Software Developmental Test and Evaluation in DevSecOps Guidebook

January 2025

Office of the Director,
Developmental Test Evaluation and Assessments (DTE&A)

Office of the Under Secretary of Defense
for Research and Engineering

Washington, D.C.

Software Developmental Test and Evaluation in DevSecOps Guidebook

Office of the Director, Developmental Test Evaluation and Assessments (DTE&A)
Office of the Under Secretary of Defense for Research and Engineering
3030 Defense Pentagon
Washington, DC 20301
https://www.cto.mil/dtea
osd.r-e.comm@mail.mil | Attention: Software Engineering Technical Director

Distribution Statement A. Approved for public release. Distribution is unlimited.
DOPSR Case # 25-T-0999

Executive Summary

The Department of Defense (DoD) has developed this guidebook to serve as a resource for the software developmental test and evaluation (DT&E) practitioner. This guidebook expands on general software test and evaluation (T&E) guidance, addressing software development and sustainment in a DevSecOps (development, security, operations) context.

The emphasis of this Guidebook is on how T&E strategy, planning, execution, and analysis must adapt to support the rapid iterative framework of DevSecOps regardless of acquisition pathway. The goal of this guidebook is to:

- Help Government test teams, in a single T&E continuum, plan and execute software intensive and cyber physical system DT&E in a DevSecOps environment; and
- Introduce to the reader how operationally relevant developmental test conditions in a DevSecOps framework support independent OT&E and LFT&E objectives.

This guidebook provides by section: (1) an introduction to the DevSecOps framework; (2) describes the teams, roles, accountability, and authority within a DevSecOps framework and impacts to Government test teams; (3) DevSecOps foundational concepts; (4) DevSecOps phase agnostic continuous activities; (5) specific DT&E activity considerations within the DevSecOps phases; and (6) recommendations for DevSecOps training.

This guidebook emphasizes the following approaches and best practices:

- DT&E engagement in the test activities within each of the DevSecOps phases ensures the system has been built correctly to meet user and stakeholder capability needs.
- Product Team and Government Test Team involvement as performance and operational requirements are developed and refined to ensure they are measurable, testable, and achievable.
- Early and continuous involvement of testing organizations throughout the software development life cycle is crucial to support effective and efficient evaluations to inform the decision authorities and delivery timelines.
- Test Teams should work with the Project Manager and Product Owner to establish clear definitions of "done" for each sprint/iteration.
- Integration of the SW Integrated Test Team (SW ITT) across the lifecycle. Paramount to mission success, the SW ITT should participate in definition of the Minimum Viable

Product (MVP) and Minimum Viable Capability Release (MVCR or equivalent), including testable criteria for fielding and employment.

- In an DevSecOps environment, testing and security are shifted left through automated unit, functional, security, and integration testing.
- DevSecOps stresses speed through incremental delivery of software capability to the user, enabled by automation.
- Maximum sharing, reuse of test results and artifacts, and reciprocity among testing and certification organizations are necessary for program success.
- Transparency and access to test data across Product Team testing, DT&E, and Operational T&E is a foundational element.

This guide ultimately aims to serve as a valuable resource for the DoD's software test and evaluation community, emphasizing integrated robust testing and evaluation practices with a strong focus on security across the entire software development lifecycle.

COLLINS.CHRISTO
PHER.CLAY.10509
67561

Digitally signed by
COLLINS.CHRISTOPHER.CLAY.
1050967561
Date: 2025.01.28 09:14:06 -05'00'

Mr. Christopher C. Collins
Director, Developmental Test Evaluation and Assessments (DTE&A)

Contents

1 Introduction

This guidebook provides focused guidance and recommended practices for early and developmental test and evaluation (T&E) of acquisitions in a development, security, and operations (DevSecOps) context. Early and continuous involvement of testing organizations throughout the software life cycle is crucial to support effective and efficient evaluations and delivery timelines. The T&E emphasis is on integrating, streamlining, and automating testing processes to enable rapid analysis of test data and evaluation of system operational effectiveness, suitability, and survivability. Program success depends on maximum sharing, reciprocity, availability, and reuse of test results and artifacts among testing and certification organizations.

1.1 Purpose and Scope of the Guidebook

The purpose of this guidebook is to help government test teams plan and execute Developmental Test and Evaluation (DT&E) of software-intensive systems in a DevSecOps environment.

This guidebook expands on the general T&E guidance provided by the Test and Evaluation Enterprise Guidebook and emphasizes how the test and evaluation strategy (TES), planning, execution, and analysis must adapt to support the rapid iterative framework of DevSecOps regardless of the acquisition pathway. This guidebook complements the software acquisition pathway-specific guidance in Chapter 5 of the T&E Enterprise Guidebook.

The first release of this guidebook, or minimum viable product (MVP), introduces and discusses the following subject matter:

- **Section 1. Introduction**. The first section introduces the DevSecOps framework (including the Department of Defense (DoD) Chief Information Officer (CIO) DevSecOps Documentation Set); enterprise T&E policy and guidance (including developmental Test and Evaluation as a Continuum (dTEaaC)); and other available guidance (including the Under Secretary of Defense for Acquisition and Sustainment (USD(A&S)) adaptive acquisition framework (AAF)).
- **Section 2. DT&E in DevSecOps – Roles, Teams, Accountability, and Authority**. Adopting Agile and DevSecOps practices for a DoD acquisition program requires an adjustment to the traditional roles and responsibilities that program offices are familiar with under traditional acquisition practices.
- **Section 3. DT&E in DevSecOps – Foundational Concepts**. In DevSecOps, DT&E takes on a specific role within the Software Development Life Cycle (SDLC) that integrates practices throughout the development, testing, and operational phases. DT&E becomes an integral part of the continuous improvement cycle.

- **Section 4. DT&E in DevSecOps – Phase Agnostic Continuous Activities**. In DevSecOps, DT&E is a part of the continuous process that requires integration between development and testing to achieve high product quality.
- **Section 5. DT&E in DevSecOps – DT&E Activity Considerations within the DevSecOps Phases**. DT&E considerations during the DevSecOps phases revolve around integrating robust T&E practices with a strong focus on security across the entire SDLC.
- **Section 6. DT&E in DevSecOps – Training**. For many individuals in DoD, DevSecOps is a new cultural change that is different from the traditional way of doing business. With DevSecOps changing the way in which DoD develops, deploys, and protects software-intensive systems, this new paradigm requires an understanding of the processes, tools, and techniques for T&E.

Planned future versions of this guidebook will include updates based on stakeholder feedback. This input, captured with publication of this guidebook, is provided as use cases in the appendix.

1.2 Scope of Developmental Test and Evaluation Activities

In accordance with DoD Instruction (DoDI) 5000.89, “Test and Evaluation,” DT&E activities support data generation for organic (internal to the program) and independent evaluations. DT&E starts with capability requirements and continues through product development (plan, develop, build, test, and release); the transition to operational test and evaluation (OT&E); delivery; deployment; and operations and support. An understanding of DT&E and OT&E in the requirements and systems engineering (SE) processes sets a foundation for a test-driven design that ensures the delivered capability is mission effective while emphasizing that requirements are measurable, testable, and achievable. Identifying and correcting deficiencies early is less costly than discovering system deficiencies late in the acquisition process.

T&E activities and outputs include the following:

- Setting an operational and technical decision foundation to support the strategic test framework.
- Providing program engineers and decision makers with information to measure progress.
- Identifying issues and maintaining an awareness of how the project team is managing technical debt.
- Characterizing system operational capabilities, linkage to mission effectiveness, and limitations.

- Evaluating product compliance with contractual and technical requirements and the associated mission impact.
- Verifying exit criteria.

All DT&E activities and outputs are directly affected by the rapid pace of DevSecOps in terms of how they are executed and how DT&E activities are integrated into the development process. The following T&E mission activities and outputs are directly affected by a DevSecOps acquisition environment:

- Measuring progress (i.e., providing program engineers and decision makers with information to measure progress).
- Evaluating compliance (i.e., evaluating product compliance with contractual and technical requirements and the associated mission impact).
- Verifying exit criteria.

The rapid pace of DevSecOps necessitates adaptive methods for progress evaluation, compliance with evolving specifications, and interactive development of exit criteria. Developmental testers play a critical role in reviewing and updating these criteria during each iteration to ensure comprehensive testing and adherence to quality standards.

The forthcoming DoDI 5000.DT, "Developmental Test and Evaluation," will expand the definition of DT&E found in DoDI 5000.89 to include the following:

- Non-acquisition and pre-acquisition program technology, system, and capability development starting in mission engineering (ME), science and technology (S&T), prototyping and experimentation (P&E), and any development for eventual insertion into DoD networks, systems, platforms of systems, and systems of systems (SoS).
- Acquisition systems in accordance with DoD Directive (DoDD) 5205.07, "Special Access Program Policy."
- Systems acquired via the Defense Acquisition System (DAS), including abbreviated acquisition programs under the U.S. Navy or U.S. Marine Corps and acquisitions pursuing any AAF pathway in accordance with DoDD 5000.01, "The Defense Acquisition System," and DoDI 5000.02, "Operation of the Adaptative Acquisition Framework."
- Systems in sustainment or post-production systems.

In accordance with DoDI 5000.89 and the forthcoming DoDI 5000.DT, DT&E activities support data generation for independent developmental test (DT) evaluations throughout the product life cycle. These activities help program engineers and decision makers measure progress, identify problems, characterize system capabilities and limitations, evaluate product compliance, and verify exit criteria.

1.3 DevSecOps Framework

DevSecOps is an organizational culture and practice that unifies software development, security, and operations derived from Agile principles. The main characteristic of DevSecOps is to automate, monitor, and apply security at all phases of the software life cycle: plan, develop, build, test, release, deliver, deploy, operate, monitor, and feedback.

The DevSecOps framework emphasizes continuous integration (CI) and continuous delivery (CD) and iterative, incremental development. DevSecOps aims to provide timely and comprehensive information on system performance and risk characterization, promoting a continuum of testing activities that align with model-based engineering practices.

As described in DoD Enterprise DevSecOps Reference Design, the DevSecOps phases are familiar from other software development models, defined below:

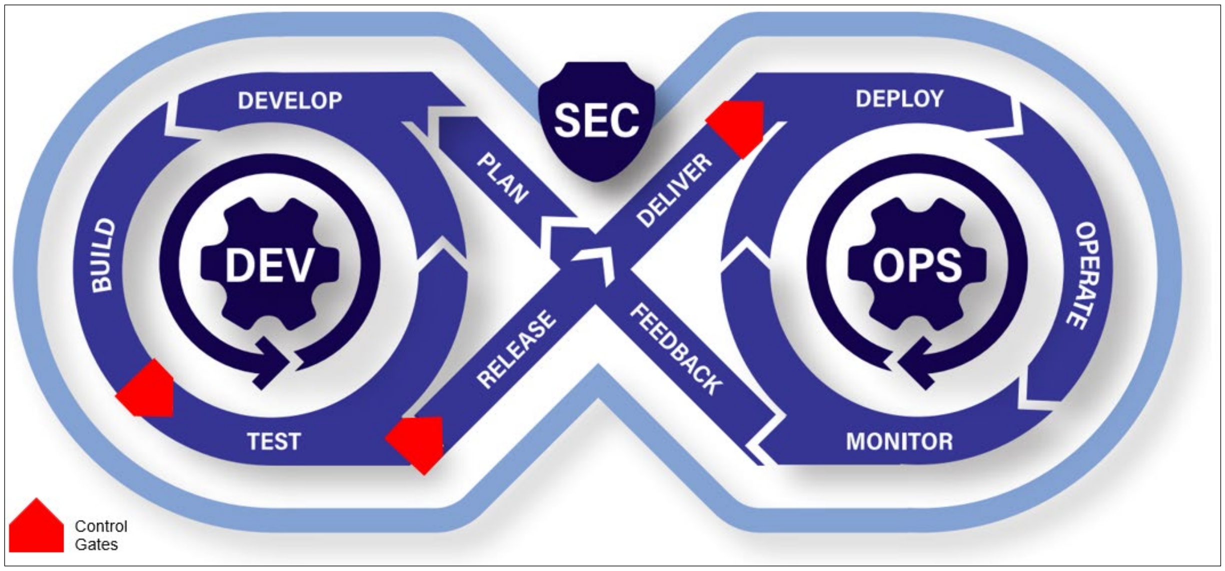
1. **Plan**: Development team members work with stakeholders to create a roadmap for developing, delivering, deploying, and operating products during current and future iterations.
2. **Develop**: Software architects, designers, and coders work to design and develop software that achieves the objectives of the current iteration.
3. **Build**: The software is placed under configuration control in a central repository. There is an expectation that the build phase is completely automated. Software may be packaged in a simulation of the delivered system.
4. **Test**: The software is tested to gather data to determine whether it is ready for release. Although the DevSecOps life cycle includes a designated "test" phase, testing activities occur throughout the life cycle.
5. **Release**: The software is packaged into a product ready to be released from the development environment into production environments.
6. **Deliver**: The packaged product is delivered from the development environment to production environments.
7. **Deploy**: The packaged product is installed in production environments, ready for production use.

8. **Operate**: Operators use the capabilities of the deployed product in performance of their mission.
9. **Monitor**: The product is monitored to ensure it operates correctly, effectively, and efficiently. To the extent possible, the automated tools performing monitoring and the operators are unaware it occurs.
10. **Feedback**: Though not explicitly listed in DoD Enterprise DevSecOps Reference Design, continuous feedback on the software's performance in real-world conditions, allows the product team to identify issues that may not have been apparent during the product development testing activity

In DevSecOps, testing and security are shifted left through automated unit, functional, security, and integration testing. Some forms of testing occur in the develop phase. This approach is a key DevSecOps differentiator since functional and security capabilities are built and tested simultaneously.

1.3.1 DevSecOps Main Characteristics

The DevSecOps life cycle is defined in the DoD Enterprise DevSecOps Fundamentals v2.5 guidance. The DevSecOps life cycle phases are illustrated in Figure 1-1 with nominal control gate overlays added. Each control gate (displayed by the red polygons) represents a feedback mechanism and decision point to move forward to the next DevSecOps phase. Feedback generated from operations (e.g., escaped defects and help desk trouble reports) will inform DT&E to better identify defects before they reach operations.



Source: DoD Enterprise DevSecOps Fundamentals, Version 2.5, p. 3

Figure 1-1. DevSecOps Life Cycle Phases (Infinity Loop) with Nominal Control Gate Overlay

The development team (discussed in Section 2.3.4) can make *lower-level decisions* (between the build and test phases), whereas the product owner (discussed in Section 2.3.2) can make other *higher-level decisions* (between the test and release/deliver phases and between the release/deliver and deploy phases). The decision authority should be based on the program's risk or acquisition profile; for example, on some programs, senior-level personnel will hold the responsibility to make release/deliver and deploy decisions. Government test teams (discussed in Section 2.4) will support criteria development for the lower-level control gates and will plan and conduct testing to support the higher-level control gates.

To reiterate, in DevSecOps, testing and security are shifted left through automated unit, functional, security, and integration testing—a key DevSecOps differentiator because security and functional capabilities are built and tested simultaneously. DevSecOps stresses speed through incremental delivery of software capability to the user; enabled by automation, including testing, monitoring, and applying security at all stages of the software life cycle. *The desire for speed of capability release (CR) must not sacrifice the delivery of quality and secure software*.

Note: though manual cyber T&E is not currently addressed in this Guidebook, it is important to include manual cyber testing in all phases of DevSecOps. More information on manual cyber T&E will be provided in the forthcoming Version 3.0 of the DoD Cyber DT&E Guidebook.

