed time from when the team begins development work until the work is completed. This measures the efficiency of the team's value delivery. It does not include the up-front effort needed to define and prepare the work to be implemented (e.g., backlog review and prioritization).

Lead Time for Change provides the elapsed time between when the work is ready for release and when it is actually released to the customer or end user. This metric shows how efficient the program is at releasing new value to customers or end users.

These two metrics answer two important questions:

- Cycle Time answers the question: Once development starts, how long does it take to get the code ready for release?
- Lead Time for Change answers the question: Now that we have code ready for release, how long will it take us to get it in use by the customer?

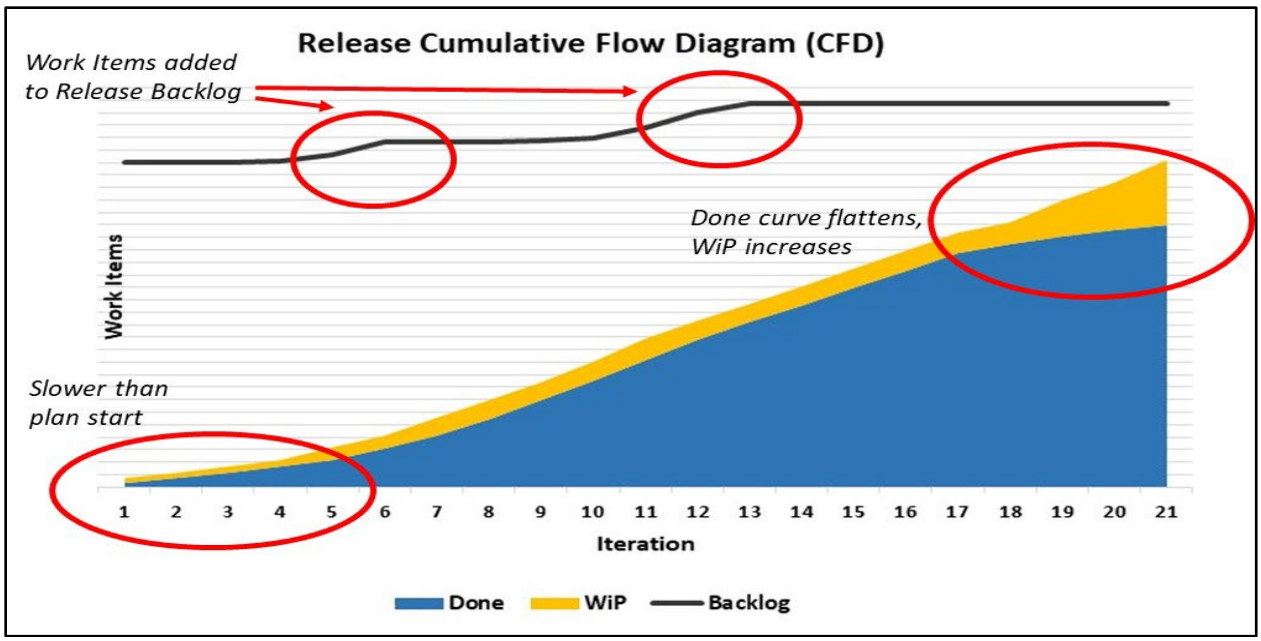
5.5.3 Cumulative Flow and Throughput (Team, Product, or Enterprise Measure)

Cumulative Flow (whose first derivative is Throughput) indicates the total value-added work output by the software team. It is typically represented by the units of work (e.g., tickets or issues) the team has completed over intervals of time. The Cumulative Flow metric needs to be aligned with current business goals. For example, if the goal is to release new bug-free modules in this sprint, one should look for a large fraction of defect tickets being resolved.

Cumulative Flow can help answer the following questions:

- Is the flow of work moving forward through the value stream (through the process workflow states)?
- Is the throughput of work predictable?
- Are there queues or delays in process workflows that prevent optimizing throughput?

OUSD(R&E) Case Study 2. In an actual MDAP, early work item throughput was well below the ideal or planned throughput to complete all work items for the release (Figure 5-2). This low result was due to slower-than-expected personnel clearance processing, delaying availability of planned developer resources. The backlog also increased because of additional work items (capability and functionality), increasing work items by 18 percent over the initial plan. As the personnel resources became available, the pace of work item transition to Done increased and work in progress (WiP) became relatively consistent. Starting at iteration 18, WiP started to rise rapidly and Done flattened out, reducing throughput. This change was caused by hardware availability issues, which prevented work transition to Done. The combination of the very slow start, the additions to backlog, and the late hardware issues negatively affected throughput, preventing the program from completing all the planned work on time.



Source: OUSD(R&E) Software Team

Figure 5-2. Cumulative Flow Diagram

Cumulative Flow/Throughput terminology:

- Throughput is the number of Work Items completed per unit of time. It corresponds to the slope (rise over run) of the cumulative Done line.
- Done is work that has completed all development, integration, test, and other transition workflow states.
- Work in Progress (WiP) is work that has started development activity but has not completed the final workflow state (Done, for purpose of the Cumulative Flow Diagram).
- Backlog captures user needs in prioritized lists, and includes new/modified capabilities/features, defect fixes, infrastructure changes, or other activities that a team may deliver to achieve a specific outcome.

If Throughput declines, this could indicate the team is:

- Blocked somewhere in the process or has encountered a bottleneck(s) preventing consistent throughput and delivery.
- Overloaded, if not meeting throughput targets. Ask the following questions:
  - Is WiP a focused, singular problem?
  - Are WiP limits in place?

- Have processes become inefficient or a bottleneck appeared?

OUSD(R&E) Lesson Learned. Workflow should be balanced, with entry into WiP balanced with departures or completion of work.

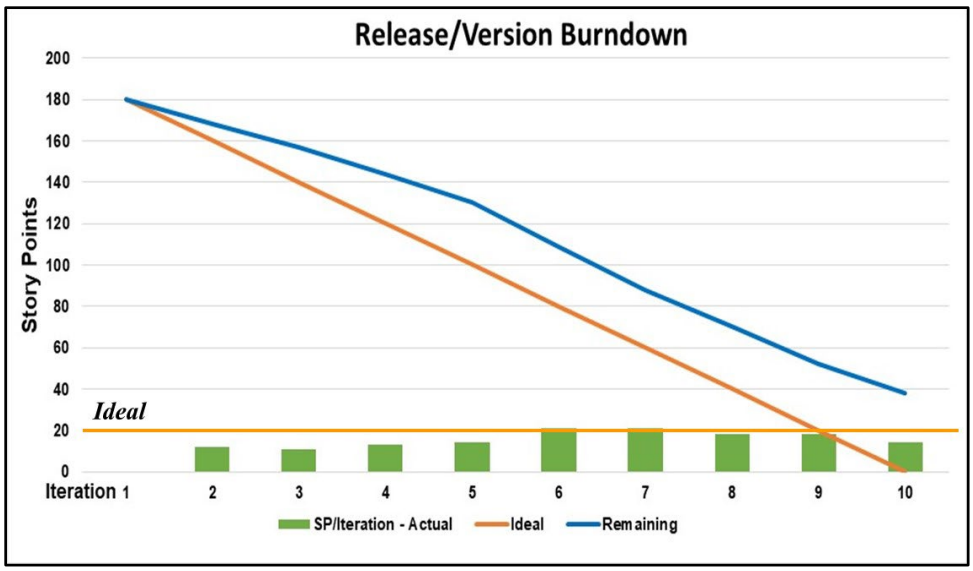
5.5.4 Sprint or Release Burndown (Team, Product, or Enterprise Measure)

Sprint or Release Burndown metrics are used to monitor completed work items (e.g., stories, features, capabilities) versus planned work items for an iteration (sprint), release, or capability. Work items may include design, code, test, and all supporting activities (e.g., requirements development, configuration management, quality engineering). Progress toward completing planned work is depicted graphically to provide an indicator of the likelihood of meeting planned goals (PSM 2022).

The goal of the Scrum/Team is to consistently deliver all work, according to the sprint forecast. The Release Burndown metric can help answer the following questions:

- What is the status of the iteration, release, or capability?
- Will all the remaining committed work be completed as planned?
- What are the features/capabilities at risk of not being completed as scheduled?
- What are the trends in execution relative to plan?

OUSD(R&E) Case Study 3. In an actual MDAP, the Release Burndown chart (Figure 5-3) shows that the team was unable to complete all planned stories (~38 stories unfinished or 21% variance). This condition met the criteria for further review. Review of this and the previous release iterations revealed that the team was consistently being asked to deliver too much work. Additional permanent resources were added to the team, which eventually allowed them to consistently finish planned work while addressing the backlog.



Source: OUSD(R&E) Software Team

Figure 5-3. Release/Version Burndown – Plan vs. Actual

Look for a gradual reduction (consistent/smooth glide slope) in "remaining values" rather than a dramatic drop as the latter will indicate that the work was not assigned in such a manner or broken down into granular pieces.

A team's burndown is rarely perfectly smooth (as represented by the ideal line in Figure 5-3). It can vary for several reasons, including inaccurate estimates or changed scope; however, consistently missed deadlines or unfinished work at the end of an iteration or release should be analyzed and addressed.

Table 5-2 summarizes what the sprint or Release Burndown metric may indicate. The observation in both cases may be indicative of improper planning or forecasts that did not include valid assumptions; e.g., productivity.

Table 5-2. Sprint or Release Burndown Metric Indications

Observation	Possible Indication	Possible Corrective Action
Consistent early sprint finishes	A lack of scheduled work/issues for the given sprint. The Team may not be committing to enough work.	1. Adjust workload 2. Reduce Team size
Consistently missed sprint deadlines or significant unfinished stories at the end of an iteration or release	Can indicate a gap in planning, features/stories are too large or that the Team was asked to deliver too much work.	1. Add resources to the Team 2. Reduce workload

5.6 Technical Performance and Mission Effectiveness Metrics

Teams should include metrics that are operationally relevant and address mission effectiveness. Such metrics are highly context dependent. The specific mission and operational context may vary widely. For example:

- A guided munition may be measured on its accuracy, its ability to evade countermeasures, time it takes to target and launch, or ability to adapt and make determinations in flight.
- An AI/ML based sensor fusion and threat recognition system may be measured on precision and recall, time to make decisions, resilience in the face of sensor loss, resistance to cyberattack, and adversarial AI countermeasures.
- An autonomous vehicle may be measured on its ability to navigate in adverse weather and terrain conditions and complete mission-related tasks in required time frames with high reliability.
- Ability of the system to fight through the loss of specific system elements, data, and communication capabilities and still complete mission threads.

Whatever the context, the team working with trained operations personnel should develop technical measures of mission effectiveness to augment the software process metrics.

5.7 Software Quality Metrics

5.7.1 Recidivism (Team or Product Measure)

The Agile framework or methodology accounts for change at any stage of the project. However, shifting requirements can negatively impact a team's performance and result in misapplied labor hours and developed code.

The project goal is to ensure the team can work at a consistent pace when presented with both static and dynamic requirements. The Recidivism metric can help gain insight into this ability.

Recidivism (or rework) is the measurement of tasks (percentage of stories) as they move backward (returned to the development team) in the predefined workflow. For example, if a task moves from development to quality assurance, fails validation, and moves back to development, this activity will increase the recidivism rate.

- OUSD(R&E) Lesson Learned. A high recidivism rate (nominally 15-20%, or higher) may indicate incomplete or inconsistent requirements and may need to be investigated. It may also point to other issues, such as code not meeting verification/validation requirements or bad (erroneous or incomplete) test scripts.

5.7.2 Defect Rate (Team or Product Measure)

A defect is a condition in a product (e.g., software, system, hardware, documentation) that (1) does not meet its requirements or end user expectation; (2) causes the product to malfunction or to produce incorrect or unexpected results; (3) causes the product to behave in unintended ways; or (4) leads to quality, cost, schedule, or performance shortfalls. Defects may be documented in problem reports (trouble tickets) or may be added to the backlog for consideration in future iterations (PSM 2022).

Defect terminology:

- Escaped Defects. Defects detected, or resolved, after release of the product and version containing the defect. Defects are generally tracked separately for internal and external releases.
- Contained Defects, also known as Saves. Defects detected and resolved before internal or external release of the product and version containing the defect.

Why is the Defect metric important?

- A key tenet of Agile/DevSecOps is continuous improvement. Reviewing where escaped defects occurred and how they happened is important to understanding and correcting potential process flaws.
- Measuring the integrity of the CI/CD pipeline to avoid errors discovered in operations is paramount.

The Defect metric can help answer the following questions:

- How many defects were contained (discovered) prior to internal release?
- How many defects were released (escaped) to an internal customer (e.g., Integration and Test, Formal Test) or released (escaped) to an external customer (e.g., end users)?
- For each major release, how many defects were detected in internal development (contained, saves)?
- What is the ratio of escaped defects (internal and external) to all defects?
- Does committed work (stories, features, capabilities) work as expected?

A Data Review Board (DRB) or Configuration Control Board (CCB) should evaluate the probability, severity, and occurrence of each defect to define the corrective path forward.

OUSD(R&E) Lesson Learned. Programs must address Severity Level 1 and Severity Level 2 defects (against the Definition-of-Done within the Agile framework) to successfully achieve DoD acquisition milestones.

Severity Level 1 defects (Fatal) are, for example:

1. A defect completely blocks testing of the Product Backlog Item
2. A defect causes failure of the functionality
3. No work-arounds identified
4. Major data corruption

Severity Level 2 defects (Major) are, for example:

1. A defect impacts major functionality or data
2. A defect causes failure of part of the product's functionality
3. Work-around difficult to complete
4. A defect impacts mandatory field validations (including calculations or business rules) or performance
5. Data incorrectly displayed, corrupted, or absent (including message traffic)

5.7.3 Test Coverage and Code Coverage (Product or Enterprise Measure)

In an iterative development approach, software teams should not only efficiently verify new features but also ensure prior functionality is not affected. Verifying the features and functionality manually can be time-consuming.

There are two forms of coverage metrics: Code Coverage and Test Coverage. Code Coverage refers to how much of the code gets executed by the tests (e.g., percentage of statements executed, or branches taken). Test Coverage refers to how much of the specified behavior is exercised by the tests (e.g., percentage of functional requirements, user stories, non-functional requirements, stress tests.) Both measure the quality of the testing and both have a role.

Typically, Code Coverage is verified primarily in structural (white box) testing at the unit level, and requirements are verified primarily in functional or system test. Efficiency and throughput can be enabled by automated test suites executed at multiple levels (unit level, functional level, regression testing) (PSM 2022).

Often, automated test suites are integrated directly in the development pipeline of the software factory and invoked upon each code commit and build, or in nightly regression test batch jobs.

Test results (tests passed, tests failed) can be distributed automatically in an email so anomalies affecting the code quality and pipeline can be quickly identified and resolved.

Modern software testing regimen emphasizes automated testing. Effort spent on test automation usually pays off in increased quality, decreased cycle time, and fewer escaped defects. Consider the following:

- How much of the testing is automated?
- How many tests have been validated and approved?
- How much credit is given in formal test; e.g., DT/OT, for automated test?

5.7.4 Cyclomatic Complexity (Product Measure)

Cyclomatic Complexity is a measurement to determine the stability and level of confidence in a program. It measures the number of linearly independent paths through a program module. Programs with lower Cyclomatic Complexity are easier to understand and less risky to modify.

Cyclomatic Complexity is computed using the control-flow graph of the program: the nodes of the graph correspond to indivisible groups of commands of a program, and a directed edge connects two nodes if the second command might be executed immediately after the first command. Cyclomatic Complexity also may be applied to individual functions, modules, methods, or classes within a program.

5.8 Software Productivity Metrics

Software productivity is defined as the ratio between the functional value of software produced to the effort (staff) and duration (time) required for that development. There are many ways to measure productivity; however, two types of metrics are common: size- and function-related.

5.8.1 Size-Related Metrics

Size-related metrics indicate the size of an outcome from an activity, e.g., lines of written source code. With respect to lines of written source code, there are two ways to count each line: (1) count each physical line that ends with a return; or (2) count each logical statement and consider that as a line of code. This difference makes it difficult to compare software simply by lines of written source code (or any other metric) without a standard definition.

Programs should keep in mind that using lines of code written as a performance metric could lead to the risk of a decline in quality as developers may begin to favor quantity over quality in order to satisfy the metric. If volume of code and errors are both used as a productivity measure,

for example, the development team might avoid tackling difficult problems to keep their lines of code up and the error counts down.

OUSD(R&E) Lesson Learned. Many of the systems developed within the DoD are real-time, communication, or embedded systems in a constrained environment such as a spacecraft, missile, or aircraft. Bloated, inefficient code will prevent the system from achieving its performance requirements.

5.8.2 Function-Related Metrics

Function-related metrics represent the amount of useful functionality shipped during a set period of time, such as function points or story points. Function points can be compared across teams or projects; however, story points cannot. Work that is "1 story point" for one team may be "3 story points" for another team. A team that is completing 40 story points per week is not necessarily more productive than a team completing 10 story points per week as the definition of a story point is unique to the team.

5.9 Continuous Integration, Test and Release, and Operations Metrics

Tables 5-3 and 5-4 provide additional metrics that teams can use to guide technical programmatic decision making, continuous improvement efforts, and remediation of impediments. See also DoDI 5000.02 (2022); DAU Metrics and Reporting (2022); and PSM (2022) for additional definition and application of these metrics and measures.

Table 5-3. Continuous Integration Metrics

Category	Metric	Measure
Code and Automated Build/Release	Build Automation	# Steps Automated, also calculated as a percentage
	Average Builds per Day/Week	# Pass, # Fail, also calculated as a percentage
	Duration per Build	# Minutes, # Hours, # Days - Minimum, Maximum, Average
Development Test	Unit Test Coverage	% Coverage, % Automated
	Static Code Analysis Coverage	% Coverage, % Automated
	Functional Thread Test Coverage	% Coverage, % Automated
System Integration	Integrated Build Frequency	# Pass, # Fail, # Deployments per Day/Week
	Integrated Build Recovery	Average time (minutes, hours) between failed deployment and system restored to good state

Category	Metric	Measure
	Change Volume	# Deployed Story Points, Equivalent Source Lines of Code (ESLOC), Source Lines of Code (SLOC), etc. in time-series
	Automated Logging, Monitoring, and Security Controls	For Cloud, Enterprise, and other Compute Intensive Systems % Automated Logging, Monitoring, and Security Controls

Table 5-4. Test and Release Metrics

Category	Metric	Measure
Test and Release	System Test Coverage	% Coverage, % Automated
	Test Progress	# Planned vs. # Attempted, categorized by Passed, Failed, Blocked
	System Test Frequency	# Tests per Build, delineated by Day or Week
	Functional Test Frequency	# Tests per Build, delineated by Day or Week
	Fix Fail Rate	% Discrepancy Report Fixes that reappear or fail in verification

Site Reliability Engineering (SRE) focuses on tasks that have historically been accomplished by operations teams, often manually, and instead gives them to engineers or operations teams who use software and automation to solve problems and manage production systems. SRE is a valuable practice for creating scalable and highly reliable software systems. Table 5-5 identifies applicable Operations metrics (Red Hat 2022).

Table 5-5. Operations Metrics

Category	Metric	Measure
Operations	Availability, SW Uptime by Environment	# Total Active Environments in Operation, less # those in creation, recovery, and maintenance
	Reliability	# Hours/Day, # Days/Week, also calculated as %
	Performance by Critical Capability	Response Time vs. Threshold/Objective
	Service/Environment Restarts	# per Day, % Automated
	Help Desk/Field Incident/Problem Ticket Volume	# New, # Closed in time-series
	SW Patches	# Available, # Applied in time-series, also Applied calculated as %
	Stability	Hours/Days, Service/application Uptime between Restarts
	Activate Recover Environments	Time to create in Seconds/Hours
	Environment Utilization in time series	# Days, in time-series
	% of automated environment monitoring of features/controls throughout lifecycle stages	Create/activate/recover stages

5.10 Benchmarking and Parametric Analysis

A common set of core metrics collected across a program's life cycle provides the basis for benchmarking, parametric analysis, and other forecasting and statistical modeling activities.

Capturing a project's past performance as a benchmark is critical to understand how that project might perform in the future. Benchmarking involves calibrating specific project actuals (e.g., software size and capability, effort, duration, quality (defects)) and using those to evaluate current or project future performance.

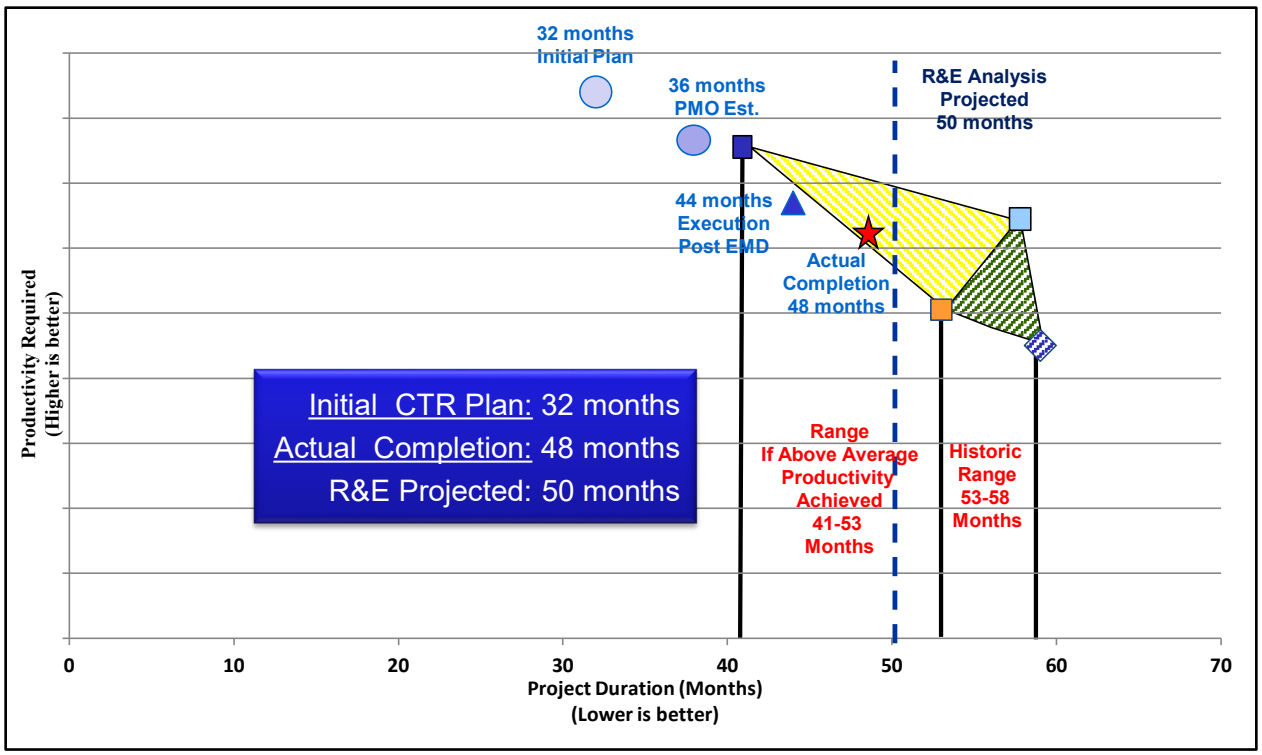
OUSD(R&E) collects a set of core software metrics (Table 5-6), common across all programs. This data is used for benchmarking, parametric analysis, and other forecasting and statistical modeling activities supporting program reviews.

Table 5-6. Core Benchmarking Metrics

Category	Metric	Measure
Size	SLOC, ESLOC, Story Points*, Function Points, etc.	# Planned and Actual SLOC, ESLOC to capture new / modified / reused / auto-generated, by Build, in time-series (optionally by Computer System Configuration Item (CSCI)). Requires common code counting formulas/tools, e.g., Unified Code Counter (UCC) *Note: While story point magnitudes are not directly comparable across teams, derived measures such as % overrun can be compared
Schedule	Software Schedule by Build and Release	Months Planned and Actual, reflecting any updates/changes
Staffing	SW Context	# Planned and Actual Full Time Equivalent (FTE), reflecting any updates/changes (optionally by labor category)
Effort	SW Context	# Planned and Actual Labor Hours, in time-series (optionally by CSCI)
Defects	Defects	# Actuals, by Severity Defects to capture discovered / fixed / closed / deferred by severity/priority, by Build, in time-series (optionally by CSCI)

The core software benchmarking data collected by OUSD(R&E) (Table 5-6) can be filtered by different criteria like domain, size, development methodology, etc. This is particularly useful for selecting data that is analogous to the program/project/data set being analyzed.

OUSD(R&E) Case Study 4. The OUSD(R&E) Software Team used benchmark data from analogous programs to help a program office determine that the contractor's (CTR) proposed software development plan was very aggressive, reliant on achieving very high productivity, and had a low probability of completion. Analogous programs showed historical performance in the 53 – 58-month range (Figure 5-4, green triangle). At 32 months, the CTR's plan required much higher productivity than analogous historical programs and much higher productivity than recorded by any program in that domain. OUSD(R&E)'s analysis projected an approximate 50-month development effort. The contractor was unable to realize the expected productivity gains and did not finish the project in the initially projected 32 months. However, the CTR did finish the project in 48 months, better than the historical efforts but still over 16 months later than the initial plan.



Source: OUSD(R&E) Software Team

Figure 5-4. Comparing Planned, Forecast, and Actual Performance

5.10.1 Software Size

Contractors often provide optimistically high software reuse estimates, with substantial risk hidden in the assumptions. The proposed reuse code often was not developed for reuse (e.g., to new cybersecurity or certification requirements) or for new architectural constructs (e.g., monolithic to microservices), which can result in the need for significant modification of code or for new coding if the modification becomes costlier than new coding.

OUSD(R&E) Finding. Based on data collected and assessed across MDAPs, most programs experience overall size growth of approximately 25 percent during Engineering and Manufacturing Development (EMD). Analysis indicates the growth can be attributed to a few common causes, such as optimistic contractor proposals and estimates, overly optimistic reuse estimates (reused code reduces the amount of required mod/new code), and requirements (poor stakeholder involvement, volatility/churn, evolving).

OUSD(R&E) Finding. Reuse of code from a monolithic system will probably require extensive modification for use on a microservices-based system (via the strangler pattern). With the transition to more modular modern architectures, the need for extensive modification has been increasingly observed, which has resulted in programs underestimating the effort needed for code reuse. A recommended best practice is that for those programs that expect significant reuse, to review the proposed reuse codebase for applicability and assess potential modification risk.

A recent program did just that and uncovered a 12 to 16-month schedule risk based on the assembled panel's code review and discussions with contractor development staff. These findings helped the program identify the risk early and work with the contractor to mitigate that risk.

5.10.2 Software Schedule

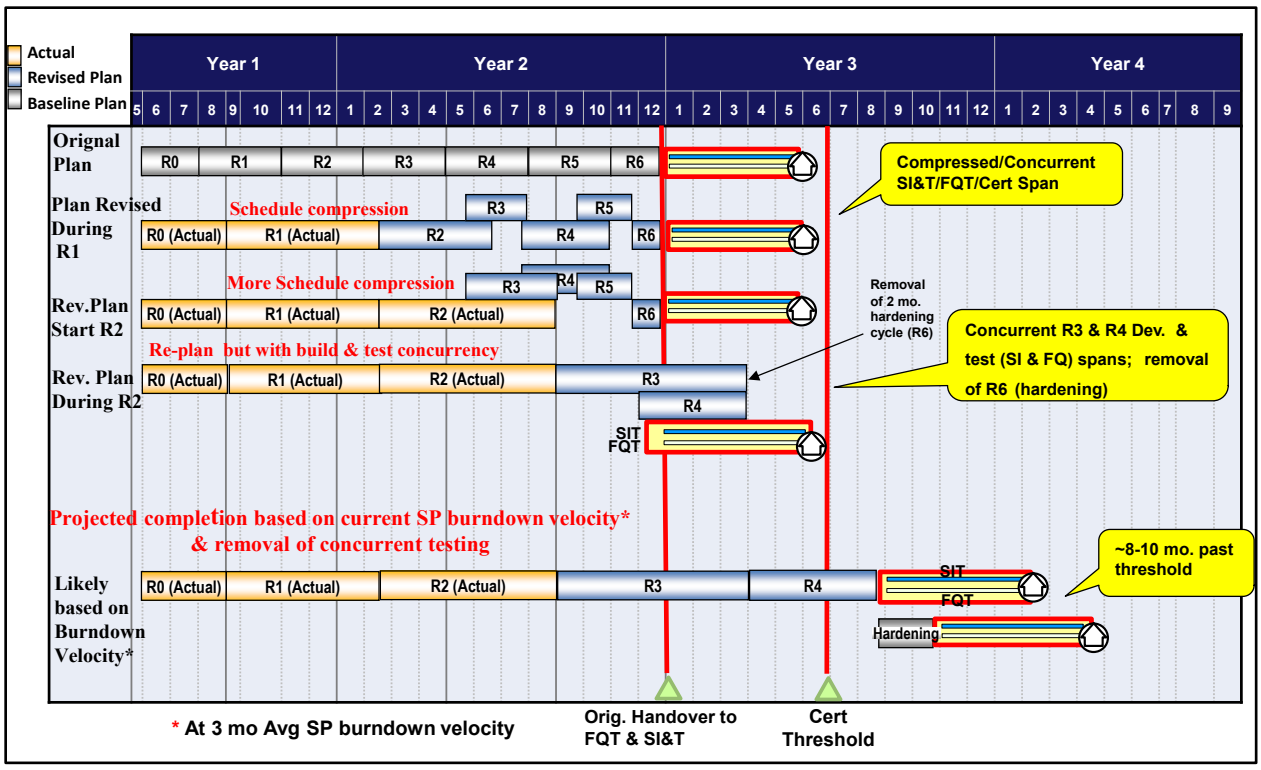
Programs often try to compress the software development effort into the available schedule without commensurate scope reductions. A highly compressed schedule rarely provides positive results, as the below example from a recent program shows.

OUSD(R&E) Finding. Data collected across MDAPs show staffing and staff skill sets continue to be a typical area of risk, particularly as the DoD becomes increasingly dependent on software.

OUSD(R&E) Case Study 5. The program was an Agile follow-on development effort (Figure 5-5) and illustrated the hazards of schedule optimism. The software development effort was organized in 2-week sprint cycles leading to Program Increment (PI) releases every 3 months. The plan consisted of a bootstrap PI, followed by five PI releases and a sixth clean-up PI. The bootstrap PI (Release (R)0) and clean-up PI (R6) were slightly shorter than 3-months in duration. From the beginning, the schedule was compressed, by running final System Integration and Test (SI&T) and Full Qualification Test (FQT) in parallel, to achieve system certification by the threshold date. The initial bootstrap PI (R0) ran into several issues resulting in a ~159% of plan overrun.

The program added temporary staffing to compensate. As it became clear that R1 would not complete on time, the program revised the plan creating more schedule compression/parallel tracks and added additional staffing to increase burndown velocity. However, R1 also resulted in an even greater overrun than R0. The schedule was un-executable within the defined factors, but the program resisted either de-scoping or extending the delivery schedule. Eventually, the program de-scoped but still tried to retain the original threshold certification date by developing two Releases in parallel, in parallel with SI&T and FQT. Clearly, this is a high-risk approach.

Eventually, the program paused to assess how, or whether, to proceed. Although the program had added staffing, the new staffing did not immediately improve burndown because of the learning curve for new personnel. The story point burndown projections became increasingly back loaded and parallel efforts created new issues.



Source: OUSD(R&E) Software Team

Figure 5-5. Schedule Optimism vs. Realism

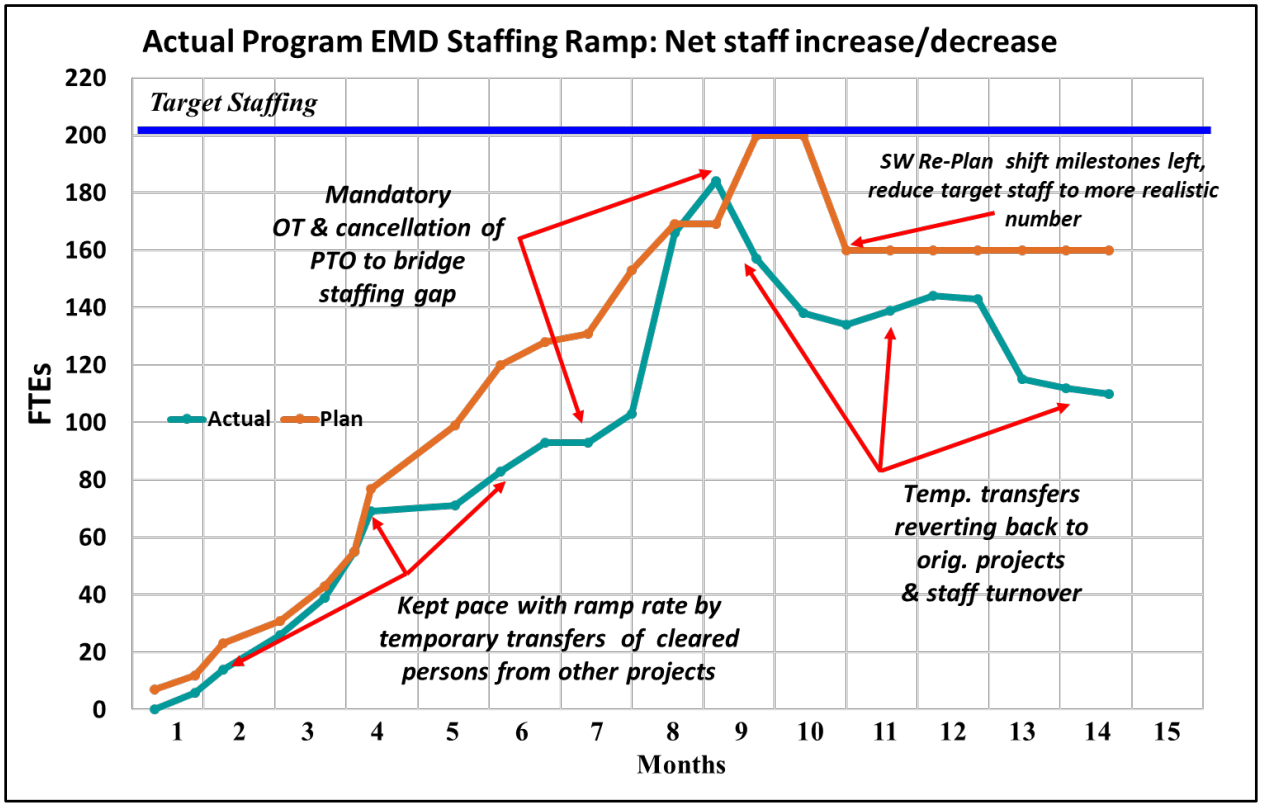
5.10.3 Software Staffing

New programs often suffer from optimistic staffing assumptions. OUSD(R&E) has observed that the increased demand for software-enabled functionality, increasingly shorter capability demand cycles, and the competitive software job market for skilled cleared individuals has led to software staffing challenges. Software staffing ramp rates are frequently optimistic given the scale of MDAPs. Program management offices and contractors often struggle to find the level of experience and skill sets needed and/or getting many individuals cleared at the pace planned. The clearance process can take many months and is out of the program's control, making it difficult to plan and keep staff productive until the clearance is granted (Tate 2020).

OUSD(R&E) Finding. Data collected across MDAPs show staffing and staff skill sets continues to be a key risk that is frequently encountered, particularly as the DoD pushes to adopt modern software development methodologies and toolsets, and program offices are encouraged to take ownership of the technical baseline and the software factory, for example, Platform One.

OUSD(R&E) Case Study 6. An MDAP had planned to ramp up its software staff from 10 experienced people to 200 people over a span of 9 months (Figure 5-6). This amounts to a ramp rate of 21 people per month. To work the project those people needed to have proper security clearances and experience. Initially the contractor was able to meet the planned ramp rate by

transferring cleared people from other projects. Soon competition with other programs in a tight market for qualified people led to increased difficulties meeting the planned ramp rate. Mandatory overtime and cancellation of leave temporarily led to increased staff hours covering some of the shortfall, but also increased retention issues. Eventually, because of difficulties reaching planned staffing and productivity levels, the contractor delayed the development milestones to conform to more achievable staffing levels.



Source: OUSD(R&E) Software Team

Figure 5-6. Planned vs. Observed Staffing Levels

5.10.4 Software Effort

Underestimation of the amount of effort needed to create software is the major driver of cost overruns and schedule delays. The amount of effort needed to deliver a feature or capability is a function of productivity. Productivity is loosely defined as the ratio of the amount of output to unit of input (Productivity = Output ÷ Input). As stated in the Software Productivity section above, two types of productivity metrics are commonly encountered: size- and function-related.

DoD productivity data collected to date shows a wide variance in productivity rates (ESLOC/hour) not only across different domains, but also within domains. The variance between high and low performance programs is stark, requiring the critical analysis of planned and actual performance. why there are such wide differences in productivity.

Claim. For benchmarking, parametric, and other forecasting analysis and statistical modeling activities, ESLOC/hour is the most frequently encountered productivity basis. DoD contractors have captured extensive historical data in SLOC and ESLOC form. In particular, the SRDR DID requires the use of the latest Government-approved version of the University of Southern California (USC) Center for Systems and Software Engineering (CSSE) Unified Code Counter (UCC). The Government has conducted Independent Verification and Validation (IV&V) that ensures the code counter can be used on Government systems and provides standardized results across the DoD community. (CAPE) Despite an increase in Agile/DevSecOps prevalence, ESLOC/hour continues to be collected and used, allowing for historical program productivity comparisons.

5.11 Weibull Analysis of Defect Trends

This section discusses Weibull analysis, a statistical technique that uses software defect data to assess and forecast software maturity. It describes the concept and provides tips on preparing data and on interpreting the results.

The underlying mathematical technique dates from the early 1960s and has been used with success in a number of fields. These include:

- Reliability measurement of materials; e.g., mechanical parts subject to wear (aircraft and ship propellers).
- Modeling the spread of fast-moving computer cyber-attacks over a network; e.g., Code Red.
- Modeling and analysis of cascading network failures.

The U.S. Air Force published a Weibull Analysis Handbook (Pratt and Whitney 1983) that provides instructions on how to do Weibull analysis and an understanding of the commonality between the military and industry.

This modeling technique has been applied to software engineering to bring analytical rigor. Two major applications include:

- Modeling the staffing levels over time on large software development projects.
- Modeling the software defect insertion and detection rates to assess software development process effectiveness as well as projecting the level of latent defects (technical debt.) in software components or systems.

The Rayleigh distribution is a special case of the Weibull distribution. This distribution forms the mathematical basis behind much of the parametric modeling approaches to modeling staffing rates and statistics-based baselined projections of defects over time. The underlying assumption is that people make mistakes (i.e., create errors in the form of defects) at a constant rate; and

people are on-boarded rapidly at the beginning of a project and later off-loaded at a slower rate as development and testing activities are completed. This phenomenon is best emulated by a Rayleigh distribution. The Rayleigh curve can be used to estimate defect insertion as well as defect detection. By minimizing the time between defect insertion and defect detection to development teams can reduce the amount of rework and operate more efficiently.

5.11.1 What Weibull Defect Trend Analysis Indicates

As software moves through phases from development to production, the plot of cumulative number of defects discovered over time follows a regular pattern. By measuring and observing this pattern, the Program Manager can gain insight into how well the software is maturing. (This is as true for software developed in Agile development processes as it is for more traditional development processes. However, the Rayleigh curve may not accurately reflect the defect profile observed for a fixed number of software staff during Sustainment.) The manager can gauge whether the projected reliability and performance of the system is in line with expectations.

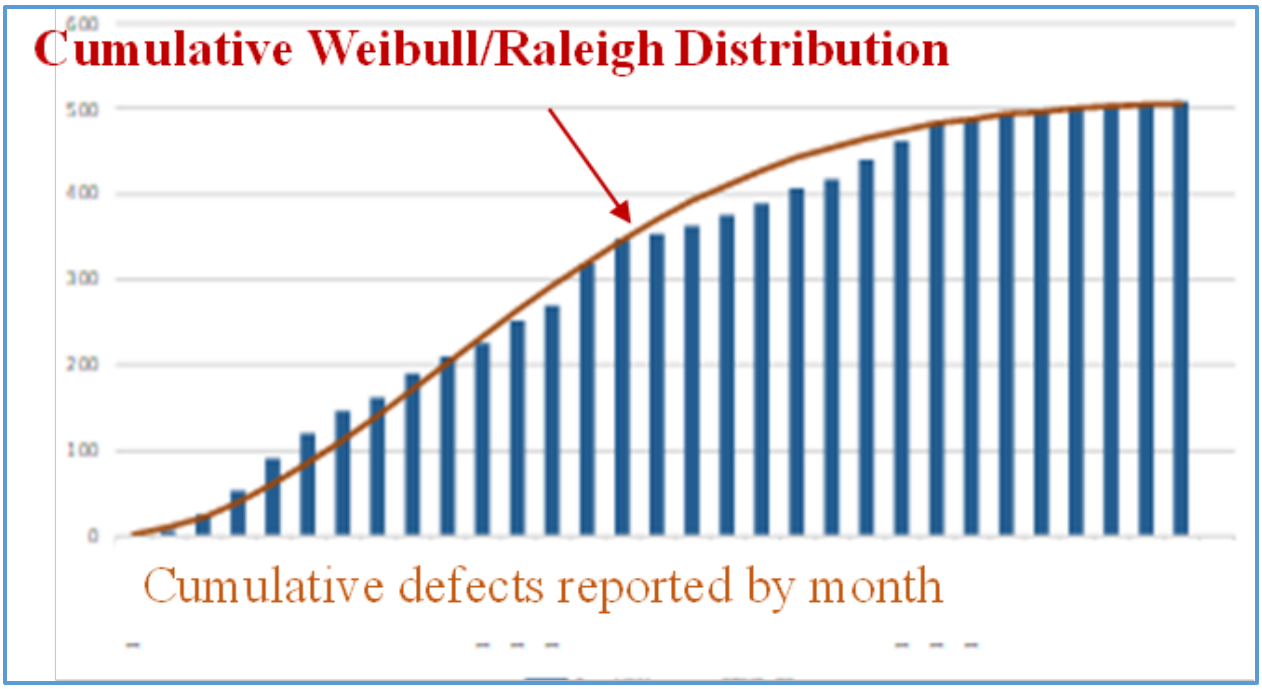
Kan (2003) reports that this method requires about 60 percent of the defects to have been discovered to produce significant forecasts. Even so, the accuracy of the forecasts for practical purposes falls in the range of plus or minus 10 to 20 percent. It can provide a ballpark estimate of future defect discovery rates, but the conclusions of this analysis should always be correlated with other data.

5.11.2 How Weibull Defect Trend Analysis Works

Time series Defect Data is used to fit a probability distribution for software defects over time.

Early in a project when little code has been produced, there are few opportunities to discover defects; there are fewer test cases to exercise and less capability to test. As the project progresses and more code is produced, more tests will be exercised. This increased testing uncovers more defects as that is reflected by the increased slope of the curve. As the system matures, this rate of defect discovery will slow, and the curve will begin to flatten.

Figure 5-7 taken from several years of actual defect data from a major DoD weapons program illustrates this behavior. This form of analysis was able to successfully forecast the maturity of this weapon system and its eventual successful fielding 6 months later.



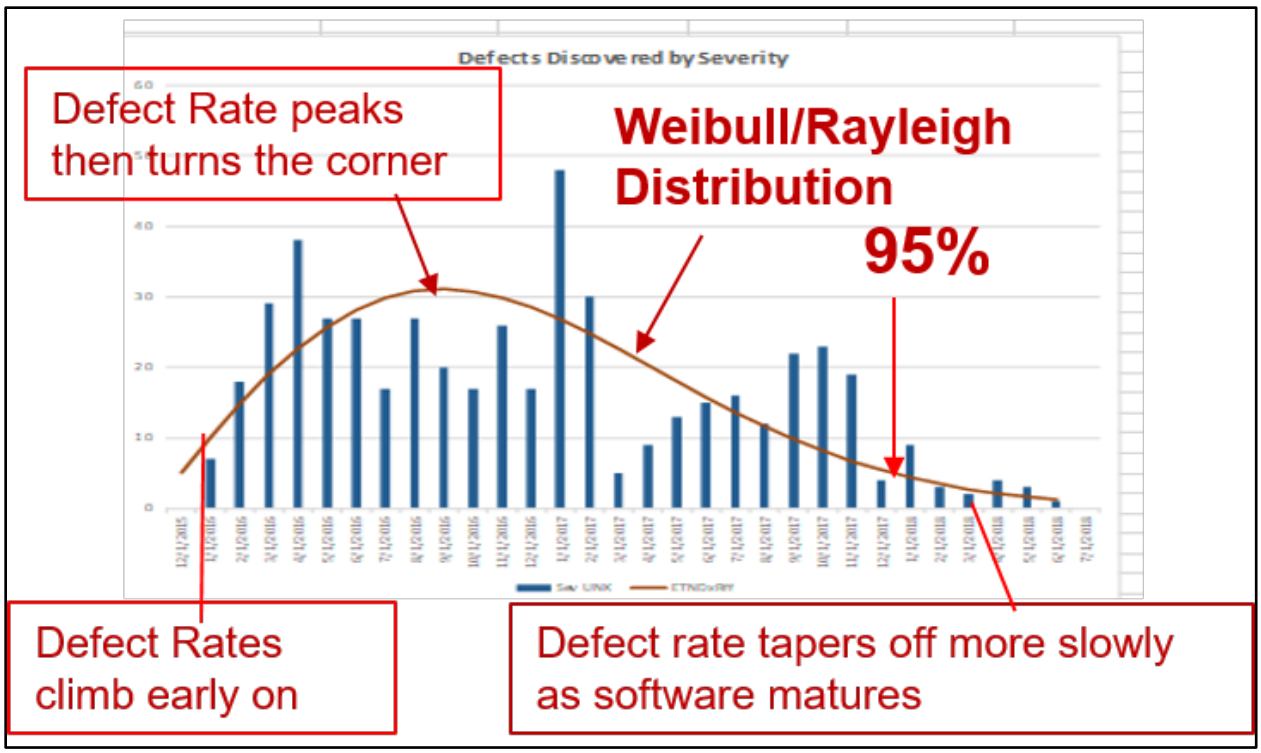
Source: OUSD(R&E) Software Team

Figure 5-7. Cumulative Defects

The smooth line shows a Weibull distribution curve calculated from defect report data. Best fit determines Weibull model parameters (size and shape) that define a curve W(t) that relates percent defect removal to time. Analysts can then solve for either, in terms of the other.