1.3.2 Iterative and Incremental Development

Agile software development is one approach for iterative and incremental development. It emphasizes early and continuous software delivery and is defined by values and principles that can be realized through a set of common practices seen in specific Agile frameworks, such as DevSecOps, eXtreme Programming, Lean, Kanban, Scrum, and others. Agile frameworks are also used to develop hardware programs and manage services. The best practices in this guidebook are intended to be applicable with any incremental development paradigm, regardless of what type of product or service is being delivered.

1.3.3 Continuous Integration and Continuous Delivery

DevSecOps makes use of software practices that emphasize automation supporting a CI/CD pipeline. Note that in some systems (e.g., classified), the CI/CD pipeline may not be fully automated.

DevSecOps incorporates automation and monitoring throughout software development and delivery, including software integration, testing, release, and management of the infrastructure. DevSecOps incorporates security as code (SaC), integrating security into the SDLC through policy and automation (see Section 4.6.4 for SaC details). DevSecOps also advocates short development cycles with frequent deployments and releases. This approach allows development to be informed by operational use of previous iterations. It acknowledges the interdependence of software development, quality assurance (QA), systems integration, and operations. The approach aims to help an organization rapidly integrate software components and services, enhance software quality, recover quickly from system failure, and improve operations performance.

Software development and operations may take place in an end-to-end system architecture with separate development, test, pre-production (staging), and production environments. Figure 1-2 illustrates a notional T&E life cycle, *noting various test activities* that occur in support of control gate decisions. The blue arrows in Figure 1-2 indicate the test input to the decision. Decision criteria and decision ownership will be specified in the Test and Evaluation Master Plan (TEMP), TES documentation, or determined by the needs of the P&E process in pre-program of record activities. The process illustrated in Figure 1-2 is idealized and may not be fully applicable or automatable based on classification or other system requirements.

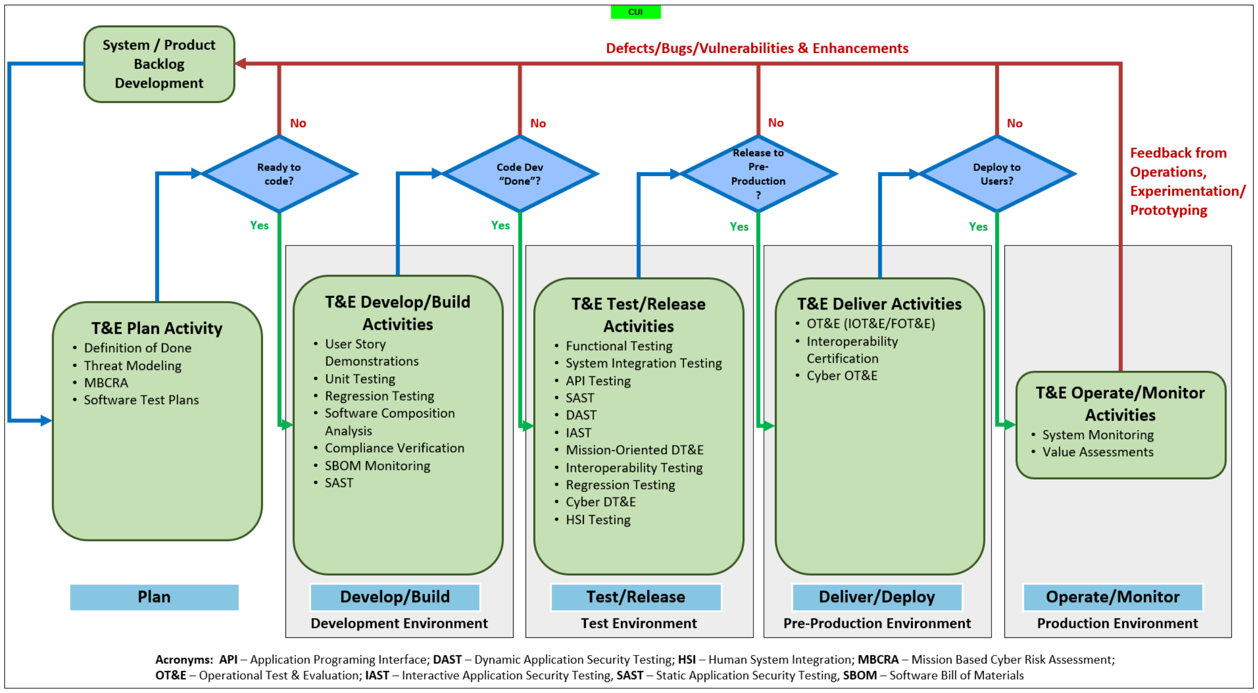

CUI
System / Product Backlog Development
Defects/Bugs/Vulnerabilities & Enhancements
No
No
No
No
Ready to code?
Code Dev "Done"?
Release to Pre-Production ?
Deploy to Users?
Yes
Yes
Yes
Yes
Feedback from Operations, Experimentation/ Prototyping
T&E Plan Activity
• Definition of Done
• Threat Modeling
• MBCRA
• Software Test Plans
T&E Develop/Build Activities
• User Story Demonstrations
• Unit Testing
• Regression Testing
• Software Composition Analysis
• Compliance Verification
• SBOM Monitoring
• SAST
T&E Test/Release Activities
• Functional Testing
• System Integration Testing
• API Testing
• SAST
• DAST
• IAST
• Mission-Oriented DT&E
• Interoperability Testing
• Regression Testing
• Cyber DT&E
• HSI Testing
T&E Deliver Activities
• OT&E (IOT&E/FOT&E)
• Interoperability Certification
• Cyber OT&E
T&E Operate/Monitor Activities
• System Monitoring
• Value Assessments
Plan
Develop/Build
Development Environment
Test/Release
Test Environment
Deliver/Deploy
Pre-Production Environment
Operate/Monitor
Production Environment
Acronyms: API – Application Programing Interface; DAST – Dynamic Application Security Testing; HSI – Human System Integration; MBCRA – Mission Based Cyber Risk Assessment; OT&E – Operational Test & Evaluation; IAST – Interactive Application Security Testing, SAST – Static Application Security Testing, SBOM – Software Bill of Materials

Figure 1-2. Notional T&E Life Cycle

1.3.4 DevSecOps Document Set

The DoD CIO published the DoD Enterprise DevSecOps Document Set for use by DoD product teams who use DevSecOps to develop, secure, deliver, deploy, and operate DoD mission applications. Although the DevSecOps Document Set is not policy, it does provide a logical description of the DevSecOps key design components and processes. The DevSecOps Document Set is available on the DoD CIO Library Website in the Modern Software Practices section. Test teams are encouraged to read this set of documents to better understand DevSecOps within DoD.

1.4 Enterprise Test and Evaluation Policy and Guidance

The following policy and guidance documents complement this guidebook and should be consulted as part of the overall DT&E approach to DevSecOps development:

- DoDI 5000.87, "Operation of the Software Acquisition Pathway."
- DoDI 5000.89, "Test and Evaluation."
- DoDI 5000.90, "Cybersecurity for Acquisition Decision Authorities and Program Managers."
- DoDI 5000.75, "Business Systems Requirements and Acquisition."
- DoD T&E Enterprise Guidebook, Chapter 5: Software Acquisition.
- Joint Interoperability Test Command (JITC) Guidebook for DevSecOps T&E.
- DoDI 5000.DT, "Developmental Test and Evaluation" (currently under development by the Office of the Under Secretary of Defense for Research and Engineering (OUSD(R&E))/Developmental Test, Evaluation, and Assessments (DTE&A) to be approved by the USD(R&E) and published on the DoD Issuances Website).
- DoD Manual 5000.UY, "Cyber Developmental Test and Evaluation" (currently under review workflow to be approved by the USD(R&E) and published on the DoD Issuances Website).
- DoD Cyber DT&E Guidebook, Version 3.0 (currently under review workflow to be published by OUSD(R&E)/DTE&A).

1.4.1 developmental Test and Evaluation as a Continuum

The dTEaaC framework leverages the principles of Agile and scales them to move T&E holistically from a serial set of activities to an integrative framework focused on a continuum of capability and outcome-focused testing, Agile scalable evaluation, and enhanced test design facilitating an ongoing campaign of learning.

This guidebook is part of a broader set of T&E guidance developed to support the implementation of a paradigm shift into dTEaaC. Key enablers include model-based systems engineering (MBSE) and digital engineering (DE) (authoritative source of truth); incorporation of technological innovations; a supportive infrastructure; and a transformed culture.

Each key enabler plays a critical role in conducting dTEaaC. Combined, the key enablers will provide timely and comprehensive information on system performance and risk characterization from the earliest design stages, enabling rapid development and fielding along with ongoing support for these increasingly complex systems and SoS. T&E, as an integral part of SE and ME processes (facilitated by using DE), not only supports decision making on individual systems but also helps enable the informed management of DoD capability development portfolios.

The three key tenets and multiple key attributes and enablers for implementing dTEaaC are as follows:

- **Key Tenet 1**: Links a Campaign of Learning to Warfighter Needs across the Capability Life Cycle
    - Establishes a Campaign of Learning based on data, information, and knowledge across the capability life cycle.
    - Promotes continuous mission validation via an Agile, iterative, data-driven, and knowledge-based framework for capability delivery, moving away from an event-based, serial approach to a data-driven approach.
    - Facilitates critical testing and validation of emerging technologies, including artificial intelligence (AI)-enabled and software-intensive systems.
    - Optimizes S&T, P&E, and test execution in support of pre-acquisition technology maturation, readiness, transition, and risk reduction.
    - Emphasizes the assessment of warfighting capabilities across the testing continuum.
    - Supports continuous validation of fielded Agile development and learning systems.
- **Key Tenet 2**: Provides Timely Decision Support
    - Promotes a decision framework that integrates insights from T&E early in the life cycle in support of complex decision making, ensuring an enhanced level of validation of warfighting needs through the alignment of data, analysis, and evaluation to support capability delivery decisions. This decision framework leverages the Integrated Decision Support Key (IDSK) framework applied both pre-acquisition and post-acquisition, integrating into an enterprise-level integrated data framework.
    - Enables earlier discovery and mission-focused decision making.

    - Provides ongoing and continuing learning and feedback to build better understanding for decision makers.
    - Enables near real-time decision dashboards and visualizations directly supported by the specific relevant authoritative data.
- **Key Tenet 3**: Aligns T&E with the Digital Innovation Ecosystem
    - Fully integrates and aligns T&E with model-based DE development and application, ensuring an integrative environment between ME, SE, and T&E models and processes across the capability life cycle.
    - Closely integrates with ongoing digital workforce initiatives and curriculum to develop the requisite T&E engineering skills and experiences for the workforce.
    - Builds the foundation for a modeling and simulation (M&S) "digital thread" continuum documenting the core testing required to validate and accredit models.
    - Leverages the modeling environment to complement and improve live testing.
    - Establishes the foundational integrated data and information architecture that is essential for seamless knowledge sharing and enhanced decision making.

1.4.2 Scientific Test and Analysis Techniques

Scientific test and analysis techniques (STAT) are deliberate, methodical approaches to develop effective, efficient, and rigorous test strategies yielding defensible results. Incorporating STAT enables quality deployed capability(ies). If a capability's functionality is not rigorously tested in development, then that capability when deployed may be rolled back with defects found, ultimately reducing mission effectiveness and delaying that capability to the warfighter. Programs developing software using the DevSecOps life cycle are encouraged to incorporate STAT throughout the DevSecOps phases.

The STAT Center of Excellence (COE), sponsored by OUSD(R&E)/DTE&A, has prepared several reference documents to address software DT&E in DevSecOps, including the following:

- Test Planning Guide. This guide is designed for test planners and engineers who would like to inject greater rigor into their testing and achieve a deeper, more quantifiable understanding of a system under test (SUT). It aims to walk the reader through the STAT Process and offers a high-level roadmap of the critical activities, techniques, and concerns for applying STAT planning, testing, and analysis.
- Automated Software Testing Implementation Guide. This guide is intended to serve those in DoD interested in applying automation to software testing. It applies an SE process based on the scientific method for the steps to conduct and to achieve an automation

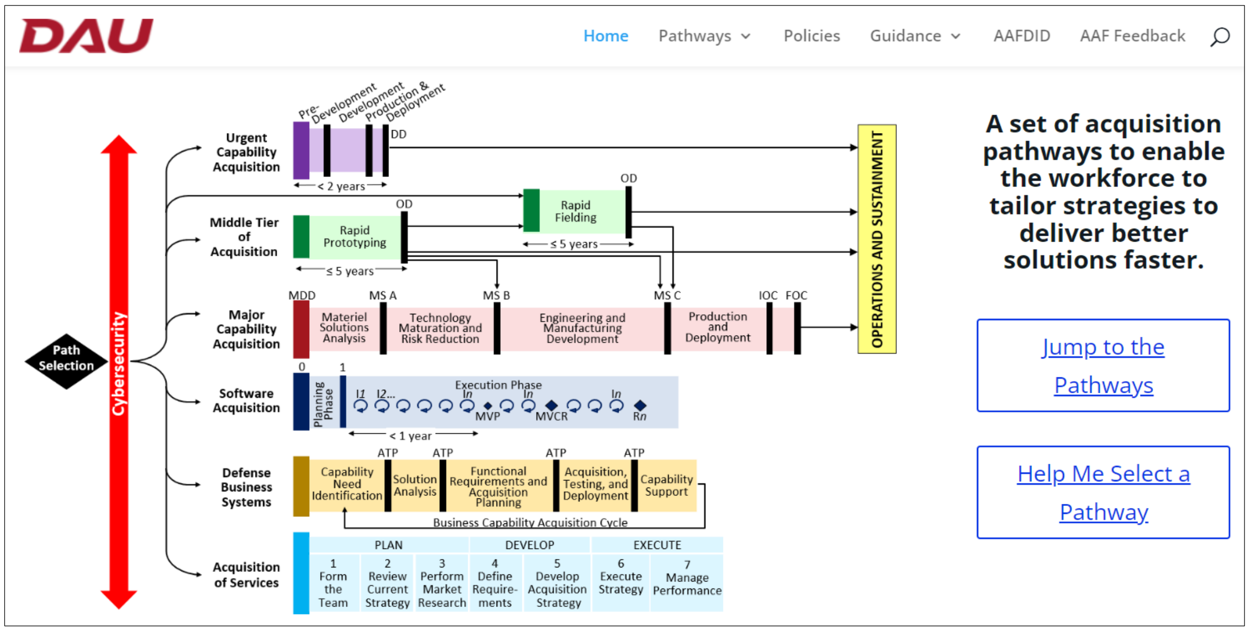
capability along with the important need to perform a return on investment (ROI) analysis to make the business case for automation.

1.5 Other Available Guidance

1.5.1 Adaptive Acquisition Framework

Because software is at the core of modern systems, this guidebook is agnostic with respect to acquisition pathways. Perhaps inevitably, it appears to emphasize the software acquisition pathway, which is the one tied most closely to DevSecOps. The material in this guidebook aims to describe how T&E differs when DevSecOps is used. The guidebook's observations on DevSecOps-based T&E are relevant to any acquisition pathway that includes T&E. To include the specialized vocabularies used in each acquisition pathway would unnecessarily complicate and confuse the discussion.

The USD(A&S) established an AAF Website, hosted by the Defense Acquisition University (DAU), to provide supplementary guidance to the DoDIs relative to the AAF. Figure 1-3 captures the featured graphic on this website. The "Software Acquisition" link leads to a page that provides guidance on Agile/DevSecOps, including the life cycle view of software acquisition illustrated in Figure 1-4.



A set of acquisition pathways to enable the workforce to tailor strategies to deliver better solutions faster.

Source: DAU Adaptive Acquisition Framework Website

Figure 1-3. USD(A&S) Adaptive Acquisition Framework

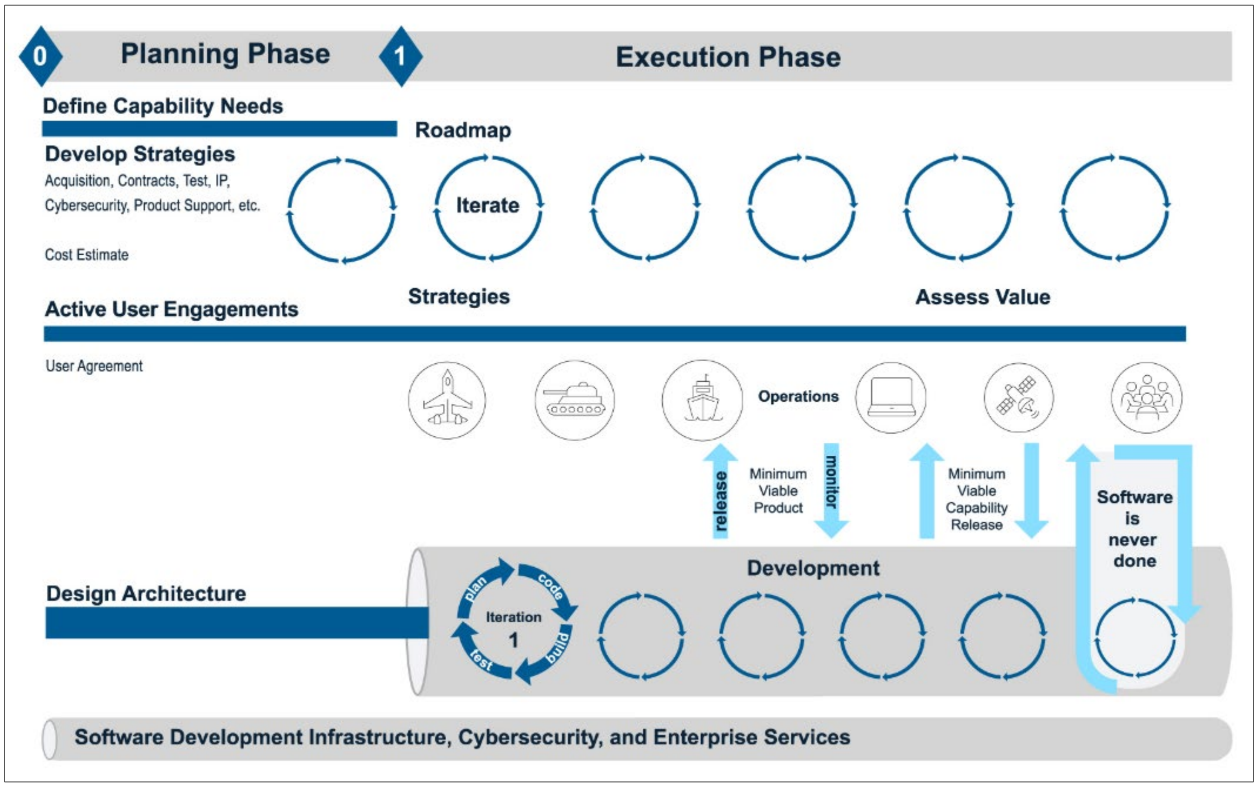


Figure 1-4. Life Cycle View of Software Acquisition

1.5.2 Risk Management Framework

DoDI 8510.01, “Risk Management Framework for DoD Systems,” establishes the cybersecurity Risk Management Framework (RMF) for DoD systems. The issuance establishes policy, assigns responsibilities, and prescribes procedures for executing and maintaining the RMF. The RMF is designed to integrate cybersecurity into the system development life cycle, ensuring that risks are managed effectively throughout the acquisition and operational phases of DoD systems. It also provides guidance on reciprocity of system authorization decisions in coordination with other Federal agencies.

2 DT&E in DevSecOps – Roles, Teams, Accountability, and Authority

Adopting Agile and DevSecOps practices for a defense acquisition program requires an adjustment to the traditional roles and responsibilities that program offices are familiar with under traditional acquisition practices. The roles in Figure 2-1 are directly affected by being in a DevSecOps environment. These roles, which are drawn from DoDI 5000.02, DoDI 5000.75, and the DAU Glossary of Defense Acquisition Acronyms and Terms, are described further in the following sections.

The Agile framework favors decision making that is based not on the command and control of management but on a collaborative and consensus-based process. This collaborative process is highlighted by two information flows, which are depicted using yellow and green arrows in Figure 2-1. The yellow arrows capture the requirements and user feedback information flow. The green arrows capture the development information flow (specifications, subject matter expertise, product owner user input, software development priorities, user demonstrations, and MVP/minimum viable capability release (MVCR) deployments).

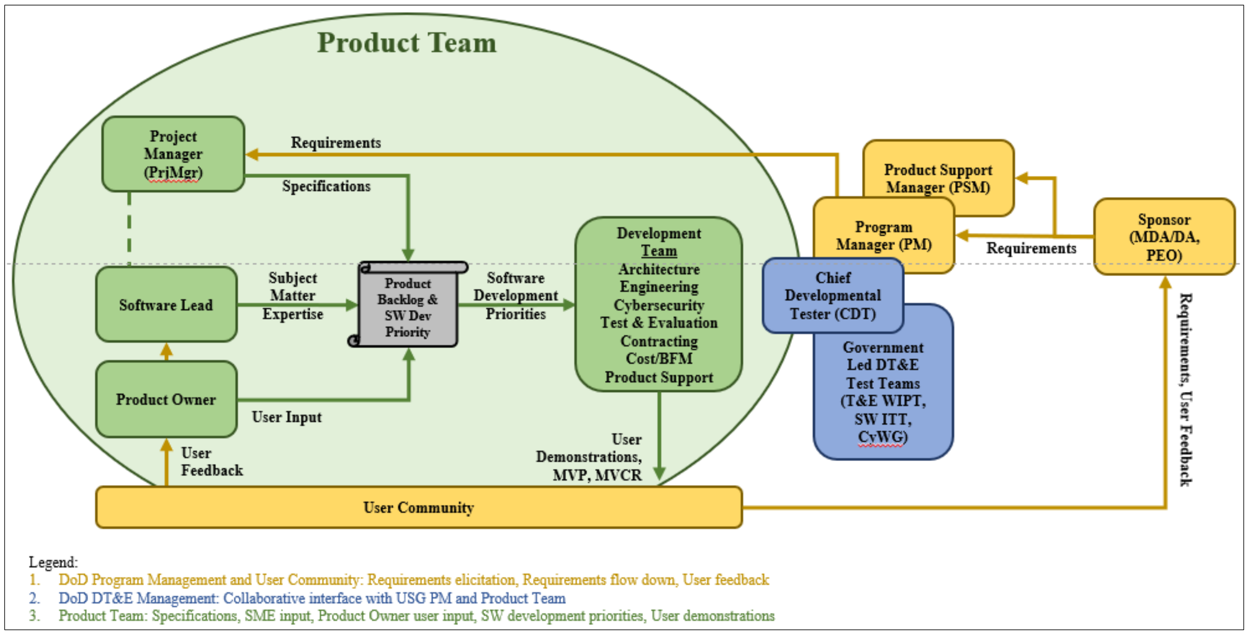

Product Team
Project Manager (PrjMgr)
Requirements
Specifications
Software Lead
Subject Matter Expertise
Product Backlog & SW Dev Priority
Software Development Priorities
Development Team
Architecture
Engineering
Cybersecurity
Test & Evaluation
Contracting
Cost/BFM
Product Support
Product Owner
User Input
User Feedback
User Demonstrations, MVP, MVCR
User Community
Product Support Manager (PSM)
Program Manager (PM)
Requirements
Sponsor (MDA/DA, PEO)
Chief Developmental Tester (CDT)
Government Led DT&E Test Teams (T&E WIPT, SW ITT, CyWG)
Requirements, User Feedback
Legend:
1. DoD Program Management and User Community: Requirements elicitation, Requirements flow down, User feedback
2. DoD DT&E Management: Collaborative interface with USG PM and Product Team
3. Product Team: Specifications, SME input, Product Owner user input, SW development priorities, User demonstrations

Figure 2-1. DT&E U.S. Government Program Manager, Product Team, User Community Relationship

2.1 DoD Program Management

DoD program management will have full and prompt access to all ongoing integrated DT activity and artifacts, including all DT records and reports, which include data from all tests, system logs, execution logs, test director notes, certifications, and user/operator assessments and surveys.

Regardless of the acquisition methodology or pathway being employed, all DoD acquisition programs need to identify those managers at varying levels who can make essential decisions regarding the initiation, goals, progress, and conclusion of that program. Section 3 of DoDI 5000.02 identifies four specific program management roles and authorities:

- Milestone Decision Authority (MDA)/Decision Authority (DA)
- Program Executive Officer (PEO)
- Program Manager (PM)
- Product Support Manager (PSM)

2.1.1 Milestone Decision Authority/Decision Authority

The MDA/DA is the program decision authority and specifies the decision points and procedures for assigned programs. MDAs/DAs tailor program strategies and oversight, phase content, the timing and scope of decision reviews, and decision levels based on the characteristics of the capability being acquired (including complexity, risk, and urgency) *to satisfy user requirements*. MDAs for Major Defense Acquisition Programs and major systems will approve, as appropriate, the acquisition strategy at all major decision points. The MDA/DA may choose to delegate some or all these decisions to lower levels of the acquisition process (e.g., PEO or Component Acquisition Executive (CAE)) in the interest of speed or efficiency.

In traditional system acquisition practices, decision points derive from plans to deliver monolithic capabilities. In DevSecOps, plans derive from (1) initial system delivery focusing on an MVP; (2) initial warfighting capabilities to enhance mission outcomes, known as the MVCR; and (3) subsequent warfighting capabilities to enhance mission outcomes, known as a CR.

- The MVP is an early version of the software to deliver or field basic capabilities to users for evaluation and to provide feedback. Insights from the MVP help shape scope, requirements, and design. The MVP is a milestone likely to occur early in the system acquisition life cycle.
- The MVCR is 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.
- A CR is a subsequent set of features suitable to be fielded to an operational environment that provides value to the warfighter or end user in a rapid timeline.

This rhythm alters the milestones from a traditional system acquisition. Traditional system acquisition intuitively takes longer to yield capability(ies) than delivering an MVCR. Adapting to this rhythm under DevSecOps will support a quicker acquisition cadence and more frequent and effective engagement with the program team throughout the acquisition life cycle.

2.1.2 Program Executive Officer

The PEO balances the risk, cost, schedule, performance, interoperability, sustainability, and affordability of an acquisition program's portfolio and delivers an integrated suite of mission-effective capability to users.

In DevSecOps, the incremental deliveries affect how capabilities are delivered to users. The PEO's balancing efforts must *account for and prioritize user needs*. DT&E continuously provides test data and evaluations to the PEO to make informed decisions.

2.1.3 Program Manager

The role of the PM, under the supervision of PEOs and CAEs, is to (1) plan acquisition programs, prepare programs for key decisions, and execute approved acquisition and product support strategies; and (2) employ a thoughtful, innovative, and disciplined approach to program management.

Within the DAS, the PM directs the development, production, and deployment of new or modified defense systems, whether a business or weapon system. The PM also defines and employs management activities to achieve the cost, schedule, and performance parameters specified.

2.1.4 Product Support Manager

The PSM develops, plans, and implements a comprehensive product support strategy for all Integrated Product Support (IPS) elements and their materiel readiness. The IPS elements are a set of 12 interrelated and structured elements defined in DoDI 5000.91, "Product Support Management for the Adaptative Acquisition Framework." PSMs make use of data-driven decision-making tools with appropriate predictive analysis capabilities to improve systems availability and reduce costs.

The role of the PSM is particularly important role in DevSecOps because of the potential rapid change in product support needs even after initial fielding. The PSM must work closely with the PM to coordinate support planning with capability delivery.

2.1.5 DoD Program Management Accountability

DoDI 5000.87 makes many references to "the sponsor" and "the user community" without providing definitive guidance. As captured in Figure 2-1, not all programs will necessarily use these terms. The critical point is that *key user community representatives work in parallel with the sponsor and PM to ensure user needs are met*. Test teams should consider these individuals for requirements clarification, mission-thread definitions, use case definitions, and user personnel to support T&E.

Sponsor

The sponsor is the MDA/DA, CAE, or PEO senior leader with mission function responsibility seeking to improve mission performance. The sponsor confirms the need for improved mission operations and *represents the user community interests throughout development and fielding*.

The sponsor, in collaboration with the PM and/or PSM:

- Leads solution analyses by:
  - Engaging stakeholders to keep them actively involved in shaping the complete future solution.
  - Making resources available for each phase of requirements and acquisition, including stakeholders and subject matter experts.
  - Programming and budgeting for life cycle costs of full mission spectrum solutions.
  - Defining requirements, including market research for developing the mission's system.
  - Validating that deployed capabilities meet mission requirements, deliver expected benefits, and provide ROI.
- Develops a high-level initial Capability Needs Statement (CNS) to guide the program planning phase if the program is on the software acquisition pathway (see guidance on the DAU AAF Website); otherwise, develops a Capability Development Document (CDD) for Joint Requirements Oversight Council validated joint requirements (discussed in Section 4.1.1).
- Approves the CNS to guide the program execution phase.
- Approves the user agreement (UA) committing users to support T&E.
- Leads and approves the periodic value assessment of the software solution.

- Develops and proposes an acquisition strategy including the timing and scope of decision reviews, metrics, and required documentation.

Program Manager

The PM, in collaboration with the PSM:

- Oversees preparation of the TEMP or TES with the T&E Working-level Integrated Product Team (WIPT) (discussed in Section 2.2.2).
- Collaborates with the sponsor on a succinct CNS that reflects the MVP along with more substantial but incremental delivery targets (MVCR).
- Develops a UA that identifies the *desired level of user involvement and expectations* for collaborative methods to evolve capability delivery timelines. The UA includes the commitment of user resources to support the execution phase. The PM should indicate in the acquisition strategy or UA which decisions are allocated to any level of decision; test teams need to know where the decisions are made relative to the activity that testing is supporting.
- Collaborates with the chief developmental tester (CDT) to ensure that, under contract, automated test tools are identified, selected, and integrated before starting development.
- Has content authority for the program backlog and is responsible for prioritizing features and developing the program vision and roadmap.

2.2 DoD Test and Evaluation Management

Involving all stakeholder communities (e.g., DT&E, OT&E, safety, cyber, and airworthiness) in integrated test planning and providing access to test data and results will save schedule duration and alleviate challenges to access that data late in the development cycle.

2.2.1 Chief Developmental Tester

The CDT (or DT&E lead) manages an acquisition program's T&E activities during all phases of the acquisition process; designates the lead cyber DT&E organization; and coordinates the planning, management, and oversight of all cyber DT&E activities performed by the product team and the lead cyber DT&E organization.

The CDT:

- Coordinates the planning, management, and oversight of all DT&E activities for the product team.
- Maintains insight into development team activities under the product team and oversees the T&E activities of other participating government test team activities under the program.
- Helps PMs make technically informed, objective judgments about product team DT&E results under the program.
- Ensures that the test activity provides the PM and sponsor with the data needed to make acquisition decisions.

In a DevSecOps construct, test data comes from the product team's test activities and from user demonstrations, integration tests, and the results of test events conducted by government test teams. Therefore, the CDT needs visibility into the developers' automated management and testing tools to extract data when necessary to support independent assessment of test results or acquisition/program management decisions. The CDT is responsible for ensuring that the test resources, infrastructure, and personnel are available to complete the test events identified in the test strategy. To that end, the CDT serves as the liaison to independent government test organizations supporting the program. More information on CDT responsibilities will be provided in the forthcoming DoDI 5000.DT.