As software is developed, the rate of defect discovery climbs, peaks, then decreases ever more slowly as the software matures. Duration sufficient to remove 95 percent of defects is expected from robust systems engineering practices (Kan 2003). The model can forecast when the 95 percent point will occur given a consistent testing regimen (i.e., test cases or hours of testing).

Compared with the non-cumulative graph of defect rates in Figure 5-8, the cumulative graph smooths out much of the variation in actual defect data. The monthly defect discovery rates will be highly variable. Curve fitting with only a few data points can produce unreliable forecasts, and these monthly reports will not inspire confidence. The curve is not intended to forecast monthly rates over the course of development. Instead, it is used to forecast long-term trends, with forecasts and cumulative actuals converging over time.



Source: OUSD(R&E) Software Team

Figure 5-8. Weibull/Rayleigh Curve Models Defect Rates

The idealized model is based on assumptions about constant rates of testing effort, stable development, etc. but has proven robust in practice even when those assumptions are violated (Kan 2003).

Monthly peaks and valleys usually can be traced to an event in the program. Peaks may be the result of inspections and reviews, or software integration and test. Valleys may be the result of lowered testing activity which may occur over a holiday or vacation season. Such variations are smoothed out in a cumulative chart of the data but prominent in time series of monthly discovery rates.

5.11.3 Tips on preparing the data

One advantage of Weibull analysis is that it relies on objective data that is typically available for any professionally managed software development effort. Defect data is routinely collected in automated bug tracking software that developers and testers use to report and monitor defects for their own internal management purposes. Using this existing data for maturity assessment requires marginally little effort once the underlying assumptions have been validated.

The data required for analysis is a time series of defect arrivals. The raw event dates of the individual defect reports are binned into convenient time frames such as weekly or monthly. The counts in these time frames can be plotted to create a “cumulative” or “arrival rates” plot.

Programs will often report defects in terms of severity and priority. Severity is often linked to the mission impact of the defect. Priority is linked to decisions to assign resources to address the defect. The two are often but not necessarily correlated.

Weibull analysis can also be performed for the subset of severe or high-priority defects, to forecast when software might be releasable.

5.11.4 Applicability to CI/CD Pipeline

The Weibull/Rayleigh analysis presented here has been applied to conventional software development efforts. How well does this approach apply to CI/CD pipelines? One might expect that the small batch sizes and continuous testing would flatten out the defect curve into a steady state. Perhaps, individual batches experience reliability growth curves that fit the Weibull/Rayleigh model, but these variations are smoothed out as the batches are staggered in time. As of this writing, there remains a lack of empirical data to validate or refute these suppositions. This suggests that programs should collect their own data and determine for themselves whether the patterns observed conform to this model in ways that are helpful in the context of their effort.

6 Software Engineering and Workforce Competencies

- DoD must compete with national and global industry for digital talent.
- Successfully executing a software effort requires the ability to staff the effort quickly with qualified software professionals, including those requiring clearances.
- The modern software factory requires features such as automated and continuous testing and advanced AI/ML capability.

DoD must identify, hire, and train professionals capable of developing software for modern warfighting. Changes in software technology require changing competencies, knowledge, skills, and abilities. As DoD competes for the same digital talent as many large companies nationwide and worldwide, identifying the correct qualifications is essential for developing an effective software engineering workforce.

The Under Secretary of Defense for Personnel and Readiness (USD(P&R)) is granted the authority to establish and implement policy, establish procedures, provide guidelines and model programs, delegate authority, and assign responsibilities regarding civilian personnel management within the DoD (DoDD 5142.02 2008). Under this authority USD(P&R) has established the Five-Tired Competency Framework as the basis for a competency-based approach to personnel management in DoD.

OUSD(R&E) commissioned a series of studies by the RAND Corporation to clarify the skills and competencies the Department needs to modernize its software engineering processes. The report *Software Acquisition Workforce Initiative for the Department of Defense* (Robson, et al. 2020) codified recommendations for 48 workforce competencies spanning software engineering, cybersecurity, artificial intelligence, embedded systems, and more.

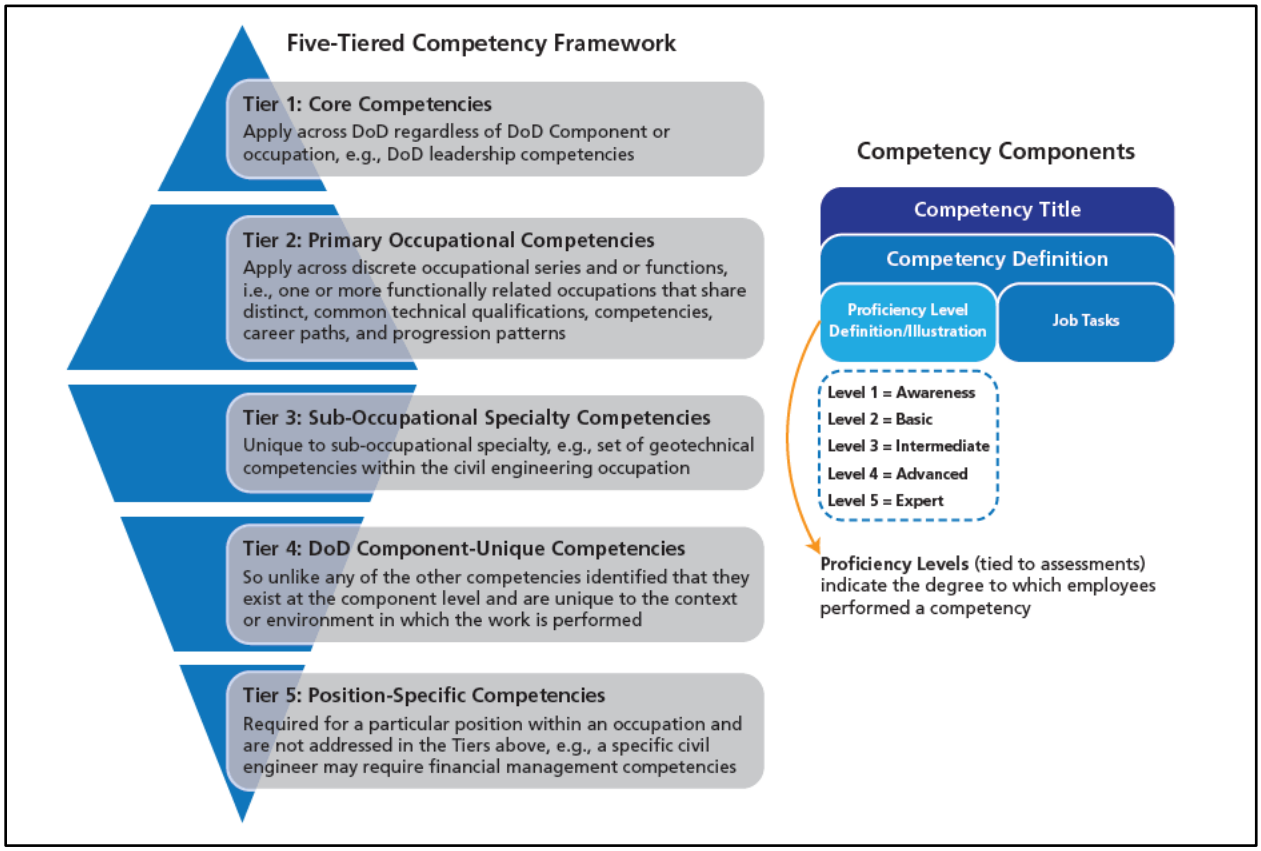
This section discusses a definition of DoD "competency," summarizes the RAND study findings regarding software development, and highlights competencies the DoD requires in the defense software workforce and software factory. The section also summarizes results as of early 2023 of DoD working groups such as the DoD Digital Talent Management Forum, as well as recent innovations such as the Defense Cyber Workforce Framework (DCWF) that advance the cause of shaping the DoD workforce to meet present and future needs.

6.1 DoD Five-Tiered Competency Framework

The Defense Civilian Personnel Advisory Service (DCPAS) (DoDI 1400.25 Volume 250 2016) defines "competency" as an observable, measurable pattern of knowledge, skills, abilities, behaviors, and other characteristics (KSAOs) needed to perform work roles or occupational functions successfully.

Figure 6-1 shows a DoD-wide Five-Tiered Competency Framework promulgated in policy by the USD(P&R). Any software engineering and workforce competencies identified must align with this framework to be implementable within the existing DoD personnel processes and governance structures.

There is no single approach to developing competency models, but DCPAS and other organizations follow a few steps, including a thorough review of existing data, drafting an initial model, gathering inputs from SMEs, refining the model, and validating the model.



Source: DoDI 1400.25 vol 250 June 2006

Figure 6-1. Five-Tiered Competency Framework

Tier 1 focuses on core competencies that apply across DoD and are not specific to a position or agency. An example of Tier 1 competency could be "demonstrates integrity."

Tier 2 competencies apply across an occupational series; for example, "cybersecurity" could be applied across all of IT.

Tier 3 competencies focus on KSAOs specific to subspecialties that may exist in one or more occupational series; for example, "software assurance" could be considered a sub-occupational competency for a specialty within IT.

Tier 4 adds further detail to components and agencies, for example, competencies required to work at the Air Force Sustainment Center.

Tier 5 competencies are meant to capture any additional KSAOs needed for a specific position that are not already addressed by Tiers 1 to 4.

6.2 RAND Software Competency Study

The RAND study (Robson, et al. 2020) identified 48 software acquisition competencies that DoD needs in order to plan and execute modern software development as practiced in the commercial sector. Identifying, prioritizing, and managing the acquisition, development, training, and retention of those competencies is critical to developing and delivering the software-enabled warfighting capability needed to perform the DoD mission.

The following 48 software engineering competencies are wide-ranging and described in terms of DoD work activities and tasks. Critical software engineering competencies and definitions for DoD software acquisition professionals supporting the pathway include those listed in Table 6-1 and discussed in the following sections.

Table 6-1. DoD Software Acquisition Workforce Competencies (RAND Study)

RAND Software Competencies and Topics	
Problem Identification	1. Capabilities Elicitation
	2. Business Case Development
Solution Identification	3. Strategic Risk/Reward Analysis
	4. Cloud Computing
	5. Software Ecosystems
	6. Model-Based Engineering
Development Planning	7. Development Tempo
	8. Release Planning
	9. Software Development Planning
	10 Planning for Continuous Delivery
	11. Planning for Continuous Deployment
	12. System Engineering Planning
	13. Software Metrics
	14. Configuration and Version Control
Transition and Sustainment Planning	15. Software Documentation
	16. Contracting for Software Development
	17. Data & Proprietary Rights Management
System Architecture Design	18. Architecture Design Approach
	19. Software Orchestration & Choreography Patterns
	20. Software Deployment Patterns
	21. Artificial Intelligence and Machine Learning Applications
	22. Augmented & Virtual Reality Applications
	23. Embedded Systems
	24. Balancing Quality Attributes
	25. Emerging Technologies
Modeling Functional Capabilities & Quality Attributes	26. Use and abuse case Modeling
	27. Validation of Performance Requirements
	28. Validation of Sustainability Requirements

RAND Software Competencies and Topics	
	29. High-Fidelity System Modeling
Building Secure, Safe & High-Availability Systems	30. Software Assurance
	31. Cybersecurity
	32. Safety Critical Systems
	33. High-availability Systems
Software Construction Management	34. Life-Cycle Management
	35. Detailed Backlog Management
	36. Release Management
	37. Change Management
	38. Automated Test & Continuous Integration
Software Program Management	39. Effort Estimation
	40. Product Roadmap & Schedule Management
	41. Cost Management
	42. Legal Policy and Regulation
	43. Risk, Issues, & Opportunity Management
Mission Assurance	44. Quality Assurance
	45. Root Cause, Corrective Action
	46. System Integration & Testing Professional Competencies
Professional Competencies	47. Strategic Planning and Change Management
	48. Innovation and Entrepreneurship

Source: Robson et al. 2020

The software engineering competencies are intended to augment but not replace any existing DoD competencies for acquisition; e.g., contract management, program / project management, systems engineering, mission assurance.

Problem Identification:

(1) Capabilities Elicitation – Engage with stakeholders (to include representative end user organizations, owners, developers, integrators, certification authorities, independent validation and verification personnel, and operators) to elicit capability objectives (i.e., functional requirements) and quality attributes (i.e., non-functional requirements) for the proposed system.

(2) Business Case Development – Explore the problem space and identify focal areas for acquisition.

Solution Identification:

(3) Strategic Risk / Reward Analysis – Evaluate and balance risk/reward from various stakeholder perspectives, including the sponsoring organization, end users, test and evaluation teams, cybersecurity compliance officers, and data rights managers.

(4) Cloud Computing – Identify resources needed to operate and sustain DoD unique Cloud platforms

(5) Software Ecosystems – Employ existing and emerging DoD, open source or third-party tech to support shared resources

(6) Model Based Software Engineering – Create a digital environment that uses high fidelity hardware and software in the loop models, prototyping, visualization, simulation, and dependency analysis

Development Planning:

(7) Development Tempo –- Determine the software life-cycle approach to be used and the tempo of software construction, release, and deployment to operations.

(8) Release Planning – Determine the MVP or MVCR and definition of "done" for each release.

(9) Software Development Planning – Apply methods, processes and training needed for software construction (design, code, test, build, build, integrate, release). Identify tools and methods for backlog management, continuous integration, automated regression testing, and release management.

(10) Planning for Continuous Delivery – Identify methods (e.g., DevSecOps), tools, processes, and training for automating the software release process.

(11) Planning for Continuous Deployment – Identify the software that could benefit from rapid delivery into operations.

(12) Systems Engineering Planning – Develop methods, processes, and training that align to the software development life cycle, tempo, and release plans.

(13) Software Metrics – Select appropriate metrics and measures at the team, program, and stakeholder level to monitor software scope, cost, schedule, and quality.

(14) Configuration and Version Control – Develop strategies for identifying and managing the configuration of the system and software development and test environment.

Transition and Sustainment Planning:

(15) Software Documentation – Document software planning, requirements, design, code, validation, verification, and sustainment needs in the program planning.

(16) Contracting for Software Development – Ensure that contract requirements, constraints, end items, and data deliverables are compatible with the selected tempo, release planning, software and system development planning, metrics, and documentation requirements.

(17) Data and Proprietary Rights Management – Negotiate data rights up front if elements of the software or system will be acquired from DoD-external sources (i.e., open source repositories, commercial-off-the-shelf (COTS) software, GOTS software, or from private entities) to ensure DoD will have assured access to all mission-critical software throughout the life of the supported system. Ensure that all software licenses are in compliance with federal regulations and compatible with program needs..

System Architecture Design:

(18) Architectural Design Approaches – Determine "how much" architectural design effort is needed to ensure a successful acquisition. Consider benefits and risks of adapting practices from modern architectural design methods such as Artifact Driven, Use/Abuse Case Driven, Attribute Driven, Domain Driven (i.e., Manage by Architecture), or Human-Centered Design when selecting an architectural design approach.

(19) Software Orchestration and Choreography Patterns – Determine the patterns the software will use and consider common orchestration and choreography patterns (e.g., client/server,

publish/subscribe, peer-to-peer, and services/ microservices) that balance quality attributes for timing performance (latency, throughput), safety and security.

(20) Software Deployment Patterns – Determine how the software will be deployed onto the computing infrastructure in the operational system.

(21) Artificial Intelligence and Machine Learning Applications – Identify and implement architectural components, methods, processes, and training of incorporating artificial intelligence and machine-learning techniques to create autonomous cyber-physical systems, automated or augmented decision support tools, or other emerging AI based systems.

(22) Augmented and Virtual Reality Applications – Identify and implement architectural methods and processes that balance correctness and safety in augmented VR applications.

(23) Embedded Systems – Employ explicit strategies for incremental realization of capabilities within the constraints of the hardware supply chain.

(24) Balancing Quality Attributes –Evaluate alternative design solutions and architectures to effectively balance the quality attributes for critical mission threads or other identified scenarios.

(25) Emerging Technologies –- Maintain an understanding of emerging technologies, the implications these technologies may have on a given organizational need and solution space.

Modeling Functional Capabilities and Quality Attributes:

(26) Use / Abuse Case Modeling – Use static and dynamic views to model the software components that implement the required capabilities of the software to identify the use cases.

(27) Validation of Performance Efficiency Requirements – Validate the capability to meet performance efficiency requirements (with margin as appropriate to the life-cycle phase) under realizable nominal, best, and worst-case conditions for each mission-critical thread.

(28) Validation of Sustainability Requirements – Validate sustainability features of the software architecture with consideration for specific needs associated with high availability and safety-critical systems.

(29) High Fidelity System Modeling – Create a digital, high-fidelity representation of the as-built system that reflects lessons learned in test or operations to support the analysis of critical quality attributes.

Building Secure, Safe and High Availability Systems:

(30) Software Assurance – Determine appropriate coding standards, static and dynamic analysis rules, test code coverage, and fuzz testing standards needed to ensure the integrity of the acquired software.

(31) Cybersecurity – Identify and continuously evaluate the key security components of the architecture (such as Zero Trust, STIGs, whitelists, audit traces, and multilevel security guards), and specify the methods and processes that will be used to assure their integrity throughout the program life cycle

(32) Safety Critical Systems – Provide technical analysis relevant to safety-critical systems (e.g., aircraft, nuclear systems, ground combat systems, missile systems, space systems) or

portions of systems (e.g., deployment mechanisms that interface with live ordnance), apply available best practices or required standards such as Radio Committee for Aeronautics (RTCA) standard DO 178C (Software Considerations in Airborne Systems and Equipment Certification), and Military Standard (MIL-STD) 882E (Department of Defense Standard Practice for System Safety) and successors to increase the safety of operational software.

(33) High Availability Systems – Establish service-level indicators to measure reliability/stability of the software and system from the user perspective. This metric should be over time and include, for example, identifying user-defined mission-critical threads and stressing test cases such as max load in off-nominal conditions. A good attribute is to have actual users demonstrate their standard operating procedures.

Software Construction Management:

(34) Life Cycle Management – Update plans to address obsolete or emerging technologies, methods, processes, and tools. Identify timing, content, and stakeholders for retrospective reviews.

(35) Backlog Management – Develop and maintain a list of capabilities (the product backlog) and the tasks that are required to realize those capabilities mapped to the release plan.

(36) Release Management – Synchronize software releases with the development of models, simulations, test beds, and operations environment(s) as needed to ensure compatibility. Use the "done" criteria from the release planning to identify the required verification steps (inspection, analysis, unit, integration, or acceptance test) for each release to higher levels of integration testing, certification activities, and/or operations.

(37) Change Management – Implement mechanisms to ensure that decisions regarding proposed and approved changes are communicated clearly to all stakeholders for the program planning, requirements, architectural design decisions, code, as well as validation and verification artifacts.

(38) Automated Test and Continuous Integration – Automate the tests (from unit tests to system integration tests) when feasible to allow for rapid discovery of integration issues. Identify a subset of the test to function as a "smoke test" for daily or on-demand builds of the software.

Software Program Management:

(39) Software Effort Estimation – Create and maintain an estimate of the total software acquisition effort (labor and material), accounting for software size, complexity, precedent, team cohesion, and the development team's direct experience. Use parametric, historical comparisons (analogies) and bottom-up effort estimates from the development team, as appropriate, to support business case development and acquisition strategy refinement. Revise the acquisition strategy accordingly.

(40) Product Roadmap and Schedule Management – Timebox releases to provide structure to your roadmap and "fix" time/schedule (the questions becomes the amount of value that can be delivered within the timebox). Implement plans for capability/feature development and release (the product roadmap) and monitor velocity of software production.

(41) Cost Management – Dedicate teams and map them to capabilities or underlying Epics/Features to the greatest extent possible to "fix" costs (this allows you to understand

the cost per capability for investment decision-making). Monitor actual software production metrics versus labor and material expenditures, and update effort estimates and cost baselines as needed.

(42) Legal Policy and Regulatory Environment Management – Understand and adhere to relevant laws, congressional budgets (fiscal year funding constraints), regulations and certification requirements, and policies (e.g., data rights, export rules).

(43) Risk, Issues and Opportunity Management – Implement and manage a closed-loop process to actively track risks and issues as they arise, identify opportunities for improving products and processes that add to the value for the user, and continuously reassess program plans to mitigate risks and realize opportunities.

Mission Assurance:

(44) Quality Assurance – Establish criteria for reviewing and auditing the software supply chain across all sub tiers as necessary to ensure program success.

(45) Root Cause Corrective Action – Monitor the program and software metrics to identify early indicators of adverse trends, defects and technical debt and determine root causes. Use statistical control or other methods to proactively propose changes.

(46) System Integration and Testing – Automate integration and test activities to the fullest extent practical and build them into the software release process.

Professional Competencies:

(47) Strategic Planning and Change Management – Take a long-term view and build a shared vision with others, act as a catalyst for organizational and cultural change. Influence others to translate strategic planning into action.

(48) Innovation and Entrepreneurship – Provide transformational solution-based approaches to problem solving and building products by employing an iterative process to empathize, define, ideate, build/prototype, and test (i.e., design thinking); and institute a culture that encourages continuous learning and innovation.

6.3 Agile/DevSecOps Software Factory

A DoD Agile/DevSecOps software factory (DoD CIO DSOERDK 2021) includes people, processes, and tools. The software factory should include the following features for best results:

- An Agile/DevSecOps software development and orchestration pipeline, using continuous integration and continuous deployment tools and techniques.
- Software architecture designs using Cloud-native microservices and automated tools.
- Software estimation, software measures, and automated metrics generation.
- Software development using automated and continuous testing.

- Software assurance, cybersecurity, and site reliability engineering.
- Machine learning, artificial intelligence, and the pervasive use of automation.

6.4 Organizational Competency Needs

Service components and agencies should form, organize, optimize, and continuously improve their program software engineering Government and contractor workforce. They should focus on the people, culture, and team cohesion, and create a constructive Government and contractor working environment.

Following are example position titles within a PMO using an Agile/DevSecOps software factory. All the aforementioned competencies should be organic across these software acquisition positions.