Where the product team (discussed in Section 2.3) also has system integrator responsibility, functional capability above the level of the user story must be demonstrated, and the CDT should expect access to test data from the product team to verify that the functionality has been delivered. Identifying individuals within the product team's organization who are responsible for the system integration testing is a key task for the CDT.

2.2.2 Test and Evaluation Working-level Integrated Product Team

As outlined in DoDI 5000.89, the T&E WIPT, chaired by the CDT:

- Provides a forum for involvement by all key organizations in the T&E effort.
- Develops the TEMP or TES documentation for the PM. Requires all key stakeholders to be afforded an opportunity to contribute to the TEMP or TES documentation development.
- Includes representatives of test data stakeholders such as SE, DT&E, OT&E, live-fire test and evaluation (LFT&E), the user, product support, the intelligence community, and applicable certification authorities.

- Supports the development and tracking of an integrated test program for DT&E, OT&E, LFT&E, and M&S to support evaluations.
- Supports the development and maintenance of the integrated test schedule.
- Identifies and provides recommended corrective action or risk assessment as necessary.
- Explores and facilitates opportunities to conduct integrated testing to meet DT&E, OT&E, and LFT&E objectives.

The T&E WIPT will identify DT&E, OT&E, and LFT&E data requirements needed to inform critical acquisition and engineering decisions. The T&E WIPT requires test objectives to be understood; the testing to be conducted in an operational context to the maximum extent possible; and the resultant data to be relevant for use in independent evaluations and the rationale behind the requirements.

In DevSecOps, the T&E WIPT is not directly involved in conducting tests but has an important oversight role. Section 5 describes the activities in which the T&E WIPT participates.

2.2.3 Software Integrated Test Team

Once the T&E WIPT identifies the data requirements, the Software Integrated Test Team (SW ITT) (made up of designated members of the government led test teams and development team) will determine which requirements can be satisfied through testing and will develop an integrated test matrix. The SW ITT also identifies responsibility of who leads and executes the tests identified in the integrated test matrix. The SW ITT will use the test matrix to document test results as described in the user stories or other artifacts.

As outlined in DoDI 5000.89, DT&E activities will start when requirements are being developed to ensure that key technical requirements are measurable, testable, and achievable, as well as provide feedback that the system and test engineering processes are performing adequately. The SW ITT is fundamental to this activity.

Although the SW ITT members may not be specifically referred to as testers, they perform other duties such as those of automation architects, automation test developers, or code developers.

2.2.4 Cyber Working Group

The Cyber Working Group (CyWG) is responsible for the planning, execution, and reporting of cyber T&E, as well as providing support to the program's RMF assessment and authorization process. The PM or S&T manager charters the CyWG to coordinate internal and external organizations.

In accordance with the forthcoming DoDM 5000.UY, cyber DT&E events (including contractor and government cyber DT&E) should be planned, as applicable, across the system life cycle to assess CyWG prioritized subcomponents, components, subsystems, systems, and SoS iteratively and recursively. With respect to a mission context, cyber DT&E events are informed by mission-based cyber risk assessments (MBCRAs) to identify exploitable cyberspace vulnerabilities and other system faults in the SUT. CyWG roles, accountability, and authority include:

- Support to the T&E WIPT and SW ITT. The CyWG will coordinate, deconflict, and schedule support for the program's cyber test activities. Whether a large or small acquisition program, continuous collaboration among the program management team will lead to successful cyber DT&E and the opportunity to design and develop a more secure, cyber-survivable, and operationally resilient system.
- Recommended Cybersecurity Assessments. The CyWG recommends cybersecurity assessments at an appropriate frequency to conduct cybersecurity and third-party assessments throughout the system development life cycle. The program office is responsible for coordinating, evaluating, and updating the MBCRA and RMF assessments.
- SaC and Manual Cyber DT&E Events. DevSecOps SaC automated testing in the pipeline informs the manual cyber DT&E that the product development team performs before the government-performed cyber DT&E. Cyber DT&E is essential when integrating software functional updates with existing software and when integrating software into the overall system. The CyWG plans all activities associated with SaC and manual cyber DT&E events.

More information on CyWG participants and roles will be provided in the forthcoming Version 3.0 of the DoD Cyber DT&E Guidebook. The CyWG should use Appendixes D and E in the forthcoming guidebook to inform their tasks.

2.3 Product Team

2.3.1 Project Manager

A key difference between traditional and Agile program management is the inclusion of the development team. Within the Agile framework, with the sponsor's input on the capabilities and features desired in priority order, the project manager (PrjMgr) defines and manages the capability definition and tasks for the product backlog.

2.3.2 Product Owner

The product owner works closely with the user community to *ensure that the requirements continue to reflect the user community's needs and priorities and align to the mission objectives*. The product owner is responsible for defining stories, defining acceptance criteria, understanding feature priorities, and prioritizing the product backlog. These activities streamline the execution of program priorities while maintaining the conceptual and technical integrity of the features or components for the team.

The definition of the user story must also include clear acceptance criteria, which will be the basis for testing the developed software. The product owner should apply the INVEST (i.e., independent, negotiable, valuable, estimable, small, and testable) criteria when evaluating user stories for inclusion in the product backlog. (See the General Services Administration (GSA) DevSecOps Website for guidance on writing effective user stories.) The product owner works with the software lead and development team to develop test cases at the user story level, defined to meet the user's needs.

For most enterprises moving to DevSecOps, the product owner is a new and critical role. Note that the PrjMgr and product owner could be one and the same.

2.3.3 Software Lead

The software lead is a key position in the DevSecOps framework that does not translate to traditional DAS methods or terminology. The software lead representative must understand the system being developed or improved and *be able to speak to the solution as it reflects the user requirements*. Users may vary from the technician on a keyboard in a business center to an operations center watch or tank commander using software on the battlefield.

The software lead:

- Reports to the PrjMgr
- Leads the definition of functional requirements, training, and deployment for the mission capability(ies).
- Depending on the mission solution, leads the business process reengineering and execution of business process changes.
- Works with the PrjMgr to identify test resources (human) and integrate them with the development team. This activity is related to the UA (i.e., documenting a commitment from the sponsor to provide adequate test resources).

- Elaborates on the CNS developed by the sponsor in sufficient detail to guide the development (execution phase).
- Can serve as an interface between the PrjMgr, product owner, and/or user community.
- Provides the PrjMgr and product owner with program backlog executability input.
- Works with the developers and test teams to create test cases for the develop, build, test, and release/deliver phases in support of the DevSecOps software life cycle (discussed in Section 1.3).
- May perform user acceptance responsibilities at the test and release/deliver phases in support of the DevSecOps software life cycle (reflected in the Figure 1-1).
- May lead the user team assigned to support the software development and testing. The user team may be quite large, representing all the military occupational skills or civilian knowledge, skills, and abilities of users in the organization who will employ the system.
- May provide end of sprint or iteration demonstrations of product delivery to all stakeholders.

2.3.4 Development Team

Roles, accountability, and authority within a software development team are often distributed based on expertise and responsibilities. Although each member may have a particular area of specialization, effective collaboration and communication are key to the team's success. Ultimately, the team works together to deliver high-quality software that meets user needs and mission objectives.

In DevSecOps, the development team plays a large role in developing and conducting tests. Government test teams (discussed in Section 2.4) oversee development team activities and review the development team's work products. Government test teams also develop and conduct tests that the development team cannot effectively perform.

The titles, roles, accountability, and authority within a software development team typically vary depending on the specific methodologies and structures followed. The following list provides a general overview of the key roles within a DevSecOps environment:

- Architect: Responsible for defining the overall architecture of the software, making high-level design decisions, and guiding the technical direction of the project. Architects often mentor other team members and ensure that the codebase adheres to best practices and architectural principles.
- Engineering: Engineering roles encompass the following:

- Software Developer/Engineer: Responsible for writing code, designing software, debugging, and testing applications. Software engineers often work closely with other team members to ensure that the software meets requirements and functions correctly.
- Quality Assurance Engineer: Responsible for testing software to identify bugs, defects, and areas for improvement. QA engineers work closely with developers to ensure that software meets quality standards before release.
- User Interface (UI)/User Experience (UX) Designer: Accountable for designing the user experience and interface of the software. UI/UX designers work with stakeholders and developers to create intuitive and visually appealing interfaces that enhance user satisfaction.
- DevSecOps Engineer: Responsible for automating and streamlining the software development process, from code deployment to infrastructure management. DevSecOps engineers ensure that development, testing, and deployment pipelines are efficient and reliable.
- Human Factors Engineer: Responsible for the planning and implementation of a human systems integration (HSI) program from initial user requirements through the program life cycle. Human factors engineers perform, document, and manage program and system human-centered design considerations and readiness risks through trade-off analyses among the HSI domains.

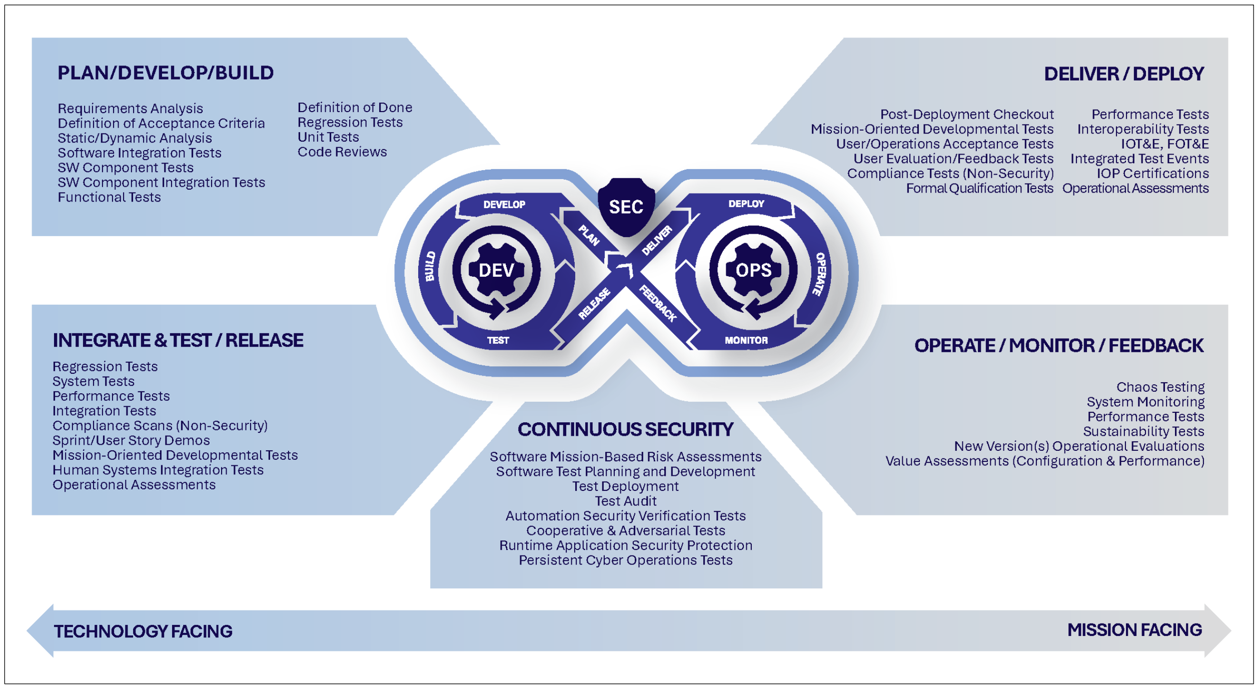
- Security Champion: As defined in the Open Worldwide Application Security Project (OWASP) Developer Guide, Security Champions Website, a security champion is a member of a software development team who is the liaison between Information Security and developers. This helps to embed security into the development organization. Security champion roles, accountability, and authority include the following:
  - Serves as a member of the product team.
  - Serves as a member of the CyWG.
  - Helps improve the communication between development teams, the cybersecurity team, and the CyWG.
  - Will usually be involved in risk/threat assessments and architectural reviews. Can often help identify opportunities to remediate security defects, making the architecture of the application more resilient and reducing the attack threat surface.
  - Participates in periodic briefings to increase awareness and expertise in different security disciplines.
  - Promotes and tracks secure coding practices and ensures that security is a constant consideration in processes and tools.

2.4 Government Test Teams

To reduce risk in the transition from the development to the production environments, government test teams should plan to use test or pre-production environments to test at the release level. The test and pre-production environments need to be increasingly representative of the production environment. A collaborative interface with the development team is crucial for project success. Additionally, government test teams may have to learn new skills in the use of automated test management and execution tools to succeed in DevSecOps (training is discussed in Section 6).

2.4.1 Government Analysis and Assessment Tasks

Testing activities for integration with the DevSecOps life cycle, captured in the DoD Enterprise DevSecOps Fundamentals v2.5 guidance, are depicted in Figure 2-2. The figure provides an illustrative summary of the government analysis and assessment tasks by DT (technology facing); operational test (OT) (mission facing); and continuous security. These testing activities were captured in coordination with the DoD CIO; Director, Operational Test and Evaluation (DOT&E); and JITC. The DT and continuous security activity considerations in Figure 2-2 are the subject matter of this guidebook.


PLAN/DEVELOP/BUILD
Requirements Analysis
Definition of Acceptance Criteria
Static/Dynamic Analysis
Software Integration Tests
SW Component Tests
SW Component Integration Tests
Functional Tests
Definition of Done
Regression Tests
Unit Tests
Code Reviews
DELIVER / DEPLOY
Post-Deployment Checkout
Mission-Oriented Developmental Tests
User/Operations Acceptance Tests
User Evaluation/Feedback Tests
Compliance Tests (Non-Security)
Formal Qualification Tests
Performance Tests
Interoperability Tests
IOT&E, FOT&E
Integrated Test Events
IOP Certifications
Operational Assessments
SEC
DEVELOP
BUILD
DEV
TEST
PLAN
RELEASE
DELIVER
FEEDBACK
DEPLOY
OPS
OPERATE
MONITOR
INTEGRATE & TEST / RELEASE
Regression Tests
System Tests
Performance Tests
Integration Tests
Compliance Scans (Non-Security)
Sprint/User Story Demos
Mission-Oriented Developmental Tests
Human Systems Integration Tests
Operational Assessments
OPERATE / MONITOR / FEEDBACK
Chaos Testing
System Monitoring
Performance Tests
Sustainability Tests
New Version(s) Operational Evaluations
Value Assessments (Configuration & Performance)
CONTINUOUS SECURITY
Software Mission-Based Risk Assessments
Software Test Planning and Development
Test Deployment
Test Audit
Automation Security Verification Tests
Cooperative & Adversarial Tests
Runtime Application Security Protection
Persistent Cyber Operations Tests
TECHNOLOGY FACING
MISSION FACING

Source: DoD Enterprise DevSecOps Fundamentals, Version 2.5, p. 23

Figure 2-2. Testing Activities for Integration with the DevSecOps Life Cycle

2.4.2 Developmental Test and Evaluation Teams

DT&E teams in the DoD DevSecOps environment fall into three primary categories:

- Program Level. The independent government test teams should strive to be active stakeholders in the development team to maximize communications and maintain continuous situational awareness of development team and product team testing.
- Service Level. Service-level/integration testing roles are not generally included in commercial DevSecOps process descriptions; however, verifying the successful integration of software developed and tested through DevSecOps methods into larger systems, or SoS, is still useful and necessary in the DoD environment. The schedule impact of additional levels of DT&E on the flow of software production can be minimized by enabling access to test data generated by the product team or the program office. With early coordination across test organizations, the program office can ensure that the test management system is accessible, and the automated test software will provide the necessary output to satisfy independent government test teams and oversight assessment requirements.
- Oversight. Oversight from Service-level test organizations and the Office of the Secretary of Defense (OSD) is still viable in DevSecOps methodologies to inform senior leadership of program progress and capture lessons learned to help future adopters.

2.4.3 Lead Developmental Test Organization

The lead developmental test organization (LDTO), when established, functions as the lead integrator for a program's DT&E activities. It is separate from the program office but supports the PM and CDT in a provider-customer relationship regarding scope, type, and conduct of required DT&E. More information on LDTO responsibilities will be provided in the forthcoming DoDI 5000.DT.

2.4.4 Operational Test and Evaluation Teams

In accordance with DoDI 5000.98, "Operational Test and Evaluation and Live Fire Test and Evaluation," DOT&E is responsible for independent evaluation of OT&E results for all DoD programs on DOT&E oversight. DOT&E approves all OT plans before the tests begin and provides objective reporting to the Secretary of Defense, Senate/House Armed Services Committees, and various DoD Components. There is ongoing DOT&E development of comparable software OT&E guidance to be captured as a companion guide to the DoDM 5000.96, "Operational and Live Fire Test and Evaluation of Software".

OT&E and LFT&E agencies should:

- Integrate with the development team.
- Integrate with DT&E planning, execution, analysis, and resourcing as DoD systems mature across the acquisition life cycle to introduce operationally relevant conditions early.
- Optimize the use of early DT&E test events in support of meeting independent OT&E and LFT&E objectives.

2.4.5 Interoperability Test and Evaluation Teams

For awareness, LDTOs and OT agencies may have organic interoperability T&E teams; however, some Military Services may have separate interoperability test organization labs. JITC must be involved with programs designated by the Joint Staff J-6 Command, Control, Communications, and Computers/Cyber Directorate for joint interoperability certification.

An interoperability T&E team is responsible for conducting interoperability tests and evaluating the results in accordance with the policies established in DoDI 8330.01, "Interoperability of Information Technology, Including National Security Systems." The interoperability T&E team is responsible for ensuring that developmental software systems work in a network of interfacing systems. The team also works with the PM to plan interoperability testing with these interfacing systems.

Procedures vary depending on whether joint or non-joint interfaces are to be tested. These procedures are identified in DoDIs 5000.87 and 5000.89. The interoperability T&E team provides input to the TEMP or TES documentation in accordance with DoDIs 5000.87 and 5000.89 to address how to evaluate (1) the timely, accurate, and complete transfer of data; and (2) the use of the data sent to or received from the interfaced system. The interoperability T&E team further works to ensure that the development environment in which the interoperability testing will occur, when distinct from the production environment, is functionally representative enough of the operational environment to fully allow the interfaces to be tested, verified, validated, and certified for joint interoperability. In the DevSecOps construct, interoperability T&E teams are also responsible for providing any additional OT&E required to provide data not output through the iterative automated or manual testing incorporated in the DevSecOps software pipeline. Sections 5.4.4, 5.5.2, and 5.6.2 address interoperability T&E team functions in more detail.

3 DT&E in DevSecOps – Foundational Concepts

In DevSecOps, DT&E takes on a specific role within the SDLC that integrates practices throughout the experimentation, development, testing, and operational phases. DT&E becomes an integral part of the continuous improvement cycle, including cyber T&E procedures that aim to ensure that security is not an afterthought but is ingrained into every stage of software development, deployment, and operations (see Paragraph 3.5. in DoDI 5000.89 for more information about cyber T&E procedures). This approach enables teams to build more effective and secure software; respond quickly to requirements changes and security threats; and maintain system capabilities and a robust security posture throughout the software's life cycle.

DT&E for DevSecOps, a combination of continuous and automated testing along with some manual testing is used iteratively and recurringly to verify that the technical functional requirements are met. Security tool integration, code analysis, threat modeling, compliance adherence, collaboration, and a continuous feedback loop are used to ensure that quality and security are an integral part of the development process.

Paragraph 1.2.k. of DoDI 5000.87 states, "Maximum sharing, reciprocity, availability, and reuse of results and artifacts between the various testing and certification organizations is encouraged." The tools, basic architecture, and instrumentation must be known by each of the respective government test parties to facilitate streamlined testing.

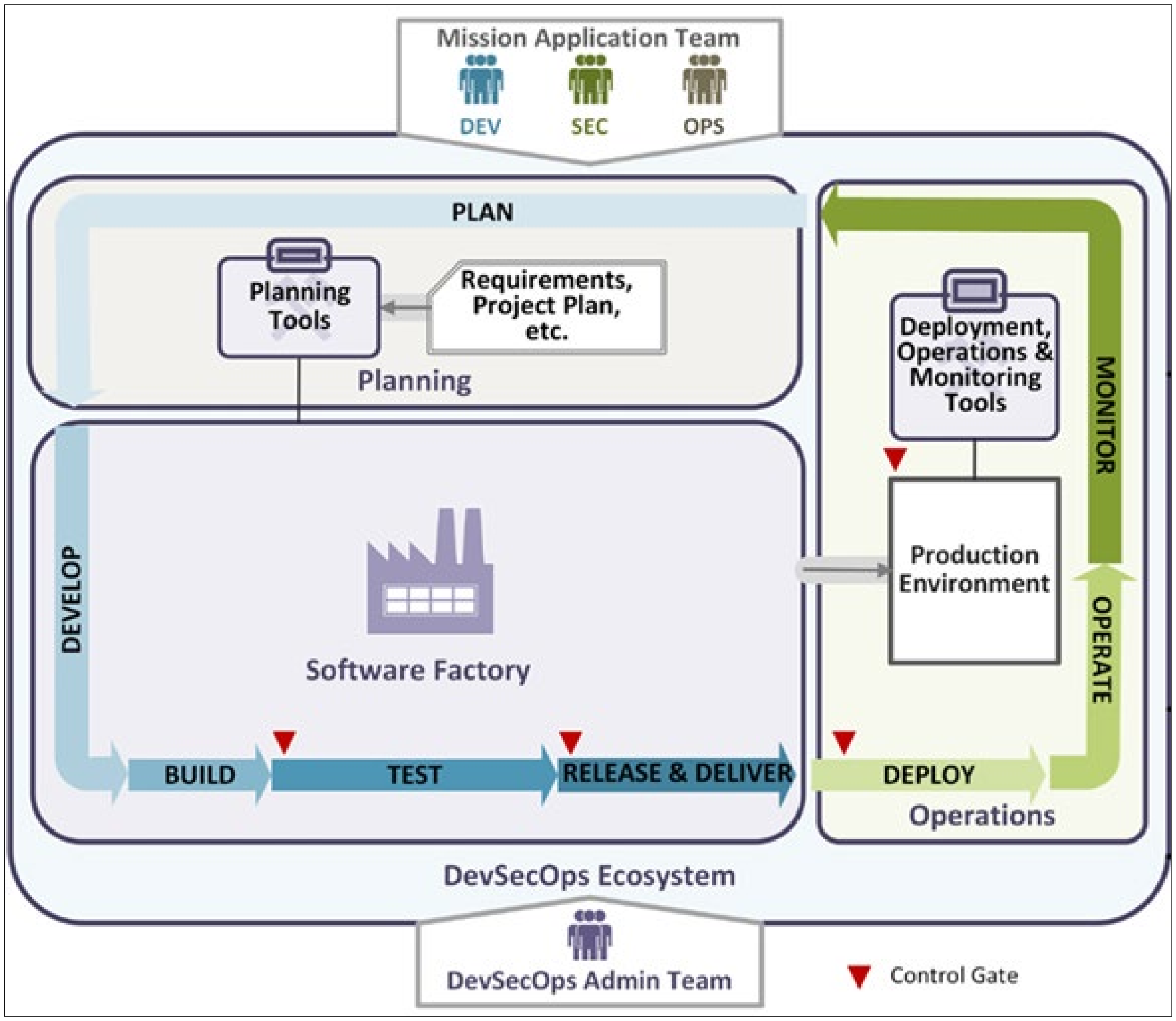
The foundational concepts associated with DT&E in DevSecOps addressed in this section include the following:

- DevSecOps Ecosystem. The DevSecOps ecosystem is a collection of software tools and process workflows that chain together to enable automation and repeatable testing throughout the life cycle. The ecosystem entails such items as software pipelines, control gates, ecosystem design, and automated tools.
- Metrics. Metrics give insights into what is happening at all stages of the pipeline, from design to deployment. Metrics are essential to the success of DevSecOps. Items such as data generation, software test metrics, and software security metrics should be considered as part of the metric foundational activities for DevSecOps.

3.1 DevSecOps Ecosystem

The DevSecOps ecosystem is a collection of software tools and process workflows that chain together to enable automation and repeatable testing throughout the life cycle. As illustrated in Figure 3-1, the DevSecOps ecosystem is composed of three subsystems: planning, the software factory, and operations. The DoD Enterprise DevSecOps Reference Design provides further details on the DevSecOps environment architecture; the processes for creating and managing these environments; and a glossary of key terms. The T&E community should become familiar with these concepts as components of the systems they will need to test.

Note: The control gate in Figure 3-1 from the “Deployment, Operations & Monitoring Tools” box to the “Production Environment” box is outside the scope of this guidebook.



Source: DoD Enterprise DevSecOps Reference Design, Version 1.0, p. 27

Figure 3-1. DevSecOps Ecosystem

Although implementing DevSecOps does not require a cloud environment, the cloud is well adapted to provide for the infrastructure, development, and maintenance of software development pipelines. It is the DoD expectation that enterprise-wide clouds (e.g., Joint Warfighting Cloud Capability, Air Force Cloud One, and milCloud) and DevSecOps platforms and software factories will reduce program life cycle costs. As a result, acquisition programs may expect to do their development in an enterprise software factory and operate in the cloud, using an infrastructure-as-a-service (IaaS), platform-as-a-service (PaaS), or software-as-a-service (SaaS) cloud delivery model.

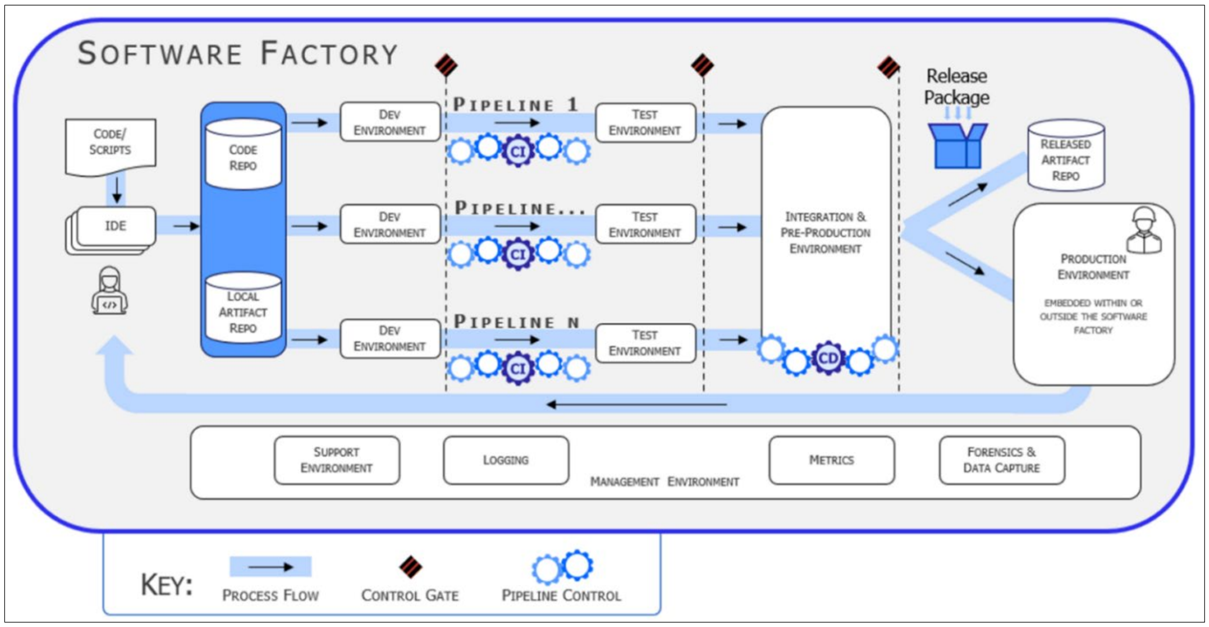
The government should verify that the shared security model for cyber T&E on the cloud environment is effective against the latest cyberspace threats. Additionally, the entire DevSecOps ecosystem should be verified and validated to be resilient against cyberspace threats using demonstrations, examinations, analyses, and testing to understand and assess risks. Additional guidance will be available in the forthcoming Version 3.0 of the DoD Cyber DT&E Guidebook.

3.1.1 Software Pipelines

In DevSecOps, the key values of visibility, repeatability, and consistency are critical to ongoing function, stability, and trust throughout the life cycle. These elements allow the various team perspectives of product owner, developer, security engineer, tester, and user to have a common foundation on which to build their process. It is this core orchestration framework, commonly called a pipeline, that provides this foundation.

A pipeline is a collection of tools, processes, and environments that chain together, much like a physical assembly line, with the output of one link in the chain becoming the input of the next. These pipelines can have varying levels of abstraction depending on their use case and the teams or organizations using them. Furthermore, they can functionally be loosely or tightly coupled given the function, team, or process that they serve.

Figure 3-2 illustrates a linear abstraction of the software factory with an integrated production environment, which does not depict many of the details, such as the iterative flow of the pipeline. The DoD Enterprise DevSecOps Fundamentals v2.5 guidance also illustrates the software factory with a separate production environment.



Source: DoD Enterprise DevSecOps Fundamentals, Version 2.5, p. 8

Figure 3-2. Generic Software Factory with Integrated Production Environment

As illustrated in Figure 3-2, a software factory may contain multiple pipelines. Each pipeline contains and defines a set of environments, tools, process workflows, and scripts. All pipelines coexist to produce an integrated set of production-quality software artifacts with minimal human intervention. The software factory automates many of the activities in the develop, build, test, release, and deliver phases of the DevSecOps life cycle.

Note: Independent government test team activities will occur within both the “Test” and “Integration & Pre-Production” environments. Additionally, the control gates depicted in Figure 3-2 are software factory decision points and support the DevSecOps life cycle nominal control gate decision points, as depicted in Figure 1-1.

Process Workflows and Automation

Process workflows specify T&E activities that occur during the SDLC. A process workflow should make clear the degree to which a T&E activity can be automated. A software pipeline should automate T&E activities to the maximum extent possible. If a T&E activity cannot be fully automated, a process workflow should describe the nature of human involvement.

Cyber DT&E routinely requires human testers because automated testing has not reached the point of automating a human cyberspace threat emulator. Cyberspace threats are ever present, ever advancing, and ever evolving. Automated adversarial test tools are available (e.g., CALDERA from MITRE) and, with planning and mature threat modeling processes, these tools can reduce the need for extensive hands-on cyber DT&E.

Each project should choose the appropriate software development pipeline and/or software factory that makes the most sense for the project's given application and associated environments. Specific tooling and processes should be chosen based on the needs of the project. Utilizing these processes throughout the life cycle of the product being developed, tested, deployed, and maintained must be considered when choosing a pipeline. The CyWG should analyze the continuous security monitoring of tools, processes, and environments during MBCRAs. These pipelines, tooling, and processes may have government test team impacts on training; see Section 6 for training considerations.

Continuous Integration and Continuous Delivery

CI/CD[1] techniques may be employed when using one or more pipelines to automate activities as repeatable and testable units that can be properly managed, audited, and monitored.

To ensure the key values of visibility, repeatability, and consistency from CI/CD in any project following DevSecOps methodologies, the project should utilize infrastructure as code (IaC) and configuration as code techniques to ensure environment parity. These techniques help produce reusable artifacts, system build definitions, configuration management code, system installation packages, and test configurations that can be deployed across multiple environments. These techniques also help reduce cybersecurity risks.