- Product Manager
- Product Owner
- Product Designer (user research, UX, UI, visual design)
- Software Engineer
- Software Developer
- Software Quality Engineer
- Safety Engineer
- Architect
- Platform Engineer
- IT Engineer
- Security Engineer
- Data Scientist
- Data Engineer
- Quality Engineer

Following are items to consider in refining competency requirements:

- What percentage of my staff has experience working on a DevSecOps project?
- What training and credentialing are available to develop and verify competencies?

6.5 DoD Digital Talent Management Forum

After publication of the RAND Report, OUSD(R&E) established the DoD Digital Talent Management Forum (DTMF) in accordance with Section 230 of the NDAA (FY 2020): "Policy on the Talent Management of Digital Expertise and Software Professionals." OUSD(R&E) and OUSD(A&S) co-lead the forum, which focuses on understanding how Components are managing digital talent. The DTMF absorbed two former Chief Digital and Artificial Intelligence Office (CDAO) groups (Section 8.2), the Artificial Intelligence (AI) Workforce Subcommittee and the Data Talent and Culture Working Group, to form an expanded DTMF. The DTMF includes members from 42 DoD organizations and meets monthly to exchange useful information and lessons learned.

6.6 DoD Cyber Workforce Framework

The Department developed the DoD Cyber Workforce Framework (DCWF) to provide a standardized way to describe cyber work for military, civilian, and contractor personnel and to support talent management for these activities. Office of the DoD Deputy CIO for Resources and Analysis, Cyber Workforce Directorate manages the DCWF. The framework is the authoritative reference for identifying, tracking, and reporting DoD cyber positions, including a coding structure for authoritative manpower and personnel systems, pursuant to DoDD 8140.01 (2020).

The warfighting domain continues to evolve in threat and complexity. Talent, and supporting workforce management practices, must then continue to evolve to address the ever-changing landscape posed by our adversaries to meet the strategic mission requirements of tomorrow. As of this publication, the DCWF has expanded from the 54 original work roles to 65 roles, with the inclusion of AI and Data & Analytics.

The DCWF approved software work roles are as follows:

- (621) Software Developer (update)
- (628) (New) Software/Cloud Architect
- (461) Systems Security Analyst (update)
- (627) (New) DevSecOps Specialist
- (625) (New) Product Designer User Interface (UI)
- (626) (New) Service Designer User Experience (UX)

- (806) (New) Product manager
- (673) (New) Software Test & Evaluation Specialist

The DCWF tool, https://public.cyber.mil/cw/dcwf/ (DCWF 2022) provides access to DoD's authoritative lexicon based on the work an individual is performing, not their position titles, occupational series, or designator. The searchable tool includes a public and a CAC-enabled version. For more information, view the DCWF Orientation Training video at https://public.cyber.mil/training/dcwf-orientation/ (DoD CIO 2022).

7 Contracting for Software Engineering in DoD

- The Software Acquisition pathway is designed to enable modern software practices, but programs in the traditional acquisition system also have successfully tailored contracts to accommodate new software methods.
- The Agile approach allows "real-time" visibility into the project status.

This section provides information to help DoD acquisition professionals navigate the contracting process and remove obstacles to modern software engineering practices. This section discusses traditional systems and recent innovations in the AAF governing DoD acquisitions. This section also identifies best practices for selecting and tailoring contract vehicles to support the software engineering processes that enable continuous delivery.

Although the Software Acquisition pathway is designed to enable modern software practices, Program Managers have succeeded in tailoring software development agreements supportive of those practices in the context of the traditional system. They report that the acquisition system is more tailorable and flexible than many realize, with proper understanding of the processes and how to adapt the agreements and contracting vehicles (DIB 2019b).

The DAU website on the DoD AAF (https://aaf.dau.edu/aaf/software/contracting-strategy) provides additional guidance on contracting for software engineering in DoD.

7.1 Agile and DevSecOps Software Development Contracting

Planners should keep in mind the following goals for any software development contract:

1. Define the purpose of the project (i.e., what are the parties trying to accomplish).
2. Define how the project is to be established and managed.
3. Define what happens if the project fails to meet its objectives.
4. Define the MVP/MVCR.
5. Define project completion.
6. Structure incentives so contractors do better when DoD gets what it needs.

Contracts for conventional Waterfall approaches have been criticized for focusing too heavily on what happens in a project when the effort diverges from detailed plans and schedules. Often these plans are mapped out far in advance over long time frames. This adversarial approach to

accountability often overlooks the importance of common understanding and essential to cooperation on the first two goals.

In contrast, Agile approaches focus on shared goals and clearly defined roles and responsibilities for continual cooperation and engagement. Sometimes termed "rules of engagement," this management structure helps the contracting parties cooperate with the flexibility to creatively achieve shared goals. As an example, the Scrum model sets out clear requirements for each of the following aspects of the project:

- The key project roles (e.g., Product Owner, development team, Scrum Master) are clearly defined up front.
    - The product owner is typically a Government role so they can guide team priorities. The Scrum Master can be either a Government or contractor role.
    - The Team should be dedicated to delivery of specific capabilities/features.
- The key planning and management meetings or "ceremonies" (e.g., grooming the product backlog, planning the sprint backlog, demonstration of releasable value, and retrospectives to continuously improve) are preset and defined.
- The key project documentation (e.g., Product Vision, Product Roadmap, product backlog, sprint backlog, sprint backlog burndown chart) are clearly defined.

The advantage of the Agile approach is that it promotes "real-time" visibility into progress, issues, and control of the project by the stakeholders, as opposed to periodic updates that may occur weeks or months apart (e.g., during typical Program Management Reviews (PMR) where issues may be communicated months after the fact, creating costly rework). Near-term and routine delivery of working product allows customers or end users to provide meaningful feedback to guide future effort, and each release generates data that can be used for decision-making.

7.2 Contract Types

Contract types often follow traditional approaches to purchasing contracts in which the buyers may outsource complex development to suppliers who can build systems with the desired capabilities. Contract types include firm fixed price (FFP), time and materials (T&M), cost plus, or target price, among others. Table 7-1 characterizes the basic contracting types as well as the potential risks associated with the various models.

Table 7-1. Contracting Types

Firm Fixed Price (FFP)	Time and Materials (T&M)	Cost Plus	Target Price
• Fixed specification • Fixed price and date • Changes with a fee • Risk to Supplier	• No complete specification • Price based on rate • Ends as specified by customer • Risk shifted to customer	• Target specification • Target date • Customer pays Supplier's cost-plus profit margin • Risk mostly shifted to customer	• Fixed specification • Fixed date • Target price • Negotiated profit for the Supplier above the target price. • Shared risk, shared economic opportunity

7.3 Contracting Maturity Models

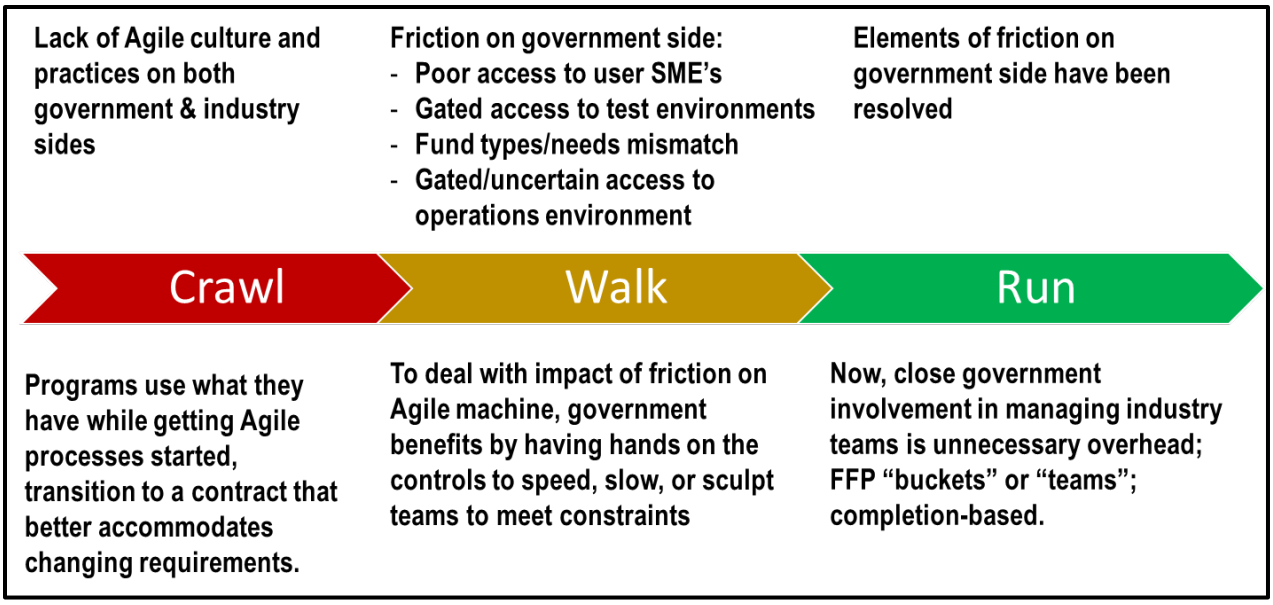
The transition from legacy Waterfall to more Agile practices does not happen overnight. It happens incrementally to substitute new practices for old. It is useful to view the transition on two levels:

- Agile contracting level: contracts and formal agreements in place between the suppliers and the Government.
- Agile development level: software engineering practices, and practices that the developers and the programs follow to build the software.

If a program attempts to transition to Agile development without complementary transition to Agile contracting, the resulting friction will inhibit progress.

7.3.1 Air Force Contracting Maturity Model

The Air Force Agile contracting maturity model shows how programs may expect to mature their contracting practices. The stages are described as crawl, walk, and run. Figure 7-1 shows some characteristics of each Agile contracting maturity stage and comments on how contracts and program office functions may evolve as programs mature through these stages.



Source: Derived from DoD DevSecOps CoP

Figure 7-1. Contracting Practices Maturity

To get started, both Government and industry must use the contracting vehicles that are in place at the time. As their understanding grows, they transition to contracting terms that better accommodate the flexibility and create incentives for a more collaborative Agile approach. As the culture shifts, some friction can be expected in areas such a continuous access to user subject matter experts, restrictions on funding, and continuous availability of and access to test facilities. As these sources of friction are eliminated out over time an integrated team emerges.

7.4 Agile Software Development using Scrum

Agile software delivery contracts may use Scrum methodology and terminology. The Scrum framework serves as an example. Many frameworks may be termed "Agile" (e.g., Kanban, Extreme Programming, Lean Agile, DevOps), but this guide refers to the Scrum framework, which a program can tailor with other frameworks as appropriate.

The following sections cover considerations in the Agile contracting process:

- Key Roles
  - Product Owner
  - Development Team
  - Scrum Master
- Product Vision
- Product Roadmap

- Product Backlog
- Sprint Process
  - Duration
  - Sprint Meetings
  - Starting the Next Sprint
  - Definition of "Done"
- Project Completion
- Pricing
- Warranties and Indemnities
- Composition of the Development Team
- Termination
- Intellectual Property (IP) Rights
- Dispute Resolution

7.5 Roles and Responsibilities

7.5.1 The Product Owner

The **product owner**[2] is the primary representative or "voice" of the **customer**[3] and is responsible for establishing feedback loops with customers/end users, understanding distinct customer/end user segments, and ensuring clear definition of their needs. The product owner then communicates the customer's vision, requirements, and project to the **development team.** The product owner also assumes the primary responsibility for the product backlog, including its initial development and its ongoing grooming over time as well as participation in meetings with the development team during each **sprint,** including assessing development items.

[2] Terms that have a specific meaning in the Scrum framework appear in **boldface** on first occurrence.
[3] In the context of planning, the Scrum term "customer" refers primarily to the operational user or end user. At other times, the term may also encompass the acquisition program acting on behalf of the eventual end user. In DoD the customer will generally be the acquiring command or component, not necessarily the operational user of the software.

Key points to address:

- The product owner is a representative of the customer (typically the Government) and as such, the contract should provide for the product owner to be identified by agreements between the operational user and the program before program execution.
- The product owner must have authority to adapt the product as needed. If they have to check in with a committee or another leader, you do not have a product owner.
- The supplier may seek some assurance from the customer that the nominated product owner is suitably experienced in Scrum development projects or has undergone specific training to acquire the requisite skills.

The contract should set out the key responsibilities of the product owner which include:

- Ownership of the content Product Vision, Roadmap, and Product Backlog and authority to prioritize work within those items as they see fit.
- Identification of the MVP and MVCR.
- Ongoing revision and re-prioritization of the product backlog as the project develops.
- Participation as the "voice of the customer" in the relevant sprint planning and review meetings.

7.5.2 The Development Team

The development team is responsible for the actual development activities within each sprint. As such, the team needs to be cross-functional and include members who are skilled in areas such as coding as well as testing, etc. The development team should be experienced in Agile development projects.

The program must decide whether the development team should be composed of only the supplier personnel or whether it should also include Government personnel. However, this approach may be problematic in practice for reasons such as the following:

- The customer may not have sufficient resources to dedicate to the project on a day-to-day basis.
- The customer may not have personnel with the necessary technical skills to participate effectively in the development team.
- From a legal perspective, a combined development team raises difficulties in establishing a clear allocation of risk and liability between the customer and the supplier.

Key points to address:

- Team Composition: During the initial software working group meetings after contract award, managers and engineers should discuss the assignment of personnel to the team. Ideally, the contract already addresses team roles and customer participation in the team. On larger projects, personnel from each CSCI and stakeholders should be included as appropriate.
- Software Verification and Validation: Who will verify that the software provides the agreed capabilities, and how will they perform this verification?
- Software Reliability: What are the reliability requirements? What metrics will the team use to quantify reliability, and what targets must the system achieve or maintain over time.
- Intellectual Property Rights: Who owns the rights to the software produced? Who owns the rights to the build environment needed to build the software from source? It is preferable for the Government to have rights to both the source code of the system and to the libraries and toolchains that provide the capability to build the software from source. Use of DoD enterprise software factory assets is one way to achieve this goal and avoid vendor lock.

7.5.3 The Scrum Master

The role of the scrum master is perhaps the most challenging to capture in the contract. In broad terms, the scrum master's role is like that of a coach or a mentor – ensuring that the development team and the product owner are working co-operatively and following the Scrum processes. More importantly, the scrum master is not a project manager. His or her role is not to assign tasks and measure progress against goals but rather to support the product owner and the development team. The scrum master may be from the supplier, the Government, or provided by a third party.

Where the parties are relatively new to Scrum projects and the scrum master's identity will depend on a range of factors that include:

- Whether the customer has personnel available and with the right skill set to act as the scrum master.
- Whether an external consultant will be able to build the necessary relationships with the product owner and the development team to act as an effective scrum master (and whether the project budget can justify an additional external resource).

Key points to address:

- The contract should identify the proposed scrum master (or include a process for the scrum master to be agreed upon by the parties). The contract should also set out the level of skill, experience, and qualifications required of any scrum master (such as years of experience on software development projects using the Scrum methodology).
- The key responsibilities of the scrum master should be set out in the contract.
- The contract should require that the scrum master does the following:
  - Is dedicated to the project during the development period (unless otherwise agreed).
  - Is not re-assigned from the project without the prior written consent of the Program.

7.6 Product Vision

The starting point for any Agile product is the product vision. This is a statement setting out the overarching goals of the project and the high-level benefits that are sought. Ideally, the product vision will have been developed before the contract negotiations start to help the negotiation team understand the intended result of the project.

Key points to address:

- The product vision should be included in an appendix to the contract as a reference point for the development of the product backlog and the project.
- In the parlance of the Software Acquisition Pathway, the CNS takes on the function of the product vision.
- All required Agile ceremonies should be defined in the contract (e.g., backlog grooming, sprint planning, daily standups, product demonstrations and retrospectives).

7.7 Product Roadmap

A product roadmap is a tool owned by the product owner that illustrates high-level, adaptable value delivery targets over a designated period. Work items on the product roadmap deliver stand-alone value to stakeholders along the release cadence leveraged by the team. It tells what will be delivered and when but leaves the how up to the trusted and empowered team.

Key points to address:

- The product roadmap should be traceable both to the product vision, required capabilities (like a CNS or requirement document) and to the product backlog.
- The product roadmap should address capability in terms that stakeholders can understand on which they can place a value.

- The product roadmap should address early delivery of the MVP and the MVCR
- The product roadmap is best supported by automated tools (e.g., Jira) that maintain a linkage between work items on the backlog delivered over short time intervals (hours, days) that roll up into capabilities delivered over a longer interval (weeks, months).
- The product roadmap is a basis for ongoing coordination between the product owner, development teams, end users, and other stakeholders over the course of the effort.
- The product roadmap should be updated frequently to reflect any changes in funding and mission.

7.8 Product Backlog

The product backlog is the Agile equivalent of a "Statement of Requirements." It essentially refers to a prioritized list of items that are to be developed up to the next release. It will continue to be burned down as requirements are completed and repopulated as new work is elaborated. One way to imagine the product backlog is to picture it as a "stack" of development items ordered by importance, with the highest priority items at the top of the stack and the lowest at the bottom.

The product backlog will be traceable to the product roadmap and product vision, and should include the following elements:

- Items. The list of epics, features, and/or stories to be developed. As the project develops, these may also include defects to be rectified and/or areas for further improvement.
- Estimate of Value. An estimate by the product owner of the value to the customer's mission of each item presented in relative terms by comparison to other items.
- Estimate of Relative Complexity. An estimate of the complexity each item presented in relative terms by comparison to other items (typically via story points.)[4]
- Priority. The priority for each item, taking account of the estimates of mission value and complexity.

In Agile projects, the development items may be articulated as "User Stories" that capture succinctly what the end user wants to achieve. The typical format for the User Story is as follows:

[4] The term "relative complexity" appears in place more traditional term "effort" in order to drive the team away from a focus on time commitments that lead to micromanaging. Instead of saying "8 hours" or "3 days" we focus on relative complexity measures that may be arrived at methodically (e.g., via agile planning poker) measured in story points.

"As a <describe end user role>, I want <describe feature or goal> so that <describe reason or benefit>". The intended purpose of this format is to keep the description of the development items short and clear. Additional features of the product backlog are:

- The high priority items should be clearly defined (by contrast, the low priority items may be more general or vague).
- Items can be re-prioritized by the product owner at any time.
- New items can be added by the product owner from time to time (and prioritized as necessary).
- Items can also be removed from the product backlog by the product owner at any time.

OUSD(R&E) Lesson Learned. It is common for the product owner and the development team to devote approximately 1-2 hours of each sprint to grooming and refining the product backlog, including the preparation of more detailed explanations of individual items, acceptance criteria, relative complexity, or dividing more general items into smaller and more specific items.

Key points to address in the contract include the following:

- The contract will need to specify how the initial version of the product backlog will be developed. There are several possible approaches that include:
    - The initial product backlog could be developed in parallel with the negotiation of the contract, in which case, one should *not* attach it to the contract as it will have the negative effect of locking in scope and/or schedule based on scope.
    - The product backlog could be developed following contract signature. In this case, the contract could provide for an initial 1-2 day workshop to be held between the product owner and the development team to discuss the development of the product backlog and, where necessary, carry out some detailed requirements analysis. The result should be a product backlog that contains at minimum all work required to complete the initial release of MVP.
- Once the initial version of the product backlog has been developed, in the first backlog grooming session, the development team provides the product owner with an estimate of the relative complexity required to develop each item in the product backlog. It is important that the team relatively estimates all of the work in the product backlog to manage and forecast progress toward the upcoming release.
- The contract should require that these estimates are prepared with appropriate care and skill, and based on fair and reasonable assumptions. It may also be appropriate for this process to be subject to the dispute resolution procedure in the contract where there is a dispute between the parties as to whether the estimates from the development team are appropriate.

- Once the estimates of relative complexity have been finalized, the contract should require the product owner assign a priority to each item based on the estimates of relative complexity and mission value.
- All ceremonies, including release planning, backlog grooming, sprint planning, scrums/standups, sprint demos, and sprint retrospectives should be addressed in the contract to ensure appropriate resourcing by all parties.
- The contract should also make it very clear that the product owner is free to amend the product backlog at any time. The exceptions to this are the following:
    - The product owner cannot change relative complexity estimates provided by the team(although they may be discussed during the workshops as discussed above).
    - The product owner cannot introduce new work during a sprint or change the scope of work committed to for the sprint unless there is an emergency (i.e., the product stopped working for all customers and not "my leadership really wants this story done now").

7.9 Sprint Process

7.9.1 Duration

The parties are free to choose the duration of sprints that will support the project, although these should be kept relatively short (e.g., 2-4 weeks with a preference to the shorter time frame). It is a key principle of Scrum that the duration of individual sprints should not be changed, even if the progress is running behind schedule. Unfinished items should instead be re-inserted into the product backlog and then reprioritized accordingly.

Key points addressed in the contract include the following:

- The contract should specify the agreed upon duration of each sprint.
- The contract should also include an acknowledgement from both parties that the duration of an individual sprint cannot be extended.

7.9.2 Meetings

Each sprint cycle will typically feature five ceremonies: Backlog Grooming, Sprint Planning, Daily Scrums, Sprint Review/Demo, and a Retrospective.

Ceremony	Frequency	Timing	Participants	Goals
Backlog Grooming	Every 2 weeks (60 minutes)	First day of each Sprint	PO, SM, All Team Members	▪ Ensure all known work is expressed as value (not tasks) ▪ Add, edit, delete until all known Stories are captured ▪ Prioritize product backlog ▪ Relatively estimate complexity of all Stories
Sprint Planning	Every 2 weeks (60 minutes)	First day of each Sprint, after Backlog Grooming	PO, SM, All Team Members	▪ Set Sprint goal ▪ Estimate work to complete in the upcoming Sprint

Daily Scrum	Daily (15 minutes)	Daily during each Sprint	SM, All Team Members	▪ Teams meet in-person (or with cameras on virtually) to make commitments to each other
				▪ NOT a forum to discuss tasks/steps toward value delivery
				▪ SM asks each team member:
				  ▪ What Story did you complete since we last met?
				  ▪ What Story are you working on currently?
				  ▪ Are you blocked by anything?

Sprint Review (Demo)	Every 2 Weeks (60 minutes)	Last day of each Sprint, before the Retrospective	PO, SM, All Team Members, Stakeholder Reps	▪ Team members demo completed work and discuss the increment ▪ PO and Stakeholders ask questions, give feedback ▪ Work is accepted/rejected by PO
Retrospective	Every 2 weeks (60 minutes)	Last day of each Sprint	PO, SM, All Team Members	▪ Continuous improvement to enhance team performance (what team should continue doing, stop doing/fix, and what we haven't tried that we should experiment with)

Source: The MITRE Corporation

Figure 7-2. Scrum Ceremonies

1. Backlog Grooming. Typically occurs the first day of each sprint to get it started.

   a. Ensure all known work is expressed as value (Stories not tasks/activities).

   b. Add, modify until all known Stories are captured.

   c. Prioritize product backlog.

   d. Relatively estimate complexity of all Stories

2. Sprint Planning. After the product backlog is groomed, the product owner, development team, and scrum master will estimate how much work they can complete in the upcoming sprint. Typically, the team uses past velocity (the average number of story points the team completed in previous sprints) to predict the amount of work they can commit to in the upcoming sprint. Once the target velocity is determined, the team selects the stories previously prioritized by the product owner until the total number of Story Points meets but does not exceed their velocity target. This work is then pulled out of the Product Backlog and into the Sprint Backlog.