3.1.2 DevSecOps Ecosystem Design

During the DevSecOps ecosystem design activity, analysts design project-specific processes; select tools used in support of performing those processes; and select the platforms on which the tools and application will execute. Page 33 of the DoD Enterprise DevSecOps Reference Design states that the activity yields a process flow chart, but any process model can be used. Examples include the Integrated Definition for Function Modeling (IDEF0), Unified Modeling Language (UML) behavioral models, or combinations. The process model should define:

- Activities, some of which relate to testing, such as unit testing, integration testing, security testing, deployment, and monitoring.

[1] Note the difference between continuous delivery, where software is delivered to the pre-production environment for staging, and continuous deployment, where software is delivered directly to the production environment of operational deployment.

- Inputs and outputs of each activity, some of which are testing work products, such as test plans, test cases, test reports, code commits, test results, build artifacts, and deployment packages.
- Ordering among activities, some of which dictate when testing activities occur, such as code review before testing, testing before deployment, and security scans before release.
- Persons who perform activities (more precisely, roles of persons), some of which dictate when testers are involved.
- Types of tools used to perform activities, including those used in testing, such as tools for CI/CD, code quality analysis, and security testing.
- Activity triggers, some of which describe events that cause testing to begin, such as a code commit triggers an automated build, a successful build triggers automated tests, and deployment to staging triggers manual security testing.
- Activity controls and constraints, some of which describe constraints on when testing may begin.
- Decision points and control gates, some of which describe how test results affect subsequent activities.

IDEF0 was developed to support computer-aided manufacturing, where activity inputs and outputs are tangible products, and one manufacturing step must finish before another step can begin (or, alternatively, one manufacturing step must produce a complete product before that product can be used in another step). In software development, half-complete activities can produce results that other activities can use.

Table 3-1 highlights the key roles responsible for performing DevSecOps ecosystem design and their accountability for DevSecOps process flows, DevSecOps tool selection, and deployment platform selection. The process flows should make clear the points at which testing will occur.

The development team has a significant testing-related part in the DevSecOps ecosystem design activity. Architects will provide inputs on (1) the plan to integrate components into subsystems and, eventually, into an MVP or MVCR and (2) how the ecosystem should be set up to perform testing. Engineering is responsible for ensuring that testing is incorporated into process flows; engineers may work on the flow design or verify the work of others. The security champion scope emphasizes cybersecurity and cyber resilience, while managing risks (e.g., supply chain risks). Team members design workflows that strive to ensure the cyber performance of delivered products during development, delivery, and deployment.

Table 3-1. Key Roles Responsible for Performing Ecosystem Design

Role	DevSecOps Process Flow Chart	DevSecOps Ecosystem Tool Selection	Deployment Platform Selection
DoD Program Management Authorities	Review and sign off	Review and sign off	Review and sign off
DoD T&E Management			
• Chief Developmental Tester	Review and sign off	Review and sign off	Review and sign off
• Cyber Working Group (CyWG)	More information on CyWG participants and roles will be provided in the forthcoming Version 3.0 of the DoD Cyber DT&E Guidebook.		
Product Team Leadership			
• Project Manager	Review and sign off	Review and sign off	Review and sign off
• Product Owner	Design process flows	Review and sign off	Review and sign off
• Software Lead	Design process flows	Review and sign off	Review and sign off
Development Team			
• Architect	Provide testing-related input on integration strategy		
• Engineering	• Design workflows that ensure testable products • Design workflows that support gathering test data • Design workflows to report test results • Provide testing-related input on integration strategy	Identify and request T&E tools	Provide input on whether the platform can be instrumented
• Security Champion	• Ensure security is a developer focus • Provide cyberspace threat information • Deliver developer feedback to the CyWG • Serve on the CyWG	Provide inputs on Threat-informed ecosystem, processes, and tool selection	Provide input on continuous monitoring

Role	DevSecOps Process Flow Chart	DevSecOps Ecosystem Tool Selection	Deployment Platform Selection
Government Test Teams			
• DT&E Team	Design process flows to support government testing	Provide inputs on tools to support government testing	Provide input on whether the platform can be instrumented
• OT&E Team	Provide input on whether process flows ensure adequate pre-operational testing	Provide inputs on adequacy of DT&E records	
• Interoperability T&E Team	Provide input on whether process flows provide opportunities to test interoperability	Provide input on adequacy of DT&E environment to test interoperability	

Classified Network Awareness

A common challenge for DoD products is to develop software on unclassified systems and deploy it on classified systems. Workflows must be designed such that unclassified information can be transferred to classified systems with minimal effort, but classified information can never be accidentally transferred to an unclassified system. If software is to be deployed in a cloud, a Third-Party Assessment Organization is responsible for determining whether workflows satisfy DoD product acquisition and Federal Risk and Authorization Management Program (FedRAMP) enhanced FedRAMP+ processes. A cyber risk in these processes emerges as embedded malware from the unclassified development environment could end up in the classified environment.

T&E Tool Selection

Individuals with the development team's T&E role will contribute to the selection of tools to support T&E. These tools should support static application security testing (SAST), dynamic application security testing (DAST), and the infrastructure tools needed to automate unit, integration, and system tests. Required tools may also include data collection tools in the operational environment; these tools are useful if operational data are to be captured and made into tests.

Depending on the developmental testing approaches used, analysts may select specialized tools. For instance, a project using the STAT process to test a system will choose the appropriate T&E method based on test objectives, responses, and input factors. Furthermore, the tools used for analysis should be selected to appropriately address the test objective. For software that has been identified as performing a mission- or safety-critical function or as a critical component through criticality analysis, additional tools may be identified for software assurance testing or thresholds for SAST and DAST. Findings may be elevated to support control gate decisions, ensuring robust security and reliability of the software.

Some projects may find they need capabilities that existing software does not fully satisfy. Analysts should identify and document missing capabilities during the DevSecOps ecosystem design activity.

3.1.3 DevSecOps Environments

A test team may encounter several different environments in a DevSecOps acquisition program. Each environment is a separate configuration of hardware, software, and network components used for a specific purpose. The operational requirements of the program are critical factors in determining the number and types of representational environments needed to meet the mission. Test expertise is a vital contributor to the identification and construction of these environments.

The software and system under development dictate the need for each environment. Testing, however, can occur in any of these environments. Mission-support software-intensive systems, such as a Defense Business System, may require only the development, system test, and production environments. Mission- and safety-critical software-intensive systems, such as a warfighting system, may need additional environments to ensure proper testing, validation, simulation, stress testing, etc., to meet their operational requirements (e.g., system integration labs and hardware-in-the-loop test benches).

Program offices should identify the environments required in their ecosystem to meet mission objectives. CDTs must ensure configuration synchronization of these environments (including tools and integrated development environments) to support test, test data integrity, and interoperability. Some environments may need to be accredited to support test.

The DevSecOps Playbook 2.1 guidance identifies the following environments which can be found in a DevSecOps software factory: The development and test environments are where DT&E occurs; the pre-production environment is where OT&E occurs; and the production environment provides data that can be useful in subsequent DT&E and OT&E activities.

The DevSecOps environments are further defined and discussed below.

- Development Environment:
    - The development environment is for the development of code and is the environment where small teams that are focused on software units—Scrum, Kanban, etc.—

normally operate. Test teams, as part of these small teams, use the development environment to test software units.

- The development environment is where test-driven development automated testing occurs.
- The development environment may be used for integration, where software units from multiple development teams come together for testing at a higher level (e.g., the integration testing of features or capabilities) that is still below a complete system.
- Although the software developer may own the development environment, it is a good place for government test teams to observe and collect some test data that may reduce the need for tests that the government test teams might otherwise repeat later.

- Test Environment:
    - The test environment is where developmental and integrated testing are conducted at the system or SoS level.
    - The test environment should represent the production environment as closely as possible, including monitoring capabilities and the ability to simulate realistic system usage. It is normally the last opportunity for developmental testing before release to the operations team. The test environment may also support early user acceptance testing (UAT).
    - The test environment, when connecting to production environments of interfacing systems within the SoS, will have read-only access. The test environment might not, however, connect to these production environments. It may instead connect to test environments of interfacing systems, or it may be a self-contained environment that simulates interfacing systems.
    - If connecting to test environments of interfacing systems, program offices should plan early to coordinate access to interfacing system test environments. These interfacing systems provide the basis for initial interoperability testing. When testing within these environments, it is important to test not only the transmission and receipt of messages but also the effect of these messages on the interfacing system. If interfacing test environments are not available, the program must obtain or develop models or simulations of the interfacing systems and incorporate them into their test environment.
    - In DevSecOps, the test environment should be automated as much as possible for regression testing and the testing of routine and repeated human interface actions (UI/UX). Test automation is discussed in Section 4.3. The test environment will also

    be used for manual testing as necessary, including cyber DT&E. See Section 4.7 for discussion of the automated and manual aspects of cyber DT&E.

- Pre-Production Environment:
    - Sometimes called "staging" or "soaking," the pre-production environment is for UAT, which is a testing event that verifies the operation of the software in a production-representative environment before full release.
    - Because the operations team owns the pre-production environment, the team can maintain the pre-production configuration as an exact replica of the production environment using IaC. The pre-production environment may connect to other production systems or databases with read-only access. See the DoD Enterprise DevSecOps Reference Design for IaC details.
- Production Environment:
    - The production environment is used for live operations with real operators. It supports the acceptance of the software by the government and the "go live" decision to shift the new or updated software to live operations.
    - The production environment may store both the latest operating software version and the previous working software version to allow rapid roll-back in case significant problems are discovered in the operating software.

Environments are unique combinations of capabilities and configurations, whether physical or virtual. Tests should verify compliance with the National Institute of Standards and Technology (NIST) Special Publication 800-218, "Secure Software Development Framework (SSDF) Version 1.1: Recommendations for Mitigating the Risk of Software Vulnerabilities."

Additionally, specific security requirements are associated with each of these environments, which the tester needs to be aware of. The DoD Cloud Computing Mission Owner Security Requirements Guide "provides the technical security policies and requirements for applying security concepts to the DOD Mission Owner's cloud computing environment." that must be employed when test and development networks are directly connected to the Defense Information System Network.

Note: During the DevSecOps deploy phase, new software versions are deployed to the operational/production environment. During the DevSecOps operations phase, users and operators use the software in the operational/production environment to conduct their missions.

3.1.4 Developmental Test Tools and Activities in DevSecOps

Test tools help decrease the time (schedule) to complete testing, thereby enabling faster capability deployment to the user community.

Version 2.2 of the DevSecOps Fundamentals Guidebook: DevSecOps Activities and Tools contains a list of tools and activities associated with each phase of the DevSecOps life cycle and is a valuable reference for test teams.

Automated Tools

One of the DevSecOps key attributes is the extensive use of automated tools, including automated test tools, to accelerate the development and deployment of software to the user. Test tools include those that automate (1) test management and (2) test execution. Government test teams must be trained in the use of these tools and use them as part of the DevSecOps process.

Test Management Tools

Test management tools automate the process of test planning, scheduling, tracking, and event reporting. When combined with SE tools, they support a Requirements Traceability Matrix (RTM) to track capabilities to features, user stories, test cases that verify the user story, and test events. Some test automation tools have internal RTM capabilities that also cover enablers and nonfunctional requirements.

The CDT will require identification of a test management tool early in the program's evolution to begin test planning. A challenge may occur during the identification of test management tools if the selected contractor employs a different tool suite from the CDT recommendation and includes it in the contract proposal. In this case, the CDT may need to adopt the contractor's tool or figure out how the two tools can interoperate. If two tools are used, developing a common understanding of terms is critical.

A subset of test management tools addresses test input data management. The "data" in this case are sample inputs from the production system that can be used to either drive test cases or evaluate the impact of testing on data resident in the system. Automated data generation tools help mask sensitive data (e.g., privacy or security) or create large data sets if production data are not available.

Test Execution Tools

Test execution tools automate the process of executing test cases or procedures.

The CDT should work with the contractor to fully understand the contractor's tools. The CDT should avoid duplicating previously successful testing and not become a choke point that interferes with the contractor's ability to maintain its delivery cadence. Because the contractor can be a government organization, government components may need to establish agreements to share and examine each other's tools and environments.

A subset of test execution tools addresses nonfunctional requirements such as system responsiveness, availability, stability, and scalability. These tools normally stress the system with high loads or other system stressors (e.g., negative testing) that can lead to failures. These failures identify areas of the software that require further development to ensure that similar loads during operations will not negatively impact the system.

Test Tool Challenges

A pipeline will be filled with automation tools supporting multiple development specialties. Interoperability among the tools to exchange data will be a challenge. These challenges may include tool integration (compatibility issues, limited application programming interface (API) support); data and environment (data format incompatibility, platform dependency); and test execution and reporting (execution synchronization, reporting standards).

One approach for mitigating these challenges is to use an enterprise-level digital infrastructure or software factory for development where all the tools have already been integrated and tested. Both OSD and the Military Services are working to make these resources available to acquisition programs.

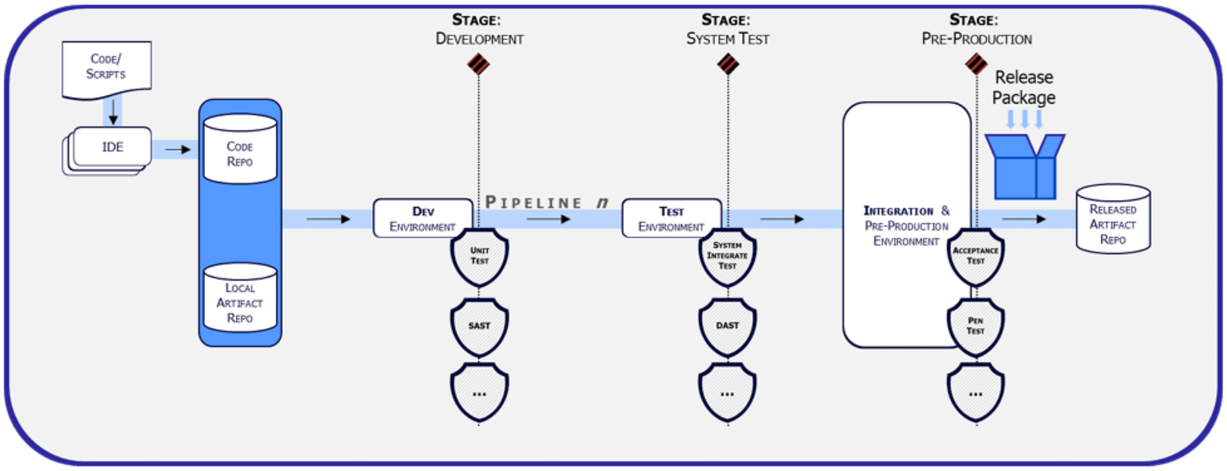
When considering enterprise services, programs should determine whether the tools available meet program requirements for cybersecurity and program protection. Some factories will offer to customize pipeline tools. Programs should establish security pipelines to enable more rigorous testing needed for critical software.

The Automated Software Testing Implementation Guide provides comprehensive guidance for managers and practitioners. The guide also sheds light on automated software testing pitfalls and best practices.

Tests Supporting Control Gate Decisions

T&E assesses whether the system was built right, and the right system was built. Unit tests, static application security tests, system integration tests, dynamic application security tests, acceptance tests, and penetration tests inform these assessments.

As illustrated in Figure 3-3, control gates (represented by diamonds) identify where test teams can provide decision makers with test data (indicated by the shields). These test data are necessary to make informed decisions (go/no-go) between DevSecOps environments. From the “Development Environment” to the “Test Environment,” T&E outputs can inform the Development control gate decision. From the “Test Environment” to the “Integration & Pre-Production Environment,” T&E outputs can inform the System Test control gate decision. From the “Integration & Pre-Production Environment” to the “Release Package,” T&E outputs can inform the Pre-Production control gate decision.



Source: DoD Enterprise DevSecOps Fundamentals, Version 2.1, p. 16

Figure 3-3. Notional Expansion of a DevSecOps Software Factory with Illustrative List of Tests

In accordance with DoDI 5000.83, “Technology and Program Protection to Maintain Technological Advantage,” stipulates the following:

- Programs should incorporate automated software vulnerability analysis tools throughout the life cycle of the system, including during development, OT, operations and sustainment phases, and retirement, to evaluate software vulnerabilities.
- Pipelines and control gates should be tailored to mitigate risks associated with known and exploitable vulnerabilities to provide a level of assurance commensurate with technology, program, system, and mission objectives.”

3.2 Metrics

Paragraph 3.3.b.(11) of DoDI 5000.87 states, “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."

Metrics give insights into what is happening at all stages of the DevSecOps pipeline, from design to development to deployment. Metrics are essential to the success of DevSecOps for several reasons:

- Visibility and Transparency. Metrics offer insight into the SDLC, fostering transparency among DevSecOps teams.
- Measurable Goals. Metrics empower teams to establish quantifiable objectives and monitor advancement, facilitating the identification of areas suitable for enhancement.
- Informed Decision Making. Through the assessment of key performance indicators, stakeholders can leverage data-driven insights to make informed decisions and identify opportunities for enhancement and resolution of bottlenecks.
- Security Posture. Consistent monitoring of DevSecOps metrics enables early identification and mitigation of risks, strengthening the security posture of a project team.
- Feedback Loop. Metrics strengthen the feedback loops that are essential to DevSecOps.

Besides the basis of a metric, project teams should consider what they need the metric to measure. For example, the velocity or acceleration of user gains might be a more useful metric than total number of users. Metrics vary from project to project, but the most helpful metrics are those with the potential to drive mission decisions.

Defining metrics to be utilized on a project can be difficult. A common mistake is to include too many metrics to avoid missing anything. But doing so can lead to ignoring or mistrusting the value of metrics that do not meet a specific need. A lesson learned is to take the time to determine the data needed to measure project success. Project teams might add or change metrics throughout the SDLC, but understanding the purpose of the metric from the beginning facilitates that process.

3.2.1 Data Generated by DevSecOps Practices

DevSecOps replaces practices that in the past have been labor-intensive (manual) and error prone. With DevSecOps implementations, the automation of CI, CD, and IaC all produce large amounts of data as a by-product. Likewise, modern development tools supporting the Agile development toolchain can produce significant amounts of data addressing the creation and operation of the software product(s).

DevSecOps metrics can originate from many different tools used at points across the entire development pipeline. Examples of tools that generate DevSecOps metrics include:[2]

- Build and release tools such as GitLab, Azure DevOps, Octopus Deploy, and Jenkins.
- Configuration management tools such as Ansible, Puppet, and Chef.
- Test automation tools such as Selenium, Worksoft, and Kobiton.
- Deployment and monitoring tools such as Nagios, Splunk, and SolarWinds AppOptics.

3.2.2 Metric Definition and Application

Software metrics empower stakeholders involved in software development by providing vital insights into various aspects of software projects. Metrics define how to measure system properties or performance to inform decisions throughout the development, production, fielding, and deployment life cycle. Metrics are not perfect, however. Measurements are just numbers, and those numbers are meaningless without proper human interpretation and understanding. Although metrics provide data, they offer no guidance or insight on how to proceed; they illustrate the what, but not the why. The power and risk of metrics lie in how the program collects, interprets, and uses that data.

Several metrics offer valuable insights into the effectiveness, security, quality, and performance of the software being developed and tested. These metrics offer a comprehensive view of the security posture, testing efficacy, compliance adherence, automation levels, collaboration efforts, and operational impacts within the SDLC. They enable teams to track progress, identify areas for improvement, and ensure the continuous enhancement of software development practices.

Doing an internet search on "agile metrics" (Microsoft Bing) will return nearly 1.6 million results. Doing an internet search on "DevSecOps metrics" (Microsoft Bing) will return nearly 1.8 million results. The takeaway is clear: A lot of material is available in the public domain providing definitions of what these metrics are; why they are used; and how they can assist with project management during the planning, development, deployment, and operation of Agile/DevSecOps projects.

The following excellent resources should be consulted as part of the overall DT&E approach to DevSecOps metric definition and application.

[2] These tools are not identified as an endorsement but strictly as an example of the toolsets available in the information technology industry.

OUSD(A&S) Agile Metrics Guide: Strategy Considerations and Sample Metrics for Agile Development Solutions

The OUSD(A&S) Agile Metrics Guide: Strategy Considerations and Sample Metrics for Agile Development Solutions offers "guidance and recommendations related to actionable metrics for Agile products and services. It provides Agile development teams, testing teams, and program management teams with an understanding of how Agile metrics differ from metrics used in traditional waterfall approaches, and identifies key Agile metrics, their benefits, and drawbacks (noted as challenges). Specifically, the document presents guidance on the types of metrics available when executing Agile projects, the benefits of each metric, and actions that can be taken because of the data being shown."

This document covers the following types of Agile metrics: process, quality, capability delivery, DevSecOps, cost, and value assessment.

OUSD(R&E) Software Engineering for Continuous Delivery of Warfighting Capability

The OUSD(R&E) Software Engineering for Continuous Delivery of Warfighting Capability guide was developed 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."

Metric categories addressed include process efficiency; technical performance and mission effectiveness; software quality; software productivity; and CI, test and release, and operations.

DAU Program Management Metrics and Reporting

The DAU Program Management Metrics and Reporting Website defines metrics supporting the following areas: process efficiency, software quality, software development progress, cyber security, cost, value, and earned value management.

GAO Agile Assessment Guide: Best Practices for Adoption and Implementation

The GAO Agile Assessment Guide: Best Practices for Adoption and Implementation was developed to "ascertain best practices for Agile software development from leading practitioners and to develop standard criteria to determine the extent to which agency software development programs meet these practices. These best practices center on Agile adoption, execution, and control."

GSA DevSecOps Guide

The GSA DevSecOps Guide Website describes a framework that "labels individual metrics as "High-Value" or "Supporting." High-Value metrics are those that provide the most critical insight into the performance of a DevSecOps platform and should be prioritized for implementation." Supporting metrics are those that a team may find useful to gain insight into and improve their DevSecOps platform "performance" to meet stakeholder requirements.

PSM Continuous Iterative Development (Agile) Measurement Framework

The Practical Software and Systems Measurement (PSM) Continuous Iterative Development (Agile) Measurement Framework (see the PSM Continuous Iterative Development (Agile) Measurement Framework Website) provides recommendations for the measurement of continuous iterative developments (CID). The report includes a CID measurement framework detailing common information needs and measures that are effective for evaluating CID approaches. The information needs address the team, product, and enterprise perspectives to provide insight and drive decision-making. The framework also identifies and specifies an initial set of measures identified as practical measures to address these information needs.

SEI Current State of DevSecOps Metrics

In "The Current State of DevSecOps Metrics," Bill Nichols (2021) discusses "ways in which DevSecOps practices yield valuable information about software performance that is likely to lead to innovations in software engineering metrics."

3.2.3 Key Software Test Metrics

Software test metrics are essential for assessing the quality of the SUT and the effectiveness of the testing process. These metrics help teams understand the status of testing activities; identify areas for improvement; and make data-driven decisions. By tracking these software test metrics, product teams can gain insights into the effectiveness, efficiency, and quality of their testing processes, enabling them to improve testing practices and deliver high-quality software products.

The following list provides a sampling of key software test metrics:

- Test Coverage: Measures the extent to which the code or SUT has been exercised by test suite runs. A higher percentage indicates more source code executed during testing, which suggests a lower chance of containing undetected bugs. The four primary types of test coverage are requirements, functional, product, and test execution. The Software Testing Stuff Website provides additional content on "What is Test Effectiveness and Efficiency?"

- Test Automation Coverage: Measures the percentage of executable code executed by automated tests. The ratio of test automation coverage to test coverage is the percentage of covered code executed automatically. By striving for a value as close to 100 percent as possible and practical, the SW ITT can improve testing efficiency and reduce the manual effort needed to conduct testing.
- Percentage of Automated Tests: Measures the percentage of all tests that are automated. By striving for a value as close to 100 percent as possible and practical, the SW ITT can improve testing efficiency and reduce the manual effort needed to conduct testing.
- Customer Reported Defects: Tracks defects reported by customers or end-users' post-release. Monitoring customer-reported defects provides feedback on the quality of the released software and helps prioritize future testing efforts.

Additional software test metrics utilized across the SDLC include the following:

- Defect Density: Calculates the number of defects identified per unit of code or feature. It helps assess the quality of the software and identify areas where defects are most prevalent. Combined with defect reports from operational use, defect density provides information on the adequacy of test resources allocated to code units.
- Defect Removal Efficiency: Measures the effectiveness of the testing process in identifying and removing defects before software release. It is calculated as the ratio of defects found during testing to total defects found (including those found post-release).
- Regression Test Suite Effectiveness: Evaluates the effectiveness of regression tests in identifying regression defects. It measures the percentage of regression defects detected by the regression test suite.
- Test Automation Execution Time: Measures the duration to execute automated tests. The ratio of test automation execution time to test execution time is the percentage of time spent on automated tests. Knowing this value helps plan future testing events.
- Test Case Maintenance: Measures the effort required to create, update, and maintain test cases over time. It helps assess the scalability and maintainability of the testing process.
- Test Effectiveness: Evaluates the effectiveness of software testing by capturing the number of bugs identified by the testing team and the number of overall bugs found in the software. It is an assessment of the defect capturing ability of a test suite. If the test effectiveness is high, it means the tests are capable of tracking most of the bugs, and the overall test suite will require less maintenance. The Software Testing Stuff Website provides additional content on "What is Test Effectiveness and Efficiency?"

- Test Environment Stability: Tracks the stability and availability of the test environment. Unstable test environments can lead to unreliable test results and delays in testing activities.
- Test Execution Time: Tracks the time taken to execute test cases. Monitoring test execution time helps optimize testing efficiency and identify potential bottlenecks. In a software factory, the important consideration is to optimize test execution time per commit, which can involve executing multiple unit tests, integration tests, and system tests. Overall test execution time can sometimes be reduced by studying test execution times of these components and either rewriting test cases to use less time or changing test execution strategies to reduce how often costly tests are executed.
- Time to Detect Defects: Measures the amount of time to detect a defect from the introduction of that defect into the system. The mean time to detect (MTTD) defects metric is then the average of these observations. Lower MTTD indicates faster defect detection and response.
- Time to Repair Defects: Measures the amount of time to correct defects once they are identified. The mean time to repair defects metric is then the average of these observations. Lower mean time to repair indicates faster defect resolution and shorter feedback loops.

3.2.4 Key Software Security Metrics

Software security metrics focus on measuring the integration of security practices within the DevSecOps pipeline. These metrics help product teams ensure that security is prioritized throughout the SDLC and that vulnerabilities are identified and addressed effectively.

By tracking and analyzing these metrics, product teams can assess their DevSecOps software security practice effectiveness; identify areas for improvement; and continuously enhance their security posture throughout the SDLC.

The following list identifies software security metrics utilized across the SDLC:

- Security Vulnerabilities Identified: A key security metric that measures the number and severity of security vulnerabilities discovered during development and testing. This metric helps evaluate the effectiveness of security testing processes.
- Automated Security Testing Coverage: Evaluates the extent to which security testing is automated within the DevSecOps pipeline. Higher automation rates streamline testing processes and reduce manual effort.

- Compliance Adherence: Tracks adherence to industry standards and regulatory requirements relevant to the application's domain. Ensuring compliance helps mitigate legal and regulatory risks.
- False Positive Rate: Evaluates the accuracy of security testing tools by measuring the rate of false positives generated. Minimizing false positives reduces the time spent investigating and addressing nonexistent issues.
- Percentage of High-Risk Vulnerabilities Mitigated: Focuses on addressing critical security vulnerabilities promptly. Prioritizing high-risk vulnerabilities helps mitigate potential security threats more effectively.
- Percentage of Security Controls Implemented: Measures the percentage of security controls and best practices implemented within the development and deployment processes. Higher implementation rates enhance the overall security posture.
- Security Awareness Training Completion Rate: Tracks the completion rate of security awareness training among development and operations teams. Well-trained teams are better equipped to identify and mitigate security risks.
- Security Incident Response Time: Measures the time taken to respond to and mitigate security incidents from their detection to resolution. A shorter incident response time minimizes the impact of security breaches and enhances overall security resilience.
- Security Testing Coverage: Evaluates the breadth and depth of security testing across the application. Comprehensive test coverage ensures that critical security vulnerabilities are identified and addressed.
- Time to Detect Security Incidents: Measures the time to detect security incidents from the occurrence of the event to its identification. The MTTD security incidents metric is then the average of these observations. A lower MTTD indicates quicker detection and response to security threats.
- Time to Remediate Vulnerabilities: Tracks the time taken to address identified vulnerabilities from discovery to resolution. Shorter remediation times indicate efficient response to security issues.
- Time to Resolve Security Incidents: Measures the time taken to resolve security incidents once detected. The mean time to resolve security incidents metric is then the average of these observations. Lower mean time to resolve indicates efficient incident response and mitigation.

4 DT&E in DevSecOps – Phase Agnostic Continuous Activities

An understanding of DT&E in the requirements and SE processes sets a foundation for a test-driven design that ensures delivered capability is mission effective while emphasizing that requirements are measurable, testable, and achievable.

In DevSecOps, DT&E is a part of the continuous process that requires integration between development and testing to achieve high product quality. Testing is executed during code development and therefore requires increased collaboration between developers and independent test teams. Development teams will perform lower-level tests such as unit tests, whereas independent test teams will perform integration and acceptance tests. Independent test teams need to continuously communicate and collaborate with developers to facilitate developing the scripts for integration and acceptance tests.

Key phase agnostic continuous activities associated with DT&E in DevSecOps that are addressed in this section include the following:

- Test Considerations. Test teams should be involved in developing operational requirements to ensure the requirements are measurable, testable, and achievable. These operational requirements include software requirements analysis, Agile requirements, flow and backlogs, cyber requirements, and authorization to operate (ATO)/continuous authorization to operate (cATO) requirements.
- Test Strategy and Planning. The test strategy and planning should describe how test teams are integrated into development teams charged with producing working software using DevSecOps. Different elements should be included in the test strategy and planning such as the IDSK, product roadmap, and software acquisition pathway.
- Test Automation Tool Selection. A foundational strength of DevSecOps is the set of automated and integrated tools that allow for rapid development, testing, cyber T&E, and deployment processes. Several characteristics are critical to good tool selection including trust and transparency, data format standardizations, integration, cyber capabilities, government software considerations, DE, and MBSE concerns.
- Agile Framework and Software DT&E Integration. In Agile methodologies, DT&E activities are integrated into the development process to ensure that quality, testing, and evaluation occur iteratively and concurrently with development. This iterative approach allows for continuous improvement, early issue detection, and faster adaptation to changes, ultimately leading to the delivery of higher-quality software.

- Cyber and Software Assurance T&E. Program offices adopting DevSecOps processes to develop and deliver applications should incorporate cyber developmental testing activities. Cyber DT&E includes automated security, evaluation of the scan results, and manual testing, and informs software assurance determinations.