Once the Sprint Planning is complete, to avoid disrupting the team, no other work should be added to the Sprint Backlog. However, the product owner can add, modify, or reprioritize items remaining in the Product Backlog as they see fit. At this point, the team is trusted and empowered to determine "how" to get the work done and how to deliver it in the most efficient way possible. Therefore, they are not asked to provide tasks/activities and are not asked to translate Stories or related relative estimates to time-based estimates. The goal is to track value delivery (completed Stories should provide stand-alone value), not steps toward value delivery (tasks, activities). Teams that populate either the Product or Sprint Backlog with tasks and

activities are not operating in an Agile fashion and will greatly inhibit the usage and value of Agile metrics.

3. Daily Scrum Meetings. After Sprint Planning occurs, the development team will hold a short, daily meeting to determine ownership of Stories and potential issues/blockers. If facilitated correctly, this meeting should take 10-15 minutes with up to 10 people. The Scrum Master facilitates the meeting by asking three questions to each team member. These questions are to be answered succinctly without providing a "status update" on progress:

- What Stories have you completed since the last meeting?
- What Stories are you working on now?
- Are there any blockers or obstacles to completing your work?

4. Sprint Review/Demo. At the end of each sprint, the product owner, development team, and scrum master review the Stories completed during the sprint. Each Story completed must adhere to the Definition of Done (that applies to all stories) and the Acceptance Criteria (that applies to a specific story). In this meeting, the team demonstrates working product (not wireframes or screenshots) and the product owner has the final say on if a Story is accepted or rejected. If rejected the Story goes back into the Product Backlog for grooming and reprioritization.

5. Sprint Retrospective. After the Sprint Review/Demo, the team huddles to evaluate the progress made during the sprint, the Agile metrics produced, and considers what continuous improvement experiments. These experiments are captured as enabling Stories in the Product Backlog. Typically, teams target 1-2 continuous improvement experiments (the ones that will add the most value) in the next sprint. The team tries to keep this number low to ensure they continue delivering value to customers but also so they can track better/worse performance back to the experiment (traceability). To guide a Sprint Retrospective, the team will usually ask the following questions: 1) What went well that we should continue doing? 2) What didn't work so well that we should stop/fix? And 3) What are some things we haven't tried to improve performance (start)?

OUSD(R&E) Lesson Learned. Typically, each sprint will follow immediately from the previous one.

Issues to consider in the contract include the following:

- It is likely that the parties will want to capture the sprint process in the contract in some form. Consider whether compliance with the sprint process should be contractually binding.

- In relation to the sprint planning meetings, the contract will need to include a regimen for the development team to determine how many of the high-priority items identified by the product owner can be developed during the current sprint (e.g. velocity as an indicator).
- The contract should include an acknowledgement from the customer that, once the items to be developed in each sprint have been identified, they are fixed for that sprint.
- The contract should set out a regimen for the development of the sprint backlog by the development team following each planning meeting. It may be useful to include an agreed format for the sprint backlog in an appendix to the contract (e.g., Jira).
- The contract should require that any future improvements agreed to at a sprint review meeting are captured in the Product backlog (and that these improvements are reviewed and prioritized at the next backlog grooming session).
- The contract should provide for the parties to move continuously into the next sprint cycle. The sprint cycle should continue until either the following occur:
    - The project is completed, or
    - The contract is terminated.

One question arises is whether compliance with the more detailed aspects of the sprint process (e.g., daily sprint meetings and/or sprint backlog burndown charts) should be contractually binding. In some projects, this may be desirable to ensure that the sprint process is properly followed by the parties. One possible solution is to provide that the key responsibilities of the parties (such as those issues to consider above) are contractually binding while leaving the more detailed day-to-day aspects of the sprint process as non-binding.

7.9.3 Definition-of-Done and Acceptance Criteria

An important Agile principle is that each sprint cycle should result in a "potentially shippable product increment." Questions arise as to how the parties will determine whether this has been achieved.

In Scrum projects, the key to this determination is establishing an agreed upon definition-of-done that applies to all Stories (e.g., you have to peer review your code) and acceptance criteria that is applies to a specific story (e.g., the shopping cart story must have shipping costs and tax included in the total). , which one Agile practitioner described this concept as the "soul" of the entire process. Some of the elements of a definition-of-done could be as follows:

- The scope of tests to be conducted and passed (e.g., user acceptance tests and non-functional tests).
- All code has been reviewed (or pair programmed).

- All coding standards have been met and code has been re-factored where necessary.
- Automated tests were developed so manual testing is not required.
- Necessary documentation has been completed.

Key points to address in the contract include:

- The definition-of-done should ideally be developed in parallel with the negotiation of the contract.
- The contract should also include an appropriate dispute resolution procedure if there is a dispute between the parties as to whether any item has been completed in accordance with the definition-of-done.

7.9.4 Project Completion

Products are never done but a development project may end when all the items listed in the product backlog have been developed and released. The list of items in the product backlog at the end of the project may not be the same as at the start – during the project, the product owner may have decided that some of the features identified at the start of the project are no longer needed.

A key issue to be addressed in the contract includes the following:

- Consider contracting for development capacity for a small period of time, or
- The contract should identify criteria for completion.

7.10 Pricing

Program offices, customers, and suppliers are likely to have different perspectives on how an Agile development project should be priced. In many cases, the program office will be seeking to agree to a fixed price while the supplier will want to work on a T&M basis. Program Managers worry that by agreeing to use Agile, which may involve an unknown number of iterations, they are effectively writing a “blank check” for the project costs with few constraints on cost escalation.

Counterarguments from Agile proponents include the following:

- Waterfall makes the promise of value delivery far off into the future with larger upfront investments. This is an incredibly risky way to invest. Instead, Agile provides valuable working product delivered over a shorter investment horizon, which allows leadership to make data-driven, micro-investment decisions reduces risk.

- Waterfall leverages documentation and adherence to schedule and budget as the primary indicators of success. The product itself isn't delivered until far off in the future and everyone has to hope until then that it will be valuable. Instead, Agile delivers working product quickly and routinely incorporates customer feedback to maximize value.
- It is unrealistic to expect any development project, whether based on the Waterfall or Agile, to execute without changes in scope
- A fixed price model unless it is used to purchase development capacity may erode the intended benefits of Agile by encouraging the parties to retreat to the traditional approach of building to rigid specifications and adversarial change management/ contract negotiation.
- If the program has a fixed budget, Agile maximizes the value of the investment (unlike many Waterfall projects) by focusing on development of the high-priority items first and allowing the product owner to deprioritize lower priority (or "nice to have") items from the scope.

It is equally unlikely that a pricing model based solely on T&M will drive the right behavior by the parties. For example, a T&M pricing model is likely to result in disincentives for the supplier to create realistic estimates and stick to them. In addition, all these issues need to be considered when determining the pricing model to be used for an Agile development project. In particular, the parties need to be conscious that pricing needs to be addressed in relation to both individual iterations and the project as a whole.

Some potential pricing models include the following:

- Fixed Price per Development team: In this instance, you are purchasing software development capacity. As your knowledge about team cost structures grows you can use this as an investment lever to ramp capacity up or down.
- Fixed Price per Iteration. Perhaps calculated by reference to the amount of work required for that iteration or the business value of the relevant development items.

The following are not recommended:

- Fixed Price per User Story. User Stories have different relative complexity and so a highly complex user story that takes longer to deliver would be paid at the same rate as an easier, less complex user story.
- Fixed Price for Agreed Number of Stories. Difficult to estimate exact number of stories required for project because scope can change.

One of the key issues for the supplier in agreeing to a pricing model will likely be revenue realization. Under a traditional Waterfall contract, the supplier may be able to invoice the full

contract price upfront whereas, under an Agile contract, the supplier may only be entitled to invoice the charges that the program is contractually required to pay. This issue may be particularly relevant if the program is entitled to terminate the contract at the end of each iteration (or at other agreed upon points) without an obligation to pay the full contract price.

Key points to address in the contract include the following:

- The agreed-upon pricing model will need to be clearly set out in the contract.
- Consider contracting for dedicated development capacity
- Contract clauses will need to include:
  - A description of the pricing methodology (e.g., fixed price per iteration),
  - When fees can be invoiced,
  - Who bears the costs for items which have not been completed during an iteration (or which have not been met as part of the definition-of-done, or equivalent), and
  - The impact of scope reductions or early termination.
- It may also be useful to specify different pricing models for the fixed content of the MVCR versus the iterative, dynamic content of subsequent batches.

7.11 Warranties and Indemnities

Given the more iterative and collaborative nature of Agile development projects, what warranties and indemnities can the supplier be expected to give? There are two aspects to consider: (a) Compliance with specification warranties, and (b) Composition of the development team.

With compliance and warranties, one of the key warranties in a traditional software development contract is that the finished product will comply with the functional specification; however, one of the features of Agile is that a comprehensive functional specification is not developed at the outset of the project. To bridge this gap, it may be useful if on completion of the project, the supplier prepares a “Product Description” which does the following:

- Contains a detailed description of the design and functions of the completed product, and
- Demonstrates how the completed product is consistent with the product vision (addressing the Capability Needs Statement, User Agreement, and Value Assessment).

As with a functional specification, the Product Description should be subject to review and comment by the Program Office and customer, with any disputes between the parties being subject to the agreed dispute resolution procedure.

With the composition of the development team, one of the potential problems mentioned in relation to a combined development team is the difficulty in establishing a clear allocation of risk and liability between the program and the supplier. The problem is particularly acute when it comes to negotiating warranties and indemnities. If the Program Office and customer personnel are to be involved in the development activities on a day-by-day basis, then the supplier will be very reluctant to:

- Offer any substantive warranties that the developed product (or individual product increments) will be free from defects, fit for purpose or of satisfactory quality, and/or
- Offer a substantive indemnity against third party IP infringement claims.

If the Program Office and customer lack the required skills to participate in the development team, it may be more appropriate for that team to be composed of only supplier personnel. This would put the supplier in a better position to offer the above warranties and indemnities.

Key issues to be addressed in the contract include the following:

- The contract should include appropriate warranties from the supplier. These could include the following:
  - The product is sufficiently free from defects and of satisfactory quality, or
  - The product will comply with the agreed Product Description.
- If agreed upon by both parties, these warranties could be limited to a defined "warranty period" as per standard contracting approaches.
- If appropriate, these warranties could be given by the supplier in relation to individual product increments at the end of each iteration.
- If a "Product Description" will be prepared, the contract should include a regimen for the development and agreement of this document (including a dispute resolution mechanism where necessary).
- The contract may also include warranties from the supplier regarding the following:
  - Use of OSS by the development team.
  - Software Assurance, Safety, Reliability, Security, and Survivability/Resiliency.
- If prescribed in the guidance under FAR 27.201-2(c)(1), procurements should include a standard FAR patent indemnity clause under FAR 52.227-3.
  - This can be supported by standard provisions dealing with conduct of proceedings and rights for the supplier to modify the product so that it becomes non-infringing.

7.12 Termination

When should either party be entitled to bring an end to the project? Is it realistic in a project of any size for the customer (and potentially the supplier) to have the right to walk away from the project after each iteration? Arguably, this is an inherent right for the customer in an Agile project – at any time, the product owner could amend the product backlog to de-scope any outstanding items and declare the project complete. On the other hand, the supplier may have invested significant time and resources in dedicating a development team to the project and feel that it is entitled to some form of compensation if the project is cancelled earlier than expected.

Key issues to be addressed in the contract include the following:

- Address each party's right to terminate the project.
- Whether the Program Office has a right to terminate after each iteration (or possibly after defined "groups" of iterations).
- Include standard rights to terminate immediately, such as material breach or insolvency.
- Address the consequences of termination, including the delivery to the Program Office / customer of work-in-progress including software code and copies of other working materials in their current state of development.
- Include conditions and terms to include compensation if the contract is terminated early (e.g., some or all the profit that the supplier may have made from future iterations).

7.13 Intellectual Property Rights

What are the Government's license rights to the software developed or delivered? What are the Government's license the rights to the build environment needed to build the software from the source code? Presuming that this approach is supported by the acquisition strategy and product support strategy, it is preferable that the Government requires delivery of source code, libraries, and toolchains that provide the capability to build the software from source code.

In addition, the Government should ensure (either through standard DFARS clauses under procurement contracts, IP clauses in other types of agreements, or by negotiation with the contractor) that the Government is granted license rights that permit the Government to use, modify, and distribute software deliverables in a manner that enables the acquisition goals established in the acquisition strategy and product support strategy. Use of DoD enterprise software factory assets is one way to achieve this goal and avoid vendor lock. These can be complex IP considerations, which should be first vetted and coordinated with the cognizant IP SME (e.g., experts from the DoD IP Cadre).

Key issues to be addressed in the contract include the following:

- Identify the respective supplier and the IP license rights necessary to support the Government's individual product increments and the fielding of the final developed product.
    - The supplier may be required to provide copies of the relevant source code to the Government.
    - Include appropriate licenses that meet the Government's needs as established in the Acquisition Strategy (AS) and Post Production Support (PSS), based on the Board Contract of Appeals (BCA) determination.
    - Identify whether software escrow arrangements[5] or other licensing arrangements may satisfy future product supportability needs.
- Data and IP rights should align with the Professional Services Schedule (PSS).
    - Even if DoD is not the developer of the software (e.g., tools, libraries, containers) then may require access to source code for validation and verification (V&V) and software assurance (SwA) purposes.
    - If PSS requires DoD to buy containers and libraries so that software can share with a third party, then DoD needs rights to that code.
- In any event, the contract should require delivery of the product vision and the product backlog and associated license rights that enable the acquisition strategy and product support strategy

7.14 Dispute Resolution

Given the more collaborative approach to agile projects, it will be important for the contract to include a procedure that promotes the quick and efficient resolution of disputes while maintaining good working relationships between the parties.

It may be that this can be best achieved through the combination of the following:

- Informal discussions by the parties, escalating up to senior management.
- Where this process does not resolve the dispute, determination of technical or financial issues by an independent expert with only the most serious or intractable disputes being referred to arbitration or court proceedings. Mediation should also be considered by the parties as a means of resolving disputes in an efficient and "non-destructive" manner.

[5] Escrow agreements require deferred delivery of technical data or software upon the occurrence of specific events indicated in the contract (e.g., the contractor's cease of sale or support of products or bankruptcy).

8 Artificial Intelligence and Machine Learning

- The importance of AI/ML is growing in defense systems and has potential to become critical to dominance on the battlefield.
- Machine learning enables computers to learn from data and data relationships without being explicitly programmed.

This section differs from the other sections in this guide in that artificial intelligence (AI) and machine learning (ML) (AI/ML) is not yet in widespread deployment across the DoD. Consequently, addressed is more background on AI/ML as a field and the Department's strategic approach and vision for deploying these technologies. This vision lays out a path for transitioning AI/ML software into much broader application across the Department.

The importance of AI/ML is growing in defense systems. The promise of radical advances in the ability to perceive and react to complex situations offers compelling advantages to the warfighting mission. The purpose of this section is to demonstrate that software engineering for systems incorporating AI/ML may differ in ways that cause adjustments to in-place practices and processes. This information should assist in the adaptation of software engineering to the special nature of AI/ML development and deployment.

The essential way in which ML systems differ from traditional software applications is their reliance on data. Data collection and curation is the critical element driving the pace of software development. In fact, for many applications, use of the Software Pathway will be infeasible for schedule reasons if the curated data is not prepared before entry into the pathway.

The AI/ML infrastructure and processes delineated here harmonize with the Agile/DevSecOps software engineering and continuous delivery approaches described in the preceding sections. Software engineering practices (including CI/CD pipelines, small batch sizes, iterative development, automated testing, emphasis on rapid deployment, and feedback gathered from operations) are all key enablers of the Department's AI/ML development and deployment strategy.

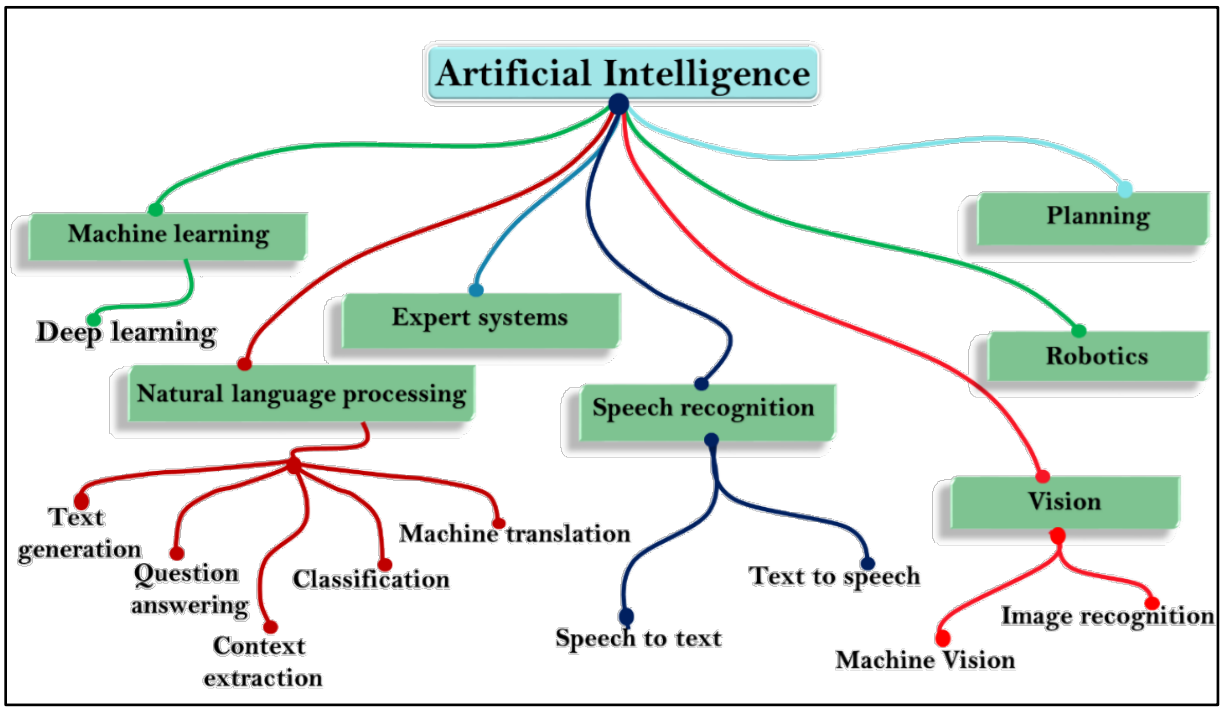
This section presents:

- Background on AI/ML, elaborating on what it is and how AI/ML relates to DoD's warfighting mission.
- How the Chief Digital and Artificial Intelligence Office (CDAO) is positioned to help programs transition AI/ML research into fielded capabilities.
- An overview of the OUSD(R&E) AI Software Roadmap to familiarize the reader with a spectrum of technologies that fall under the umbrella of AI/ML.

- A conceptual model of how AI/ML development aligns with the CI/CD pipeline construct as part of a DevSecOps approach featuring short cycle iteration and continuous feedback from development to operations.

8.1 Background on Artificial Intelligence and Machine Learning

AI is the pursuit of the ability of machines to perform tasks that normally require human intelligence or the programmed ability to process information. AI focuses on aggregating ML, Planning, Expert Systems, Natural Language Processing, Speech, Robotics, and Vision to create a "Decision Platform" independent of human intervention (Crisman 2020). This taxonomy has evolved over many years, captured in Figure 8-1.



Source: JAIC 2020

Figure 8-1. AI Taxonomy

The definition of AI now encapsulates ML, or techniques that give computers the ability to learn without being explicitly programmed to do so. These techniques allow data scientists to develop AI applications rapidly with data. The speed at which applications can be developed depends upon the prior existence of suitable data, and/or the collection and preparation of new data. Aggregating both data and ML creates a model for information/data processing with mission implications. This combination, known as "Deep Learning" (DL), allows AI practitioners to develop complex AI applications with larger data sets or "big data."

ML enables computers to learn from data and data relationships without being explicitly programmed. The best way to understand ML is to contrast it with an older approach to AI, human-centric knowledge systems. Knowledge systems or expert systems are AI systems that use traditional, rules-based software to codify subject matter knowledge of human experts into a long series of programmed "if given x input, then provide y output" rules. For example, the AI chess system Deep Blue, which defeated the world chess champion in 1997, was developed in collaboration between computer programmers and human chess grandmasters. The programmers wrote (literally typed by hand) a computer code algorithm that considered many potential moves and countermoves reflecting rules for strong chess play given by human experts (Greenemeier 2017).

ML includes the "No Free Lunch" theorem, which means that no one ML algorithm works best for every problem, especially relevant for predictive modeling. For example, one cannot say that neural networks are always better than decision trees or vice versa. There are many factors to consider, such as the size and structure of the data set. As a result, many different algorithms should be applied to the problem, while using an actual, curated test set of data to evaluate performance and identify the highest performing algorithm to be deployed.

Currently, ML is the preferred "go to" approach in the development of AI applications, which streamlines and optimizes the software development process carried out by humans. Realized benefits of ML include:

- Better performance than that provided by humans.
- Shortened development time of new code, i.e., code potentially developed in minutes.
- Expansion of new problem sets.

Achievement of ML is closely related to statistics and requires "big data" for successful task completion, and data that is of high quality. Rapid and successful development moving forward will require the capture of the right and accurate data in the right way and connecting that data to the right team(s) for synthesis and development.

With all the positives, ML is vulnerable to errors in modeling and simulation of mission problems, which can lead to errors in AI reasoning solutions. ML is also vulnerable to biases and inaccuracies in training data, which can lead to biases and inaccuracies in trained ML models. In addition, securing and preserving ML integrity can be both challenging and time consuming; ML is vulnerable to adversarial attacks on training data sets, attacks on model parameters, attacks on model outputs, and alterations to the environment such as deception (NISTIR 8269 (draft) 2019).

8.2 Chief Digital and Artificial Intelligence Office (CDAO) Strategy

The DoD's Chief Digital and Artificial Intelligence Office (CDAO) was established to preserve and expand U.S. military advantage in support of the Department's 2018 National Defense Strategy. As a primary executing body, it is tasked to accelerate the delivery of AI-enabled capabilities, scale the Department-wide impact of AI, and synchronize DoD AI activities to expand Joint Force advantages.

The CDAO is responsible for the acceleration of DoD's adoption of data, analytics, and AI to generate situational advantages. The goal is to use AI to solve large and complex problem sets that span across the DoD, then ensure real-time access to ever-improving libraries of data sets and tools. The CDAO's holistic approach includes activities to:

- Accelerate the delivery and adoption of AI.
- Scale the impact of AI across the Department.
- Defend U.S. critical infrastructure from malicious cyber activity that alone, or as part of a campaign, could cause a significant cyber incident.
- Establish a common foundation that enables decentralized execution and experimentation.
- Develop partnerships with industry, academia, allies, and partners.
- Cultivate a leading AI workforce.
- Lead in military AI ethics and safety.

The CDAO delivers AI capabilities to the Department through two categories: National Mission Initiatives (NMIs) and Component Mission Initiatives (CMIs):

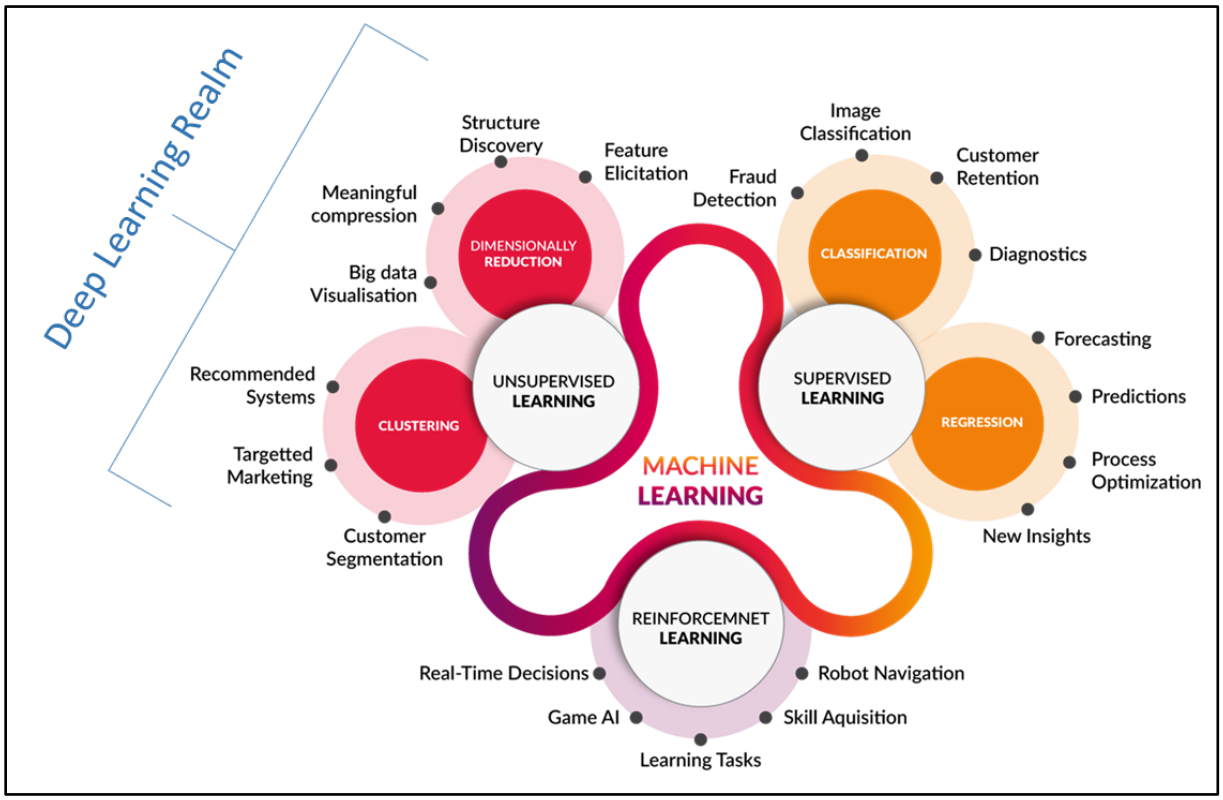
- NMIs are broad, joint, and cross-cutting AI/ML challenges that the CDAO must tackle using a cross-functional team approach.
- CMIs are component-specific and solve a particular problem. CMIs will be run by the components, with support from CDAO in several ways that include funding, data management, common foundation, integration into programs of record, and sustainment.

8.3 OUSD(R&E) Artificial Intelligence Software Roadmap

The DoD AI Strategy (DoDAIS 2018) (JAIC 2020) defines AI as the "ability of machines to perform tasks that normally require human intelligence." This definition includes decades-old application, such as aircraft autopilots, missile guidance, and signal processing systems. Though many AI technologies are old, there have been technological breakthroughs over the years that have greatly increased the diversity of applications where AI is practical, powerful, and useful.

Most of the breakthroughs in AI over the past decade have focused on ML. The ability of ML to provide applications to the Department is based on what it can affect in terms of functions and how these functions can be used during software development for warfighter capabilities. ML can be thought of as applications in "Unsupervised," "Supervised," and "Reinforcement" learning, all of which provide specific functions associated with each area. Source: JAIC 2020

Figure 8-2 illustrates and provides bullet points for these three learning types.

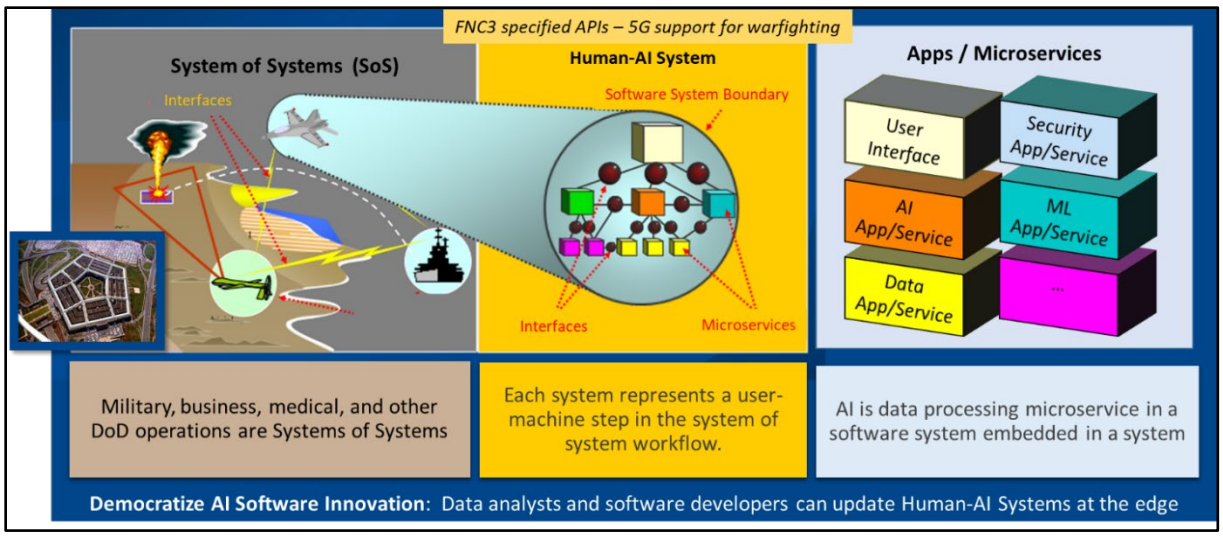


Source: JAIC 2020

Figure 8-2. Machine Learning Composition Domain

As part of the software system, data analysts and software developers can update human AI systems at the edge because AI and ML is part of the overall software system architecture. Edge computing is defined as client data processed at the periphery of the network, as close to the originating source as possible.

Within a system of systems (SoS), military, business, medical, and other DoD operations are part of its composition. Within the human-AI system, each system represents a user-machine step in the SoS workflow. Within the application and microservices realm, AI is a data processing microservice in a software application that is embedded in a system. Figure 8-3 illustrates this concept, providing capabilities within software for supporting the warfighter and the Department's missions.



Source: JAIC 2020

Figure 8-3. AI and ML as Part of a Software System

High-quality ML requires expertise to be effective, useful, and implementable. Human resource attributes include personnel who:

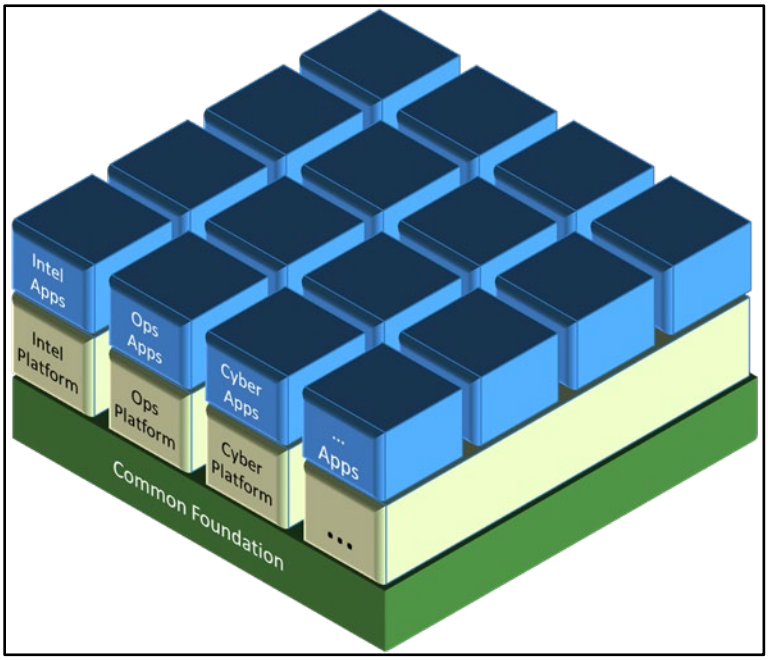
- Understand data and data processing mathematically; typically, AI/computer scientists, data scientists, or ORSA (Operations Research and Systems Analysis) analysts.
- Understand how to make machines collect, store, retrieve, and process data efficiently and securely; typically, data/software/cyber engineers.
- Understand the meaning of source data and its use for tactics, techniques, and procedures; typically, data users and operational SMEs.

The DoD AI strategy (DoDAIS 2018) is based on the notion that by combining mission platforms, having a common foundation, the right workforce with the right skills and training, forged partnerships with commercial, academic, and international allies and partners, as well as a solid policy that has leadership in military ethics and AI safety, AI-enabled capabilities will be achieved through the rise from experiments at the "forward edge" discovered by the users. Mission platforms need to deliver AI-enabled capabilities that address key missions and allow for experimentation and development at the edge. The common foundation must scale AI's impact across the DoD by enabling decentralized development and experimentation. Development of the workforce that supports the AI strategy must cultivate a leading AI workforce for rapid experimentation and an iterative, risk-informed approach to AI implementation.

8.4 Vision for Accelerated, Continuous Delivery of AI/ML Capability

Accelerating AI adoption requires a common foundation that has building blocks for software and AI practitioners to build mission platforms and applications; e.g., Intel, Operations, and Cyber. Source*: JAIC 2020*

Figure 8-1 illustrates this AI adoption layer model.



Source: JAIC 2020

Figure 8-1. AI Adoption Layer Model

Success starts with having a viable infrastructure that uses the commercial Cloud and "cloudlet" services or Infrastructure as a Service (IaaS), as well as DoD High Performance Computing (HPC) and other DoD-owned systems. The Department envisions common software services such as identity management services, data management, DevSecOps, mapping, visualization, feature extraction, and ML training.

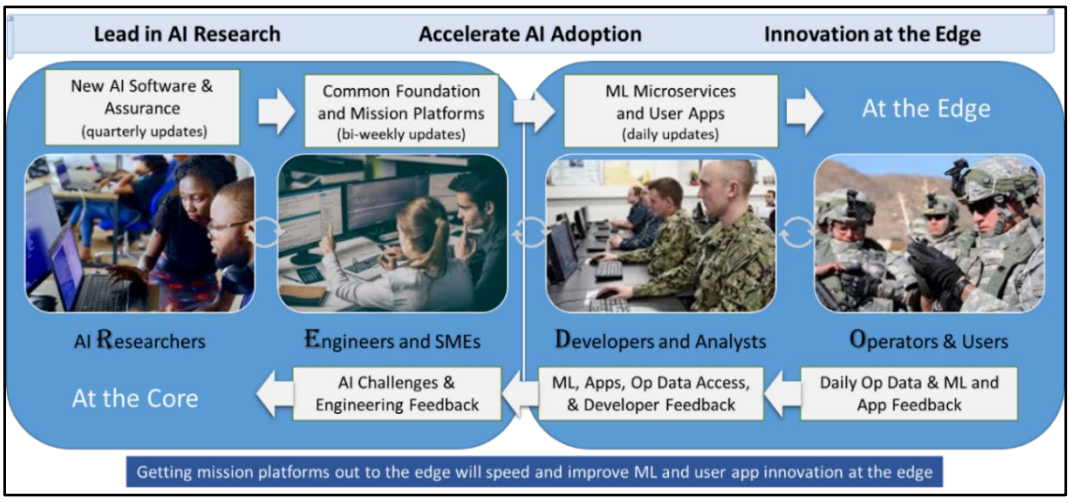
Using Mission Platforms as the "backend" that enables rapid ML, application development, and testing, supports acceleration of AI adoption across the ecosystem. This is accomplished by understanding several components:

- Microservices Software Architecture, and the use of
  - Documented APIs for all software services supporting a defined multi-user mission workflow.
- Unique Testing Needs, and the need to

  - Reserve test data that was not used in training.
  - Assess adversarial countermeasures that may defeat ML recognition by creating small differences in inputs.
  - Understand and explain why the ML system behaves as it does.
  - Understand the strict boundary between fully autonomous and human in the loop (HITL) decisions.
- Mission-specific Data and Software Services, using
  - Raw and processed sensor data.
  - Sensor calibration and signal processing services.
- Mission-specific AI/ML Development Services, and the use of
  - Simulations and game engines.
  - Training data set, algorithms, and models.
  - AI/ML testing harnesses to include ethics and vulnerability measures.
- Incremental ATO on mission end points, and the use of
  - DoD Cloud for business services.
  - Cloudlets for contested logistics.
  - DoD vehicles or devices for military operations.

Within the secure DevSecOps process associated with the ATO, the Department's vision foresees an active effort to develop AI/ML software in conjunction with the ATO accreditation process for the deployment on the cloud instances.

Figure 8-5 illustrates this closed-loop input/output, feedback process.



Source: Adapted from OUSD(R&E) DoD-Wide Software S&T Strategy 2021

Figure 8-2. AI Process Automation to the Edge

AI innovation requires continual updates to user applications and ML resources. As new data becomes available, it may be added to curated training data sets. These data sets enable new ML models to be rapidly retrained, tested, and deployed, expanding the capabilities of the system. When new AI/ML vulnerabilities or biases are uncovered, the AI/ML and testing services can be updated to screen for those vulnerabilities and biases.

Implementing mission platforms and scaling AI and ML from research requires inputs and feedback from the various stakeholders involved in the process. These stakeholders include AI researchers, engineers, and SMEs; developers and analysts; and operators and users.

- AI researchers provide new AI software methods and data assurance methods.
- Engineers and SMEs provide a common foundation and mission platform.
- Developers and analysts provide daily ML microservices, applications, and application updates.
- Operators and users provide daily operational data, and ML output and application feedback.

8.5 Summary

As AI/ML grow in acceptance the Department must meet a collection of new challenges:

- Validation and verification of capability when an AI/ML changes its behavior
- Curation of high quality training data reflective of operational environments
- Ethical concerns regarding what decisions get delegated to AI/ML systems.
- Workforce competency development and training
- Transitioning AI/ML research into new capabilities in operational systems

Glossary

Term	Definition
Adaptive Acquisition Framework (AAF)	A series of acquisition pathways to enable the workforce to tailor strategies to deliver better solutions faster. The AAF pathways provide opportunities for Milestone Decision Authorities, Decision Authorities, and Program Managers to develop acquisition strategies and employ acquisition processes that match the characteristics of the capability being acquired.
Application Programming Interface (API)	A set of definitions and protocols for building and integrating application software. Source: https://www.redhat.com/en/topics/api/what-are-application-programming-interfaces A system access point or library function that has a well-defined syntax and is accessible from application programs or user code to provide well-defined functionality. Source: https://csrc.nist.gov/glossary/term/application_programming_interface NIST SP 1800-16C NIST SP 1800-21C NIST SP 5153 under Application Program Interface
Artifact	*Software Artifact:* A consumable piece of software produced during the software development process. Except for interpreted languages, the artifact is or contains compiled software. Important examples of artifacts include container images, virtual machine images, binary executables, jar files, test scripts, test results, security scan results, configuration scripts, Infrastracture as Code, documentation, etc. Software artifacts are usually accompanied by metadata, such as an identifier, version, name, license, dependencies, build date, and time. Items such as source code, test scripts, configuration scripts, build scripts, and infrastructure as code are checked in to the source code repository, not the artifact repository, and are not considered artifacts. *Artifact Repository:* A system for storage, retrieval, and management of artifacts and their associated metadata. Programs may have separate artifact repositories to store local artifacts and released artifacts. It is also possible to have a single artifact repository and use tags to distinguish the content types.
Backend	The part of a computer system or application that is not directly accessed by the user, typically responsible for storing and manipulating data.
Backlog	Program backlogs that identify detailed user needs in prioritized lists. The backlogs allow for dynamic reallocation of scope and priority of current and planned software releases. Issues, errors, and defects identified during development and operations should be captured in the program's backlogs to address in future iterations and releases.
Bare Metal/Bare Metal Server	A traditional physical computer server dedicated to a single tenant and which does not run a hypervisor. This term is used to distinguish physical computer resources from modern forms of virtualization and Cloud hosting.

Term	Definition
Binary/Binary File	A data file or computer executable file that is stored in binary format (as opposed to text), which is computer-readable, but not human-readable. Examples include images, audio/video files, exe files, and jar/war/ear files.
Build (Software)	The process of creating a set of executable code that is produced by compiling source code and linking binary code. The term build may also refer to the end product created by that process.
Build Tools (Software)	Tools used to retrieve software source code, build software, and generate artifacts.
Capability	Higher level solution typically spanning multiple releases. A capability is made up of multiple features to facilitate implementation.
Capability Needs Statement (CNS)	A high-level capture of mission deficiencies, or enhancements, to existing operational capabilities, features, interoperability needs, legacy interfaces, and other attributes that provides enough information to define various software solutions as they relate to the overall threat environment.
Competency	An observable, measurable pattern of knowledge, abilities, skills, and other characteristics that individuals need to perform work roles or occupational functions successfully. They are categorized by: non-technical. Demonstrate the “soft skills,” (ability to relate to others) or personal attributes associated with successful performance of current and future job tasks or mission requirements as defined in DoDI 1430.16 [...] technical. Associated with a specific occupation or function to successfully perform the job tasks required. These competencies reflect domain-specific requirements and are associated with analysis of occupational job groups or families, occupational series, DoD critical functions particular groups of jobs. These competencies also refer to specific occupational skills gained from education or training or which are based on a particular area of expertise. Source: DoDI 1400.25 Volumn 250, June 2016
Continuous Authorization to Operate (cATO)	Concept of building software security controls into the software development methodology so the authority to operate process (as with the testing process) is completed alongside development. If designed correctly, an authority to operate is nearly guaranteed once the software is ready for release.

Term	Definition
Continuous Integration/Continuous Delivery (CI/CD)	*CI/CD Orchestrator:* A tool that enables fully or semi-automated short-duration software development cycles through integration of build, test, secure, store artifact tools. *CI/CD Pipeline:* A set of tools and the associated process workflows to achieve continuous integration and continuous delivery with build, test, security, and release delivery activities, which are steered by the CI/CD orchestrator and automated as much as practice allows. *CI/CD Pipeline Instance:* A single process workflow and the tools to execute the workflow for a specific software language and application type for a project. The pipeline process is automated as much as practicable.
Cloud Native Computing Foundation (CNCF)	An open source software foundation dedicated to making cloud native computing universal and sustainable. Cloud native computing uses an open source software stack to deploy applications as microservices, packaging each part into its own container, and orchestrating those containers to optimize resources. Cloud native technologies enable software developers to build products faster. Source: https://www.cnfc.io/
CNCF-Certified Kubernetes	Kubernetes that has been endorsed by the CNCF Certified Kubernetes Conformance Program. Software conformance ensures that every vendor's version of Kubernetes supports the required Application Performance Interfaces. Conformance guarantees interoperability among Kubernetes from different vendors. Most of the world's leading vendors and Cloud computing providers have CNCF-certified Kubernetes offerings.
Code	Software instructions for a computer, written in a programming language. These instructions may be in the form of either human-readable source code or machine code, which is source code that has been compiled into machine-executable instructions.
Code Coverage	*A* measure used to describe what percentage of application code is exercised when a test suite runs. A higher percentage indicates more source code executed during testing, which suggests a lower chance of containing undetected bugs.
Configuration Management	Capability to establish and maintain a specfic configuration within operating system and applications.
Container	A standard unit of software that packages code and its dependencies down to, but not including, the OS. The container is a lightweight, stand-alone, executable package of software that includes everything needed to run an application except OS: code, runtime, system tools, system libraries, and settings. Source: https://docker.com/resources/what-container A method for packaging and securely running an application within an application virtualization environment. Also known as an application container or a server application container. Source: (NIST SP 800-190 2017)

Term	Definition
Continuous Build	An automated process to compile and build software source code into artifacts. The common activities in the continuous build process include compiling code, running static code analysis such as code style checking, binary linking (in the case of languages such as C++), and executing unit tests. The outputs from continuous build process are build results, build reports (e.g., the unit test report, and static code analysis report), and artifacts stored in the artifact repository. The trigger to this process could be a developer code commit or a code merge of a branch into the main trunk.
Container Orchestration	The automation of much of the operational effort required to run containerized workloads and services. This includes a wide range of activities software teams need to manage a container's lifecycle, including provisioning, deployment, scaling (up and down), networking, load balancing, and more. Source: https://vmware.com/topics/glossary
Continuous Delivery	An extension of continuous integration to ensure that a team can release the software changes to production quickly and in a sustainable way. The additional activities involved in continuous integration include release control gate validation and storing the artifacts in the artifact repository, which may be different than the build artifact repository. The trigger to these additional activities is successful integration, which means all automation tests and security scans have been passed. The human input from the manual test and security activities should be included in the release control gate. The outputs of continuous delivery are a release go/no-go decision and released artifacts, if the decision is to release.
Continuous Deployment	An extension of continuous delivery. It is triggered by a successful delivery of released artifacts to the artifact repository. The additional activities for continuous deployment include, but are not limited to, deploying a new release to the production environment, running a smoke test to make sure essential functionality is working, and a security scan. The output of continuous deployment includes the deployment status. In the case of successful deployment, it also provides a new software release running in production. On the other hand, a failed deployment causes a rollback to the previous release.
Continuous Engineering	A practice that merges requirements, design, development, quality assurance, security, test, integration, delivery, and deployment into a single, continuous set of processes to continually, or iteratively, provide working functional systems to internal and external users users and to deliver high-quality software more frequently.
Continuous Integration	A step further than continuous build. Continuous integration extends continuous build with more automated tests and security scans. Any test or security activities that require human intervention can be managed by separate process flows. The automated tests include but are not limited to integration tests, a system test, and regression tests. The security scans include but are not limited to dynamic code analysis, test coverage, dependency/bill of material (BOM) checking, and compliance checking. The outputs from continuous integration include the continuous build outputs, plus automation test results and security scan results. The trigger to the automated tests and security scans is a successful build.

Term	Definition
Continuous Integration / Continuous Delivery (CI/CD) Pipeline	A collection of DevSecOps tools, with which the DevSecOps process workflows can be created and executed. DevSecOps tools are composed of a tailored series of software products configured to integrate end-to-end software definition, design, development, test, delivery, and potentially deployment in a highly automated and secure way.
Continuous Monitoring	An extension of continuous operation. Operators are supported by automated services that continuously monitor and inventory all system components, monitors the performance and security of all the components, and audit and log system events.
Continuous Operation	An extension of continous deployment. Continuous operation is triggered by successful deployment. The production environment operates continuously with the latest stable software release. The activities of continous operation include but are not limited to system patching, compliance scanning, data backup, and resource optimization with load balancing and scaling (both horizontal and vertical).
Cybersecurity	The art of protecting networks, devices, and data from unauthorized access and the practice of ensuring confidentiality, integrity, and availability of information. The term covers preventative methods used to protect software from threats that may exploit weaknesses, and vulnerabilities in the software.
Cycle Time	The elapsed time from when work is started until the time the work has been completed.
Decision Authority (DA)	The official responsible for oversight and key decisions of programs that use the software acquisition pathway and related component policies. The DA designates a Program Manager (PM) and supports the PM in tailoring and streamlining processess, reviews, and decisions to enable speed of capability delivery. The DA may be the Defense Acquisition Executive, Component Acquisition Executive, or the Program Executive Officer, or other official designated by the Component Acquisition Executive (CAE).
Defense Business System (DBS)	Defined in Title 10 USC 2222.
Defect (Contained)	A defect that is introduced, detected, and repaired within a given development stage before moving to a later stage.
Defect (Escaped)	A defect that is introduced in a given development stage but not detected or repaired until a later stage.
Delivery	The process by which released software is placed into an artifact repository where it becomes available for deployment to the operational environment.
Deployment	The process by which released software is downloaded and deployed to the operational environment.

Term	Definition
DevSecOps	An organizational software engineering culture and practice that aims at unifying software development, security, and operations. The main characteristic of DevSecOps is to automate, monitor, and apply security at all phases of the software lifecycle: plan, develop, build, test, release, deliver, deploy, operate, and monitor. In DevSecOps, testing and security are shifted left through automated unit, functional, integration, and security testing. This shift differentiates DevSecOps from other methods of software development in that security and functional capabilities are tested and built simultaneously.
DevSecOps Phase	Any of eight phases of software development, security, and operation activities in the software life cycle. Each phase completes a part of related activities using tools.
Digital Engineering (DE)	An integrated digital approach that uses authoritative sources of systems' data and models as a continuum across disciplines to support life cycle activities from concept through disposal. Definition source: "DAU Glossary: Digital Engineering," DAU, 2017, https://www.dau.edu/glossary/Pages/Glossary.aspx.
DoD Centralized Artifact Repository (DCAR)	A repository holding the hardened container images of DevSecOps components that DoD mission software teams can use to instantiate their own CI/CD pipeline. Also holds the hardened containers for base operating systems, web servers, application servers, databases, Application Performance Interface (API) gateways, and message busses for use by DoD mission software teams as a mission system deployment baseline. These hardened containers, along with security accreditation reciprocity, greatly simplify and speed the process of obtaining an Approval to Connect (ATC) or Authorization to Operate (ATO).
Embedded Software	Software with a dedicated function within a larger mechanical or electrical system, often with real-time computing constraints, or software applications embedded in a platform (e.g., air vehicle, ground vehicle, or ship).
End User	The persons who will ultimately use the software solution. The term end user thus distinguishes the user for which the product is designed from other users who are making the product possible for the end user. Often, the term user would suffice. In the context of Agile/DevSecOps end users convey operational concepts, requirements, and needs, participate in continuous testing activities, and provide feedback on developed capabilities.
Enterprise Services	A term for services that have the proper scope to play a productive role in automating business processes in enterprise computing, networking, and data services. Enterprise services include technical services such as Cloud infrastructure, software development pipeline platforms, common containers/virtual machines, monitoring tools, and test automation tools. Responsibility for these functions is generally above the Program Manager.
Environment	A runtime boundary within which a software component may be deployed and executed. Typical environments include development, integration, test, pre-production, and production.

Term	Definition
Factory	*Software Factory:* A software assembly plant that contains multiple pipelines, which are equipped with a set of tools, process workflows, scripts, and environments, to produce a set of software-deployable artifacts with minimal human intervention. It automates the activities in the develop, build, test, release, and deliver phases. The software factory supports multi-tenancy. *Software Factory Artifact Repository:* A collection of artifacts pulled from DCAR as well as locally developed artifacts to be used in DevSecOps processes. The artifacts include, but are not limited to, virtual machine (VM) images, container images, binary executables, archives, and documentation. It supports multi-tenancy. A program could have separate artifact repositories to store local artifacts and released artifacts. It is also possible to have a single artifact repository and use tags to distinguish the contents.
Feature	A service or distinguishing characteristic of a software item (e.g., performance, portability, or functionality) that fulfills a stakeholder need and includes benefit and acceptance criteria within one release. Features are used to complete capabilities and are composed of multiple stories (or tasks, use cases, etc.).
Government Developmental Testing	Testing conducted by the Government to verify and demonstrate how well the system under development meets its technical compliance requirements, to provide data to assess developmental risk for decision making, and to ensure that the technical and support problems identified in previous testing have been corrected.
Human Systems Integration	The management and technical discipline of planning, enabling, coordinating, and optimizing all human-related considerations during system design, development, test, production, use and disposal of systems, subsystems, equipment, and facilities. Source: https://www.sebokwiki.org/wiki/Human_Systems_Integration
Hypervisor	A program used to run and manage one or more virtual machines on a computer. A hypervisor allows one host computer to support multiple guest VMs by virtually sharing its resources, such as memory and processing. Source: https://vmware.com/topics/glossary/content/hypervisor.html

Term	Definition
Immutable Infrastructure	An infrastructure paradigm in which servers are never modified after they are deployed. If something needs to be updated, fixed, or modified in any way, new servers built from a common image with the appropriate changes are provisioned to replace the old ones. After they are validated, they are put into use and the old ones are decommissioned. The benefits of an immutable infrastructure include more consistency and reliability in the infrastructure and a simpler, more predictable deployment process. It mitigates or entirely prevents issues that are common in mutable infrastructures, like configuration drift and snowflake [no two alike] servers. However, using it efficiently often requires comprehensive deployment automation, fast server provisioning in a cloud-computing environment, and solutions for handling state or ephemeral data like logs. Source: https://www.digitalocean.com/community/tutorials/what-is-immutable-infrastructure
Infrastructure as Code	The management of infrastructure (networks, virtual machines, load balancers, and connection topology) in a descriptive model, using the same versioning that the DevSecOps team uses for source code. Infrastructure as Code evolved to solve the problem of environment drift in the release pipeline.
Kubernetes	An open source system for automating deployment, scaling, and management of containerized applications. It was originally designed by Google and is now maintained by the CNCF. Many vendors also provide their own branded Kubernetes. It works with a range of container runtimes. Many Cloud services offer a Kubernetes-based platform as a service.
Lockdown	The closing or removal of weaknesses and vulnerabilities from software.
Microservices	Both an architecture and an approach to software development in which a monolithic application is broken into a suite of loosely coupled independent services that can be altered, updated, or taken down without affecting the rest of the application.
Mission Application Platform	The underlying hosting environment resources and capabilities, plus any mission program enhanced capabilities that form the base upon which the mission software application operates.
Mission Operations	The real-world environment within which the needed military capability is required and must perform.
Model-Based Systems Engineering (MBSE)	Model-based systems engineering (MBSE) is the formalized application of modeling to support system requirements, design, analysis, verification and validation activities beginning in the conceptual design phase and continuing throughout development and later life cycle phases.” Source: (INCOSE 2023)

Term	Definition
Modern Software Development Practices	Practices (e.g., Lean, Agile, DevSecOps) that focus on rapid, iterative development and delivery of software with active user engagements. Small cross-functional sofware development teams integrate planning, design, development, testing, security, delivery, and operations with continuous improvement to maximize automation and user value.
Modularity	The degree to which a system's components may be separated and recombined, often with the benefit of flexibility and varity in use. Source: (Farley, Modern Software Engineering 2022) page 105
Monolithic	A monolithic application is a single-tiered software application in which the user interface and data access code are combined into a single program from a single platform.
Minimum Viable Capability Release (MVCR)	The initial set of features suitable to be fielded to an operational environment that provides value to the warfighter or end user in a rapid timeline. The MVCR delivers initial warfighting capabilities to enahance some mission outcomes. The MVCR is analogous to a minimum marketable product in commercial industry. Source: DoDI 5000.87, October 2, 2020
Minimum Viable Product (MVP)	An early version of the software to deliver or field basic capabilities to users to evaluate and provide feedback. Insights from MVPs help shape, scope, requirements, and design. Source: DoDI 5000.87, October 2, 2020
Node (or Cluster Node)	*In the context of CNCF Kubernetes a node* is a worker machine in Kubernetes cluster. The node may be a VM or physical machine, depending on the cluster. Each node contains the services necessary to run pods and is managed by the master components, including the node controller. A node is also referred to as a cluster node.
Open Container Initiative (OCI)	An open governance structure for the express purpose of creating open industry standards around container formats and runtime. Source: https://www.opencontainers.org
OCI-Compliant Container	Container image that conforms with the OCI Image Specification.
OCI-Compliant Container Runtime	Software that executes containers and manages container images on a node. OCI-compliant container runtime must conform with the OCI Runtime Specification.
Operational Acceptance	A decision by one or more military units to use the software in military operations.
Operational Release	A software release that has been approved for operational use.
Operational Users	See End User.
Orchestration	In the context of containerized software, the automated configuration, coordination, and management of containers and container instances to achieve a desired effect.
Platform	A group of resources and capabilities that form a base upon which other software-enabled capabilities or services are built and operated.
Pod	A group of containers that run on the same CNCF Kubernetes worker node and share libraries and OS services.

Term	Definition
Product Owner	An active member of the software development team working closely with the user community to ensure that the requirements reflect the needs and priorities of the user community and align to the mission objectives.
Product Roadmap	A high-level visual summary that maps out the vision and direction of product offerings over time. It describes the goals and features of each software iteration and increment.
Provisioning	Instantiation, configuration, and management of software or the environments that host or contain software.
Release	A collection of one or more new or changed services or service components deployed into a live environment as a result of one or more changes. Source: ISO/IEC 20000-10:2018
Release Burndown	Technique to display publicly the progress of the current release. Typically a release burndown graph is used. The vertical axis shows the work remaining for a release.
Reporting	An account or statement describing an event.
Repository	A central place in which data is aggregated and maintained in an organized way.
Resource	In the context of computer systems, resources are used for computation such as central processing unit, memory, disk, network- connections and bandwidth.
Risk-Based Test Management Approach	An approach that determines the level of testing based on the probability and consequences of system or software failures.
Risk Management Framework (RMF)	A set of standards that enable DoD agencies to effectively manage cybersecurity risk and make more informed, risk-based decisions. Source: NIST 800-53
Scanning	*Scanning Security:* The evaluation of software for cybersecurity weaknesses and vulnerabilities.
Safety Critical	Used in the context of software system safety, a condition, event, operation, process, or item whose mishap severity consequence is either catastrophic or critical (e.g., safety-critical function, safety-critical path, and safety-critical component).
Scrum	A process framework used to manage software product development and other knowledge work. Scrum is empirical in that it provides a means for teams to establish a hypothesis of how they think something works, try it out, reflect on the experience, and make the appropriate adjustments. Source: https://agilealliance.org/glossary
Scrum Master	A role within the Scrum framework. The scrum master ensures the scrum framework is followed. He/she is committed to the scrum values and practices, but should also remain flexible and open to opportunities for the team to improve their workflow. *Source: https://atlassian.com/agile/scrum/scrum-master*

Term	Definition
Service Mesh	A dedicated infrastructure layer that developers can add to applications. It allows developers to transparently add capabilities like observability, traffic management, and security, without adding them to the developer's own code. The term "service mesh" describes both the type of software developers use to implement this pattern, and the security or network domain that is created when developers use that software. Source: https://istio.io
Sidecar	A container used to extend or enhance the functionality of an application container without strong coupling between the two. When using CNCF Kubernetes, a pod is composed of one or more containers. A sidecar is a utility container in the pod. Its purpose is to support the main application container or containers inside the same pod.
Sidecar Container Security Stack	A stack of sidecar containers aimed to enhance the security capabilities of the main containers in the same pod.
Software as a Service (SaaS)	A method of software delivery and licensing in which software is accessed online via a subscription, rather than bought and installed on individual computers.
Software Engineering	Software Engineering may be defined as the systematic design and development of sofware products and the management of the software process. (Mills 1980)
Software Factory	A structured collection of related software assets that aids in producing computer software applications or software components according to specific, externally defined end user requirements through an assembly process. Various types of factories include: embedded, application, safety critical, AI/ML, data (source: DSO CoP, 06-09-2022 DevSecOps CoP Slides – Software Factories FINALv2)
Software-Intensive	A system in which software represents the largest segment in one or more of the following criteria: system development cost, system development risk, system functionality, or development time.
Sponsor	The organization that holds the authority and advocates for needed user capabilities and associated resource commitments. The sponsor is typically the Military Service or Command that will use the capability once delivered.
Sprint	A set period of time during which specific work has to be completed and made ready for review. Typically one or two weeks. Agile development projects are carried out in a series of sprints.
Strangler Pattern	A way of migrating a legacy system incrementally by replacing existing functionalities with new applications and services in a phased approach. After the replacement of the entire functionality the new application system eventually replaces all the old legacy system's features. " https://www.castsoftware.com/blog/how-to-use-strangler-pattern-for-microservices-modernization N Natesan, "How to Use Strangler Pattern for Microservices Modernization," *Software Intelligence Pulse,* September 2019.
Security Monitorting	*Security Monitoring*. The regular observation, recording, and presentation of activities.

Term	Definition
Task	Individual activities to be completed to satisfy a user story or use case (e.g., implement code for a specific feature or complete design for a specific feature).
Technical Debt	Design or implementation decisions that are expedient in the short term but that set up a technical context that can make a future change costlier or impossible. Technical debt may result from having code issues related to architecture, structure, duplication, test coverage, comments and documentation, potential bugs, complexity, coding practices, and style, which may accrue at the level of overall system design or system architecture, even in systems with great code quantity.
Telemetry	The process of recording system behavior; including the capability to take measurements, collect, and distribute the data.
Test Driven Development (TDD)	Test driven development is an iterative, fine grained approach to coding -- far more fine-grained that sprints. It treats test programs as executable specification of program behavior that written before the code. “It is often described the practices that contribute to it: Red Green Refactor. • Red: Write a test, run it, and see it fail • Green: Write just enough code to to make the test pass, run it and see it pass. • Refactor: Modify the code and the test to make it clear, expressive, elegant, and more general. Run the test after every tiny change and see it pass.” Source (Farley, Modern Software Engineering 2022)
Use Case	In software and systems engineering, a list of actions or event steps, typically defining the interactions between a user and a system (or between software elements) to achieve a goal. Use cases can be used in addition to or in lieu of user stories.
User Agreement (UA)	A commitment between the sponsor and the PM for continuous user involvement and assigned decision making authority in the development of and delivery of software capability releases.
User Story	A small desired behavior of the system based on a user scenario that can be implemented and demonstrated in one iteration. A story is composed of one or more tasks. In software development and product management, a user story is an informal, natural language description of one or more features of a software system. User stories are written from the perspective of a user of a system.
Value Assessment	An outcome-based assessment of mission improvements and efficiencies realized from the delivered software capabilities, and a determination of whether the outcomes have been worth the investment. The sponsor and user community perform value assessments at least annually, to inform DA and PM decisions.
Virtual Machine (VM)	A machine that appears to the applications running on it to be dedicated hardware, but which is in fact hardware shared by other applications. Software termed a hypervisor manages the sharing of hardware between virtual machines. The hypervisor manages this in such as way that the vast majority of machine instructions run directly on the base hardware.

Term	Definition
Virtual Network	Network constructed of software-defined devices.
Virtual Storage	Storage constructed of software-defined devices.
Zero Trust	A security framework requiring all users, whether in or outside the organization's network, to be authenticated, authorized, and continuously validated for security configuration and posture before being granted or keeping access to applications and data. Zero Trust assumes that there is no traditional network edge; networks can be local, in the cloud, or a combination or hybrid with resources anywhere as well as workers in any location

A complete glossary of acquisition terms is maintained on the Defense Acquisition University website. The Defense Acquisition University Glossary can be found at https://www.dau.edu/tools/t/DAU-Glossary

Acronyms

AAF	Adaptive Acquisition Framework
AI	Artificial Intelligence
API	Application Programming Interface
ATC	Approval to Connect
ATO	Authorization to Operate
BOM	Bill of Materials
CAE	Component Acquisition Executive
CAPE	Cost Assessment and Program Evaluation
cATO	Continuous Authorization to Operate
CD	Continuous Delivery
CI	Continuous Integration
CID	Continuous Iterative Development
CIO	Chief Information Officer
CNCF	Cloud Native Computing Foundation
CNS	Capability Needs Statement
COTS	Commercial Off-the-Shelf
CPU	Central Processing Unit
DA	Decision Authority
DCAR	DoD Centralized Artifact Repository
DE	Digital Engineering
DevSecOps	Development, Security, and Operations
DIB	Defense Innovation Board
DoD	Department of Defense
DoDD	DoD Directive
DoDI	DoD Instruction
DORA	DevOps Research and Assessment
DOT&E	Director, Operational Test and Evaluation
DSB	Defense Science Board
EMD	Engineering and Manufacturing Development
EO	Executive Order
ESLOC	Equivalent Source Lines of Code

FQT	Full Qualification Test
FY	Fiscal Year
GOTS	Government Off-the-Shelf
HSI	Human Systems Integration
HTTP	Hypertext Transfer Protocol
IaaS	Infrastructure as a Service
INCOSE	International Council on Systems Engineering
IP	Intellectual Property
IT	Information Technology
MBSE	Model-Based Systems Engineering
MDA	Milestone Decision Authority
MDAP	Major Defense Acquisition Program
MOSA	Modular Open Systems Approach
MTTR	Mean Time to Restore
MVCR	Minimum Viable Capability Release
MVP	Minimum Viable Product
NDS	National Defense Strategy
NIST	National Institute of Standards and Technology
OCI	Open Container Initiative
OS	Operating System
OSS	Open-Source Software
OT	Operational Testing
OUSD(A&S)	Office of the Under Secretary of Defense for Acquisition and Sustainment
OUSD(R&E)	Office of the Under Secretary of Defense for Research and Engineering
PEO	Program Executive Officer
PI	Program Increment
PIT	Platform Information Technology
PM	Program Manager
PMO	Program Management Office
PSM	Practical Software and Systems Measurement (Group)
RMF	Risk Management Framework
SaaS	Software as a Service

SaC	Security as Code
SI&T	System Integration and Test
SLOC	Source Lines of Code
SPO	System Program Office
SW-ICD	Software Initial Capabilities Document
TDD	Test Driven Development
TMRR	Technology Maturation and Risk Reduction
UA	User Agreement
USC	United States Code
USD(A&S)	Under Secretary of Defense for Acquisition and Sustainment
USD(P&R)	Under Secretary of Defense for Personnel and Readiness
USD(R&E)	Under Secretary of Defense for Research and Engineering
VM	Virtual Machine

References

AFCEA.org. 2021. "Army's New Tactical Network Faces First Test in 'Crucible of Combat'." *Federal News Network.* April 5. Accessed June 1, 2021. https://federalnewsnetwork.com/army/2021/04/armys-new-tactical-network-faces-first-test-in-crucible-of-combat/.

ATC. 2017. *Report to the President on Federal IT Modernization.* Washington, D.C.: American Technology Council (ATC), Office of the President. https://www.cio.gov/assets/resources/Report-to-the-President-on-IT-Modernization-Final.pdf.

Brosseau, Daniel, Sherina Ebrahim, Christopher Handscomb, and and Shail Thaker. 2019. *The Journey to an Agile Organization.* Washington, D.C.: McKinsey & Co. https://www.mckinsey.com/business-functions/organization/our-insights/the-journey-to-an-agile-organization.

CAPE. 2020. *DoD Cost Estimating Guide.* Washington, D.C.: Department of Defense (DoD) Cost Assessment and Program Evaluation (CAPE) . https://www.cape.osd.mil/files/Reports/DoD_CostEstimatingGuidev1.0_Dec2020.pdf.

—. 2022. *Unified Code Counter: Government.* Department of Defense Cost Estimation and Program Evaluation (CAPE). Accessed August 19, 2022. https://cade.osk.mil/tools/unifiedcodecounter.

Chaillan, Nicolas. 2020. "From Waterfall to DevSecOps." https://software.af.mil/dsop/documents/.

CJCSI 3162.02. 2019. *CJCSI 3162.02, Methodology for Combat Assessment.* Washington, D.C.: Chairman of the Joint Chiefs of Staff (JCS).

CJCSI 8510.01C. 2012. *CJCS Instruction 8510.01C, Management of Modeling and Simulation.* Washington, D.C.: Chairman of the Joint Chiefs of Staff (JCS) J-8. https://www.jcs.mil/Portals/36/Documents/Library/Instructions/8510_01.pdf?ver=-7XBK8wXzkM9LoRqAWD10A%3d%3d.

CNSSI 4009. 2015. *CNSSI No. 4009, Committee on National Security Systems (CNSS) Glossary.* Fort Meade, Md.: CNSS, National Security Agency.

Conway, Melvin E. 1968. "How Do Committees Invent." *Datamation*, April: 28-31. http://www.melconway.com/Home/pdf/committees.pdf.

Costello, Katie. 2019. "The Secret to DevOps Success." *Smarter with Gartner*, April 11. https://www.gartner.com/smarterwithgartner/the-secret-to-devops-success/.

Crisman, Dr. Jill. 2020. "Artificial Intelligence, OUSD (R&E) AI Software Roadmap." *Presentation for OUSD (R&E) Brown Bag Series.*

DAG 3-2.3.1. 2017. "Software." In *Defense Acquisition Guidebook (DAG)*. Fort Belvoir, Va.: Defense Acquisition University (DAU). https://www.dau.edu/pdfviewer?Guidebooks/DAG/DAG-CH-3-Systems-Engineering.pdfaspx.

DAG 3-2.4.1. 2017. "Modular Open Systems Approach." In *Defense Acquisition Guidebook (DAG)*. Fort Belvoir, Va.: Defense Acquisition University (DAU). https://www.dau.edu/pdfviewer?Guidebooks/DAG/DAG-CH-3-Systems-Engineering.pdfaspx.

DAU AAF. 2020. *Adaptive Acquisition Framework (AAF) Pathways.* Defense Acquisition University (DAU). Accessed 2022. https://aaf.dau.edu/aaf/aaf-pathways/.

DAU AAF Software. 2022. *Software Acquisition.* Defense Acquisition University (DAU). Accessed 2022. https://aaf.dau.edu/aaf/software/.

DAU DevSecOps. 2022. *Defense Acquisition University (DAU) DevSecOps Academy.* Accessed Nov 6, 2022. https://www.dau.edu/cop/it/Pages/Topics/DevSecOps.aspx.

DAU JCIDS Primer. 2019. *Joint Capabilities Integration and Development System (JCIDS): A Primer.* Fort Belvoir, Va.: Defense Acquisition University (DAU). https://myclass.dau.edu/bbcswebdav/institution/Courses/Rapid%20Deployment/JCIDS%20Primer.

DAU Metrics and Reporting. 2022. *Metrics and Reporting.* Defense Acquisition University (DAU). Accessed March 29, 2023. https://aaf.dau.edu/aaf/software/metrics-and-reporting/.

DAU Selecting Pathways. 2022. *Selecting and Transitioning Pathways.* Accessed 2022. https://aaf.dau.edu/aaf/selecting-a-pathway/.

DCWF. 2022. *DoD Cyber Workforce Framework (DCWF).* DoD Chief Information Officer. https://public.cyber.mil/cw/dcwf/.

DIB. 2019a. *Software Acquisition and Practice (SWAP) Study.* Washington, D.C.: Defense Innovation Board (DIB), Department of Defense. https://innovation.defense.gov/software/.

DIB. 2019b. *Software Is Never Done: Refactoring the Acquisition Code for Competitive Advantage.* Washington, D.C.: Defense Innovation Board (DIB), Department of Defense. Accessed 11 21, 2019. innovation.defense.gov/software.

DoD CIO. 2022. *DCWF Orientation.* DoD Chief Information Officer (CIO) Cyber Workforce Management Directorate, Washington, D.C.: DoD Cyber Exchange, Defense Information Systems Agency. https://public.cyber.mil/training/dcwf-orientation/.

DoD CIO. 2018. *DoD Cloud Strategy.* Washington, D.C.: DoD Chief Information Officer (CIO). https://media.defense.gov/2019/Feb/04/2002085866/-1/-1/1/DOD-CLOUD-STRATEGY.PDF.

DoD CIO. 2020. *DoD Data Strategy.* Washington, D.C.: DoD Chief Information Officer (CIO). https://media.defense.gov/2020/Oct/08/2002514180/-1/-1/0/DOD-DATA-STRATEGY.PDF.

DoD CIO. 2019a. *DoD Digital Modernization Strategy.* Washington, D.C.: DoD Chief Information Officer (CIO). https://media.defense.gov/2019/Jul/12/2002156622/-1/-1/1/DOD-DIGITAL-MODERNIZATION-STRATEGY-2019.PDF.

DoD CIO. 2019b. *DoD Enterprise DevSecOps Reference Design, Version 1.0.* Washington, D.C.: Department of Defense (DoD) Chief Information Officer (CIO).

—. 2021. *DoD Open Source Software FAQ.* Department of Defense (DoD) Chief Information Officer (CIO). October 28. https://dodcio.defense.gov/open-source-software-faq/.

DoD CIO DSOERDK . 2021. *DoD Enterprise DevSecOps Reference Design: Multi-Cluster CNCF Kubernetes.* Washington, D.C.: DoD Chief Information Officer (CIO). Accessed August 22, 2022. https://dodcio.defense.gov/Portals/0/Documents/Library/DevSecOpsRefDesign-Multi-ClusterKubernetes.pdf.

DoD. 2020a. *Cybersecurity Test and Evaluation Guidebook 2.0.* Guide, Washington, D.C.: Department of Defense (DoD). Accessed August 13, 2020. https://www.dau.edu/cop/test/DAU%20Sponsored%20Documents/Cybersecurity-Test-and-Evaluation-Guidebook-Version2-change-1.pdf.

DoD. 2020b. *DDR&E Advanced Capabilities: Software Engineering.* Washington, D.C.: Department of Defense (DoD). Accessed August 13, 2020.

—. n.d. *DoD DevSecOps Software Factory Templates.* https://intelshare.intelink.gov/sites/SWMod/DoDDevSecOps/_layouts/15/start.aspx#/DevSecOps%20Software%20Factory%20Inventory/Forms/AllItems.aspx .

—. 2021. *DoD Software Modernization Strategy, Version 1.0.* Washington, D.C.: Department of Defense (November).

DOD DSO Playbook. 2021. "DevSecOps Fundmentals Playbook." *dodcio.defense.gove.* https://dodcio.defense.gov/Portals/0/Documents/Library/DevSecOpsFundamentalsPlaybook.pdf.

DoD Enterprise DevSecOps Portal. 2023. https://intelshare.intelink.gov/sites/SWMod/DoDDevSecOps/_layouts/15/start.aspx#/SitePages/Home.aspx.

DoD. 2018. *Summary of the 2018 National Defense Strategy - Sharpening the American Military's Competitive Edge.* Washington, D.C.: Department of Defense. https://dod.defense.gov/Portals/1/Documents/pubs/2018-National-Defense-Strategy-Summary.pdf.

DoD. 2018. *Summary of the 2018 National Defense Strategy - Sharpening the American Military's Competitive Edge.* Washington, D.C.: Department of Defense.

DoDAIS 2018. 2018. "Summary of the 2018 Department of Defense Artificial Intelligence Strategy - Harnessing AI to Advance Our Security and Prosperity." https://media.defense.gov/2019/Feb/12/2002088963/-1/-1/1/SUMMARY-OF-DOD-AI-STRATEGY.PDF.

DoDD 5000.59. 2018. *DoD Directive 5000.59, DoD Modeling and Simulation Management.* Washington, D.C.: Office of the Under Secretary of Defense for Research and Engineering. https://www.esd.whs.mil/Portals/54/Documents/DD/issuances/dodd/500059p.pdf.

DoDD 5100.01. 2020. *DoD Directive 5100.01, Functions of the Department of Defense and Its Major Components.* Washington, D.C.: Secretary of Defense. https://www.esd.whs.mil/Portals/54/Documents/DD/issuances/dodd/510001p.pdf?ver=2020-09-17-154337-967.

DoDD 5142.02. 2008. *Under Secretary of Defense for Personnel adn Readiness (USD(P&R)).* Department of Defense.

DoDD 8140.01. 2020. *DoD Directive 8140.01, Cyberspace Workforce Management.* Washington, D.C.: Deputy Secretary of Defense. https://www.esd.whs.mil/Portals/54/Documents/DD/issuances/dodd/814001p.pdf.

DoDI 1400.25 Volume 250. 2016. "DoD Civilian Personnel Management System: Civilian Strategic Human Capital Planning (SCHP)."

DoDI 5000.02. 2022. *DoD Instruction 5000.02, Operation of the Adaptive Acquisition Framework, Change 1.* Office of the Under Secretary of Defense for Acquisition and Sustainment (June). https://www.esd.whs.mil/Portals/54/Documents/DD/issuances/dodi/500002p.pdf?ver=2020-01-23-144114-093.

DoDI 5000.85. 2020. *DoD Instruction 5000.85, Major Capability Acquisition.* Washington, D.C.: Office of the Under Secretary of Defense for Acquisition and Sustainment. http://acqnotes.com/wp-content/uploads/2020/08/DoD-Instruction-5000.85-Major-Capability-Acquisition-6-Aug-2020.pdf.

DoDI 5000.87. 2020. "Operation of the Software Acquisition Pathway." Office of the Under Secretary of Defense for Acquistion and Sustainment, October 2. Accessed August 17, 2022. https://www.esd.whs.mil/Portals/54/Documents/DD/issuances/dodi/500087p.PDF.

DoDI 5000.88. 2020. *DoD Instruction 5000.88, Engineering of Defense Systems.* Washington, D.C.: Office of the Under Secretary of Defense for Research and Engineering. https://www.esd.whs.mil/Portals/54/Documents/DD/issuances/dodi/500088p.PDF?ver=O8LFc8NzlyJX-SgM2Haalw%3d%3d.

DoDI 5000.89. 2020. *DoD Instruction 5000.89, Test and Evaluation.* Washington, D.C.: Office of the Under Secretary of Defense for Research and Engineering and Office of the Director, Operational Test and Evaluation. https://www.esd.whs.mil/Portals/54/Documents/DD/issuances/dodi/500089p.PDF?ver=Plc85E0-NVNide91K3XQLA%3D%3D.

DoDI 5000.95. 2022. *DoD Instruction 5000.95, Human Systems Integration in Defense Acquisition.* Washington, D.C.: Office of the Under Secretary of Defense for Research and Engineering. https://www.esd.whs.mil/Portals/54/Documents/DD/issuances/dodi/500095p.PDF?ver=C1L4ZM9Wi4Qa4p7JP7EPtA%3D%3D.

DoDI 8510.01. 2020. *DoD Instruction 8510.01, Risk Management Framework (RMF) for DoD Information Technology (IT).* Washington, D.C.: DoD Chief Information Officer . https://www.esd.whs.mil/Portals/54/Documents/DD/issuances/dodi/851001p.pdf?ver=qEE2HGN_HE4Blu7161t1TQ%3D%3D.

DORA. 2022. *DevOps Research and Assessment (DORA) Website.* Accessed August 19, 2022. https://devops-research.com/research.html#reports.

DSB. 2018. *Design and Acquisition of Software for Defense Systems.* Washington, D.C.: Defense Science Board (DSB), Office of the Under Secretary of Defense for Research and Engineering. https://dsb.cto.mil/reports/2010s/DSB_SWA_Report_FINALdelivered2-21-2018.pdf.

DSOF. 2021. *DoD Enterprise DevSecOps Fundamentals (DSOF): Glossary.* Washington, D.C.: Department of Defense (DoD) Chief Information Officer (CIO).

DSOP. 2022. *DoD Enterprise DevSecOps Initiative (DSOP).* Washington, D.C.: U.S. Air Force, https://software.af.mil/dsop/.

EO 14028. 2021. *Executive Order (EO) 14028, Improving the Nation's Cybersecurity.* Washington, D.C.: The White House. https://www.whitehouse.gov/briefing-room/presidential-actions/2021/05/12/executive-order-on-improving-the-nations-cybersecurity/.

Farley, David. 2022. *Modern Software Engineering.* Boston: Addison-Wesley.

—. 2022. *Modern Software Engineering.* Boston: Addison-Wesley.

FITARA. 2014. *Federal Information Technology Acquisition Reform Act.* 113th Congress (2013-2014), Washington, D.C.: Government Publishing Office. https://www.congress.gov/bill/113th-congress/house-bill/1232.

Forbes Technology Council. 2019. "Three Engineering Performance Metrics That Business Can Understand." *Forbes*, August 5. Accessed October 2020. https://www.forbes.com/sites/forbestechcouncil/2019/08/05/three-engineering-performance-metrics-the-business-can-understand/#72c68a3b704d.

GAO. 2020. *Agile Assessment Guide: Best Practices for Agile Adoption and Implementation.* U.S. Government Accountability Office, Washington, D.C.: U.S. Government Accountability Office. https://www.gao.gov/assets/710/709711.pdf.

GAO. 2019a. *Space Command and Control: Comprehensive Planning and Oversight Could Help DoD Acquire Critical Capabilities and Address Challenges.* Washington, D.C.: Government Accountability Office (GAO). https://www.gao.gov/products/GAO-20-146.

GAO. 2019b. *Weapon System Sustainment, DoD Needs to Better Capture and Report Software Sustainment Costs.* Washington, D.C.: Government Accountability Office (GAO). Accessed June 2, 2021. https://www.gao.gov/assets/gao-19-173.pdf.

Greenemeier, Larry. 2017. "20 Years after Deep Blue: How AI Has Advanced Since Conquering Chess." *Scientific American*, June 7. Accessed August 18, 2022. https://www.scientificamerican.com/article/20-years-after-deep-blue-how-ai-has-advanced-since-conquering-chess/.

Humble, Jez, and David Farley. 2011. *Continuous Delivery.* Boston: Addison-Wesley.

INCOSE. 2023. *MBSE Initiative.* Accessed March 30, 2023. https://www.incose.org/incose-member-resources/working-groups/transformational/mbse-initiative.

Intellipedia Secure Coding Guidelines. 2022. *Secure Coding Guidelines (CAC Required).* Intelink (CAC Required). https://intellipedia.intelink.gov/wiki/Secure_Coding_Guidelines.

ISO 9241-210. 2010. "ISO 9241-210:2010. Ergonomics of human-system interaction — Part 210: Human-centred design for interactive systems."

ISO/IEC/IEEE DIS 24641. 2021. "DIS 24641:2021(E) System and Software Engineering: Methods and Tools for Model-Based Systems and Software Engineering." Accessed 11 5, 2022. https://www.iso.org/standard/79111.html.

JAIC. 2020. *Understanding AI Technology.* Department of Defense (DoD) Joint Artificial Intelligence Center (JAIC).

Jaiswal, Sonoo. n.d. *Difference between Microkernel and Monlithic Kernel.* Accessed November 16, 2022. https://www.javatpoint.com/microkernel-vs-monolithic-kernel.

JCS. 2016. *Cross-Domain Synergy in Joint Operations: Planner's Guide.* Washington, D.C.: Joint Chiefs of Staff (JCS) Joint Force Development (J7). https://www.jcs.mil/Portals/36/Documents/Doctrine/concepts/cross_domain_planning_guide.pdf?ver=2017-12-28-161956-230.

JCS. 2018. *Joint Concept for Integrated Campaigning (JCIC).* Washington, D.C.: Joint Chiefs of Staff (JCS). https://www.jcs.mil/Portals/36/Documents/Doctrine/concepts/joint_concept_integrated_campaign.pdf?ver=2018-03-28-102833-257.

—. n.d. *Joint Doctrine Publications.* Joint Chiefs of Staff (JCS). https://www.jcs.mil/Doctrine/Joint-Doctine-Pubs/.

JCS. 2020. *Joint Planning.* Washington, D.C.: Joint Chiefs of Staff (JCS) Joint Publications 5-0. https://www.jcs.mil/Portals/36/Documents/Doctrine/pubs/jp5_0.pdf?ver=ztDG06paGvpQRrLxThNZUw%3d%3d.

JCS NMS. 2018. *National Military Strategy 2018.* Department of Defense, Washington, D.C.: Joint Chiefs of Staff (JCS) Directorate for Strategy, Plans, and Policy (J-5). https://www.jcs.mil/Portals/36/Documents/Publications/UNCLASS_2018_National_Military_Strategy_Description.pdf.

JDMS. 2021. *Journal of Defense Modeling and Simulation: Applications, Methodology, Technology - Special Issue: Modeling and Simulation as a Service.* Vol 18, Issue 1. Society for Modeling and Simulation International, SAGE Journals. https://journals.sagepub.com/toc/dms/current#sage_toc_section_SpecialsectiononTransformingtheengineeringenterpriseSpecialsectionarticles.

Kan, Stephen. 2003. *Metrics and Models in Software Quality Engineering.* Boston: Addison-Wesley Professional.

Kim, Gene, Jez Humble, John Willis, and and Patrick Debois. 2016. *The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations.* Portland, Ore.: IT Revolution Press. https://itrevolution.com/book/the-devops-handbook.

Mills, Harlan D. 1980. "The Management of Software Engineering, Part I: Principles of Software Engineering." *IBM Systems Journal* (IBM) 19 (4).

MITRE. 2019. *A New Battle Command Architecture for Multi-Domain Operations: Countering Peer Adversary Power Projection.* McLean, Va.: MITRE Center for Technology and National Security. https://www.mitre.org/sites/default/files/publications/Joint-All-Domain-Command-Control.pdf.

NDAA 230. FY 2020. *National Defense Authorization Act (NDAA) for Fiscal Year 2020, Section 230, Policy on the Talent Management of Digital Expertise and Software Professionals.* P.L. 92, 116th Congress, Washington, D.C.: Government Publishing Office. https://www.govinfo.gov/content/pkg/PLAW-116publ92/html/PLAW-116publ92.htm.

NDAA 868c. FY 2019. *National Defense Authorization Act (NDAA) for Fiscal Year 2019, Section 868(c), Implementation of Recommendations of the Final Report of the Defense Science Board Task Force on the Design and Acquisition of Software for Defense Systems.* P.L. 115-232, 115th Congress, Washington, D.C.: Government Publishing Office. https://www.govinfo.gov/content/pkg/PLAW-115publ232/html/PLAW-115publ232.htm.

NDAA. FY 2017. *National Defense Authorization Act (NDAA) for Fiscal Year 2017.* S.2943, 114th Congress, Washington, D.C.: Government Publishing Office, December 23, 2016. https://www.congress.gov/bill/114th-congress/senate-bill/2943/text.

NDS Fact Sheet. 2022. *Summary of the 2022 National Defense Strategy.* Washington, D.C.: Secretary of Defense.

Newman, Sam. 2019. *Monolith to Microservices: Evolutionary Patterns to Transform Your Monolith.* Online: O'Reilly Media, Inc. https://www.oreilly.com/library/view/monolith-to-microservices/9781492047834/.

NIST. 2022. *NIST Special Publication 800-218. Secure Software Development Framework (SSDF) Version 1.1.* NIST. https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-218.pdf.

NIST SP 1800-16. 2020. *Securing Web Transacvtions, TLS Server Certificate Management.* National Institute of Standards and Technology. Accessed March 29, 2023. http://doi.org/10.6028/NIST.SP.1800-16.

NIST SP 800-190. 2017. *Application Container Security Guide.* National Insitute of Standards and Technology.

NIST SP 800-207. 2020. *NIST Special Publication 800-207. NIST Guidance on Zero Trust Architecture.* Gaithersburg, Md.: National Institute of Standards and Technology. https://doi.org/10.6028/NIST.SP.800-207.

NIST SP 800-53. 2020. *SP 800-53, Rev. 5, Security and Privacy Controls for Information Systems and Organizations.* Gaithersburg, Md.: National Institute of Standards and Technology (NIST), Department of Commerce. https://csrc.nist.gov/publications/detail/sp/800-53/rev-5/final.

NISTIR 8269 (draft). 2019. *A Taxonomy and Terminology of Adversarial Machine Learning. .* Gaithersburg, Md.: National Institute Standards and Technology. Accessed August 18, 2022. https://csrc.nist.gov/publications/detail/nistir/8269/draft.

Northen, Carlton, Dr. Kathleen Mayfield, Robert Benito, and Michelle Casagni. 2020. *Handbook for Implementing Agile in Department of Defnse Information TechnologyAcquisition.* MITRE Technical Report, McLean, Va.: The MITRE Corporation.

NSS. 2017. *National Security Strategy of the United States of America.* Washington, D.C.: White House. https://www.whitehouse.gov/wp-content/uploads/2017/12/NSS-Final-12-18-2017-0905.pdf.

NTIA SBOM. 2021. *Software Bill of Materials.* National Telecommunications and Information Administration, Department of Commerce (NTIA). https://ntia.gov/page/software-bill-materials.

OMB. 2019. *Federal Cloud Computing Strategy: From Cloud First to Cloud Smart Strategy.* Chief Information Officers (CIO) Council, Office of Management and Budget. https://cloud.cio.gov/strategy/.

OUSD(A&S). 2020. *Software Acquisition Pathway Interim Policy: Procedures.* Memorandum, Washington, D.C.: Under Secretary of Defense for Acquisition and Sustainment OUSD(A&S). Accessed August 13, 2020. https://www.acq.osd.mil/ae/assets/docs/USA002825-19%20Signed%20Memo%20(Software).pdf.

OUSD(R&E). 2018. *DoD Digital Engineering Strategy.* Washington, D.C.: Office of the Under Secretary of Defense for Research and Engineering (OUSD(R&E)).

OUSD(R&E). 2022. *Engineering of Defense Systems Guidebook.* Washington, D.C.: Office of the Under Secretary of Defense for Research and Engineering.

OUSD(R&E). 2020. *Mission Engineering Guide.* Office of the Under Secretary of Defense for Research and Engineering (OUSD(R&E)).

OUSD(R&E) Software S&T Strategy. 2021. *DoD Software Science and Technology Strategy. In Response to NDAA FY 2020.* Washington, D.C.: Office of the Under Secretary of Defense for Research and Engineering (OUSD(R&E)).

Pratt and Whitney. 1983. *Weibull Analysis Handbook.* ADA 143100, West Palm Beach, Fla.: Pratt and Whitney Government Products Division.

PSM. 2022. "Continuous Iterative Development (Agile) Measurement Framework." Vers. 2.1. *Practical Software and Systems Measurement (PSM).* April 15. Accessed August 19, 2022. http://www.psmsc.com/CIDMeasurement.asp.

Red Hat. 2022. *What Is Site Reliability Engineering.* Accessed August 19, 2022. https://redhat.com/en/topics/devops/what-is-sre.

Robson, Sean, Bonnie L. Triexenberg, Samantha E. Dinicola, Linsey Polley, John Davis, and Maria C. Lytell. 2020. *Software Acquisition Workforce Initiative for the Department of Defense.* Santa Monica, Calif.: RAND Corporation. Accessed August 5, 2022. https://www.rand.org/t/RR3145.

SAE6906. 2019. "SAE6906, Standard Practice for Human Systems Integration." https://www.sae.org/standards/content/sae6906/.

SEI CERT Coding Standards. 2020. *SEI CERT Coding Standards.* Accessed March 29, 2023. https://wiki.sei.cmu.edu/confluence/display/seccode/SEI+CERT+Coding+Standards.

Serbu, Jared. 2021. "Army's New Tactical Network Faces First Test in 'Crucible of Combat'." *Federal News Network.* April 5. Accessed June 2, 2021. https://federalnewsnetwork.com/army/2021/04/armys-new-tactical-network-faces-first-test-in-crucible-of-combat/.

Space Command and Control. 2020. *FY 2020 Space Command and Control Annual Report to Congressional Defense Committees in Response to NDAA FY20 Sec.1613 Public Law 116-92, Page 12: Challenges Encountered and Lessons Learned.* Space Command and Control.

Tate, David M. 2020. "Software Productivity Trends and Issues." *Defense Acquisition Research Journal* (Defense Acquisition University) 27 (2). https://www.dau.edu/libary/arj/.

Triezenberg, Bonnie L., Jason M. Ward, Jonathan Chait, Devon Hill, Sean Robson, and Jeff Fourman. 2020. *The Composition and Employment of Software Personnel in the U.S. Department of Defense.* Santa Monica, Calif.: RAND Corporation. Accessed August 5, 2022. doi:https://doi.org/10.7249/RRA520-1.

U.S. Army. 2021. "Mission Command Battle Laboratory, MBCL Looks to Future C2 Information Systems to Enable Decision Dominance." April 30. Accessed June 2, 2021. https://www.army.mil/article/245810.

Software Engineering for Continuous Delivery of Warfighting Capability

Executive Director for Systems Engineering and Architecture
Office of the Under Secretary of Defense for Research and Engineering
3030 Defense Pentagon
Washington, DC 20301
https://www.cto.mil
osd.r-e.comm@mail.mil | Attention: Software Engineering Team