4.1 Test Considerations During the Strategic Planning Phase

Although this guidebook focuses on the DevSecOps life cycle depicted in Figure 1-1, DoD software development efforts will have a strategic planning period before entering the DevSecOps cyclic process. During this period, strategic documents such as the acquisition strategy, high-level requirements documents, schedules, UAs, and metrics are defined. This strategic planning should not be confused with the plan phase in the DevSecOps life cycle, which is focused on planning sprints and releases.

4.1.1 Requirements Analysis

Test teams (including the cyber test teams) should be involved in developing high-level operational requirements to ensure the requirements are measurable, testable, and achievable. This involvement also gives the test teams an opportunity to fully understand the users' intent and need. Testers must be aware of the problem space early and up front.

Requirements lay the groundwork for successful software acquisition by aligning stakeholder expectations, guiding development efforts, managing scope, fostering communication, mitigating risks, and ensuring quality. Requirements are the cornerstone for effective project management and the foundation for T&E and successful software delivery.

DoDI 5000.87 defines the highest level of requirements to be the CNS. Other acquisition pathways—for example, the major capability acquisition pathway described in DoDI 5000.85, "Major Capability Acquisition"—use other terms to define the highest-level requirements document, such as the Joint Capabilities Integration and Development System (JCIDS) CDD (including the Information Systems CDD and Software CDD variants) (see the DAU Glossary of Defense Acquisition Acronyms and Terms Website for more detail). For embedded software, the high-level requirements may be embedded in the hardware requirements document. The Agile methodology defines the foundation of requirements at the software development level as the user story. Regardless of the title given to the high-level requirements document, if the software is being developed using DevSecOps, the lowest requirement will likely be a user story.

The Manual for the Operation of the JCIDS introduces the new Software Initial Capabilities Document (ICD). Paragraph 3.2.2.1.1. of Appendix A to Enclosure A states, "Software capability requirements are required to be submitted to the Joint Staff for determination of Joint Equities prior to seeking an acquisition decision. If joint equities are present, the Software ICD will be staffed and validated by the Joint Staff utilizing an expedited process." Test teams and systems engineers should be aware of this requirement, which may also apply to the software acquisition pathway.

Test teams should be aware of the following:

- Words such as enhanced, improved, reduced, faster, etc., are not measurable unless the baseline from which to measure is defined.
- Any use of "percent" is not measurable unless the "100 percent to measure from" value is defined.
- Any "timed requirement" is not testable unless the start and stop times are clearly defined. For example, "Upon detection of unauthorized data, the system will render harmless the unauthorized data within xx seconds without delaying system processes by more than yy seconds."
- Requirement statements using words such as "ensure a user-friendly interface" or "ensure the software is aesthetically pleasing" focus on the subjective aspect of user-friendliness or aesthetics, which can vary greatly from person-to-person.
- Certain concepts make the requirement unachievable. An example of a requirement that is not achievable is the demand for a software program to be 100 percent bug free or for the system to be secure. Although both are desirable goals, these are virtually impossible to guarantee because of the complexity of software systems and the ever-changing nature of technology, threats, and user needs.
- Requirements should not be ambiguous. Ambiguous: The <system or API> shall prevent unauthorized data connections to the <system or API>. Improved: The <system or API> shall detect unauthorized data connections to the <system or API> within <constraints> without <restraints>. Constraints dictate an action, thus restricting freedom of action. Restraints prohibit an action, thus restricting freedom of action.

4.1.2 Software Requirements Analysis

The basic process of SE is the definition of high-level objectives and the decomposition of those objectives through multiple sublevels to a point where engineers and software developers can develop software, and test teams can verify that the software meets the requirements from the lowest to the highest level.

DoD policy is very flexible in terms of describing requirements documents. Likewise, Agile literature has many names for requirements, such as visions, capabilities, epics, themes, features, enablers, scenarios, workflows, story maps, tasks, and user stories. Test teams and systems engineers must recognize that relevant interface control elements such as interface control documents and/or API definitions are also requirements.

This guidebook presents a notional software requirements hierarchy with supporting documents to describe the overall capability that the test teams need to evaluate. Different programs may use different terms. To ensure that everyone within a program is using the same terms consistently and understands how the requirements relate to each other, it is critical to define the software requirements hierarchy for the program early in the SDLC and document it in an acquisition strategy, an SE plan, or a test strategy.

Table 4-1 lists this notional structure and additional supporting documents that may expand or clarify the requirements.

Table 4-1. Software Requirements Hierarchy and Supporting Documents

Software Requirements Hierarchy	Supporting Documents
• Capabilities/Sub-Capabilities • Epics • Features • User Stories	• Architectures • Capability Needs Statement (CNS) • Concept of Operations (COO) • Mission Threads/Business Processes • Nonfunctional Requirements • Software Initial Capabilities Document (ICD) • Workflow Diagrams

Test teams must be able to understand these requirements and documents and be able to trace the requirements from the highest level to the lowest level, which will be the basis for detailed test planning. Test teams defined by the SW ITT must also be able to trace each test event back up to the highest capability to know when their testing is sufficient to verify/validate complete satisfaction of the high-level capabilities. Test teams should be involved early in developing all requirements to ensure they are clearly stated, mission relevant, and sufficiently complete, ultimately supporting measurable, testable, and achievable criteria.

4.1.3 Cyber DT&E Requirements

Cyber DT&E requirements in DevSecOps need to emphasize the functional cyber performance requirements for the software-based system. Cyber requirements should be handled the same as all requirements in terms of being part of Agile, DevSecOps software requirements and analysis.

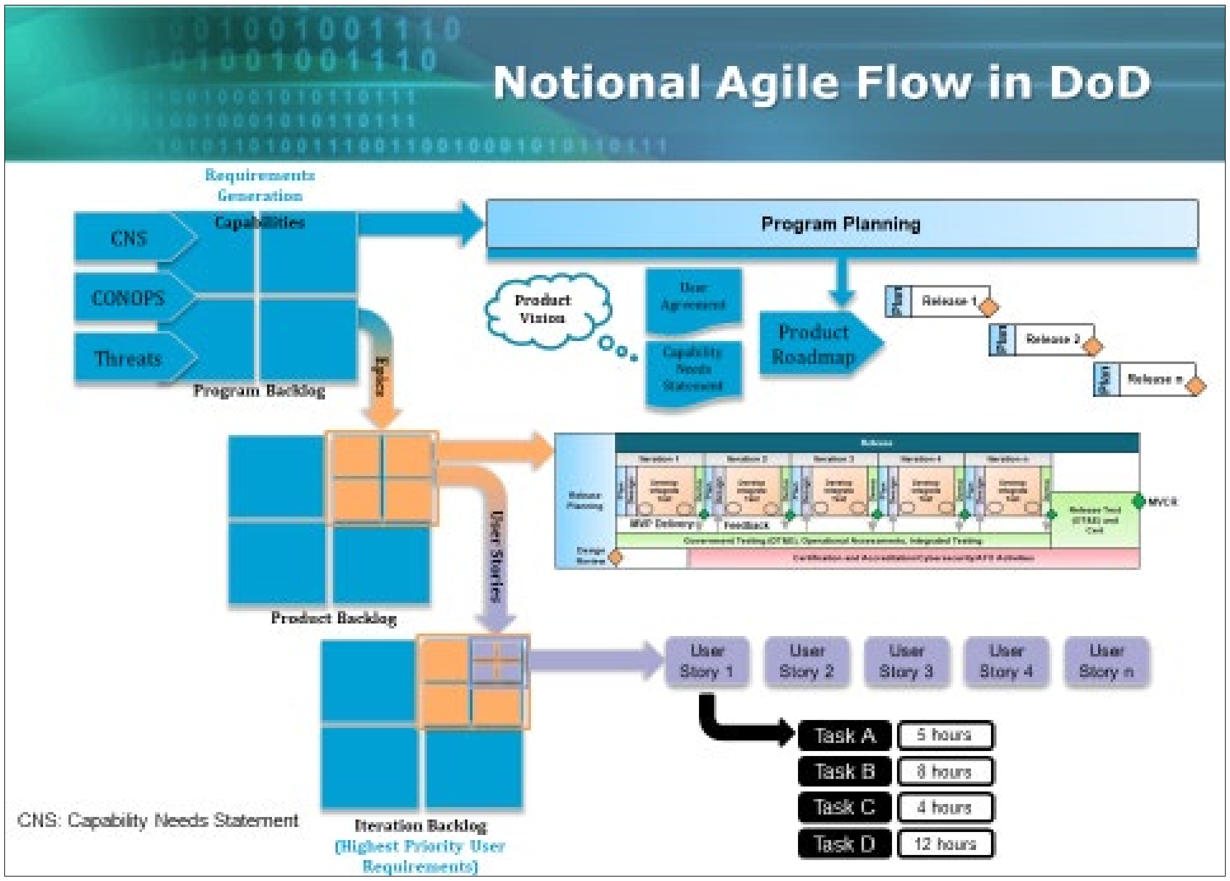
As with all requirements addressed by the DT&E teams, the CyWG should carefully review system documentation and provide advice on cyber performance requirements. These requirements must also be measurable, testable, and achievable while addressing performance to satisfy cybersecurity standards, cyber resilience, and operational requirements. The CyWG will inform test conditions, environments, and methods and the prioritization of testing.

Some of the more important cyber requirements address (1) performance of hardware and software; (2) anti-tamper; (3) electronic warfare; (4) threat intelligence; and (5) operational issues. More information on sources of cyber DT&E requirements will be available in Appendix B in the forthcoming Version 3.0 of the DoD Cyber DT&E Guidebook.

Program protection requirements are an integral part of cyber requirements and should be included in the overall DevSecOps requirements analysis. DoDI 5000.83 states, "To design, develop, test, and acquire systems that can successfully operate in the face of threats, to include cyber threats, as well as in denied environments, lead systems engineers will include cybersecurity, security, and other system requirements into system performance specifications and product support needs." These requirements support cyber resilience. Requirements derivation methods, such as system modeling and analysis, security use and abuse cases (McDermott and Fox 1999) or misuse cases, criticality analysis, and vulnerability analysis are used to derive security and exportability requirements that are sufficient to minimize vulnerabilities introduced by design, implementation, system interfaces, and access points. The testing of program protection requirements should be automated where possible and completed along with the larger cyber DT&E efforts.

4.1.4 Notional Agile Requirements Flow and Backlogs

Figure 4-1 illustrates the notional flow of requirements in Agile.



Source: DAU Course ACQ 1700: Agile for DoD Acquisition Team Members

Figure 4-1. Notional Agile Flow

The program backlog, shown in the upper left corner serves as a repository of yet-to-be-delivered capabilities and features. The program backlog serves as a repository of requirements, expressed specifically in terms of how those requirements will satisfy the operators in a delivered, deployed system. It is not necessarily expressed in the same way as a traditional requirements document. Its capability-oriented focus may lead to a less-cohesive overall picture of the system under development. If written in this manner, it does not eliminate the need for a traditional requirements document, expressed in terms of how a system will satisfy program objectives. Instead, it complements the requirements document by providing a view into capabilities to be delivered. The program backlog is especially important during the first iteration because it shows the capabilities necessary to achieve an MVCR. Each entry in the program backlog is a traceable link between the overall requirements and the eventual plans to test the implementation of those requirements.

Capabilities, epics, and/or features from the CNS are entered into the program backlog. Capabilities are decomposed into additional epics and/or features and captured in the product backlog. The product backlog is further decomposed into user stories for the iteration (or sprint) backlog. Software development teams pull the user stories from the iteration (or sprint) backlog to develop code during sprints. Any defects or failures discovered during testing are returned to the backlog at the appropriate level for reprioritization and resolution.

4.1.5 Authorization to Operate/Continuous Authorization to Operate

An integral part of DevSecOps is the ability to rapidly respond to changing threats through the continuous integration and delivery of cybersecurity, resiliency, and survivability. The cyber landscape, with its advanced persistent threats and vulnerabilities, requires a shift from the current assessments and authorizations process to a continuous monitoring and assessment of risk, using automation and dashboards that assist in managing the risk.

This approach has advantages to DevSecOps by accelerating software deployment with continuous authorization. In many DoD Components, obtaining an ATO is typically the longest step in developing and deploying software. Delivering new features rapidly requires an authorization process that can keep pace with continuous change for a developing capability.

As discussed in the DevSecOps Continuous Authorization Implementation Guide, cATO "is the state achieved when the organization that develops, secures, and operates a system has demonstrated sufficient maturity in their ability to maintain a resilient cybersecurity posture that traditional risk assessments and authorizations become redundant. This organization must have implemented robust information security continuous monitoring capabilities, active cyber defense, and secure software supply chain requirements to enable continuous delivery of capabilities without adversely impacting the system's cyber posture." An organization with a cATO continuously assesses and deploys systems that meet the risk tolerances for use within an authorization boundary. A cATO moves away from a control assessment approach to focusing on continuous risk determination and authorization through assessing, monitoring, and risk management.

Systems seeking a cATO must have already achieved an ATO and have entered the RMF monitor stage. Continuous authorization for DevSecOps includes additional aspects, such as assessing the team and a DevSecOps platform for supporting continuous risk monitoring.

For more information on the cATO process, see the DevSecOps Continuous Authorization Implementation Guide.

4.2 Test Strategy and Planning

4.2.1 Introduction

DoDI 5000.89 states that "the PM will use the TEMP, test strategy, or other pathway-appropriate test strategy documentation as the planning and management tool for the integrated T&E program." Additionally, DoDI 5000.89 states that "a TEMP may be waived or other tailored test strategy documentation be specified for certain acquisition pathways. In cases where a TEMP is not needed, early briefings to Service stakeholders … are required to facilitate cross-organizational alignment and subsequent approval of test planning documentation." More information on test strategy and planning will be provided in the forthcoming DoDI 5000.DT.

The outcome of the strategic planning is a TEMP or high-level test strategy documentation that guides the testing from user story demonstrations to final UAT before deployment for OT&E of the fielded system. The test strategy should describe how test teams are integrated into development teams charged with producing working software using a DevSecOps pipeline. This integration of teams provides T&E information throughout the software's development and allows flexibility when stakeholder priorities shift. The test strategy should include a discussion of the following elements:

- Mission- and Safety-Critical Functions. Software functions in this category are those whose failure would likely cause a failure of the mission. These functions should derive from the highest-priority capability needs.
- Cyber. Testing should evaluate the cybersecurity and cyber resilience performance of the system, including software systems as well as hardware platforms with embedded software.
- Human Factors. Software applications are often intended to be used by humans. Therefore, software testing must include an evaluation of HSI and how people interact with the software. Typical measures for human factors evaluations include system usability, user workload, and user trust in the system.
- Doctrine. Tests must address whether the doctrine and the tactics, techniques, and procedures (TTPs) incorporate the software capability.
- Documentation and Training. Users need training and documentation to operate the system. Test teams should evaluate the associated documentation and training, as well as the process for updating both, with regular and frequent changes to the software. This process should include getting new and updated information to all users.
- Change Management. Test teams should evaluate the software version and update control and deployment processes from the user perspective.

- Personnel, Logistics, and Manpower Supportability. Test teams should evaluate whether sufficient logistics and manpower exist to iteratively deploy and utilize the new software and support software development. The intended user population should have adequate training and embedded help functions to effectively use the software and receive software updates.
- System Reliability, Availability, and Maintainability (RAM). RAM metrics should be captured with respect to both the hardware and software, as applicable. The test strategy should also reference the Failure Definition and Scoring Criteria, which defines how to categorize the defect cause (hardware, software, user error, etc.); what is considered a failure that affects system availability; what constitutes a system downing event and associated restoral activity; and what failures constitute a mission failure.
- Interoperability. Software systems almost always work in a network of interfacing systems with which the system under development must work. Special attention should be placed on planning interoperability testing with these interfacing systems. The TEMP or TES documentation should address how to evaluate both the timely, accurate, and complete transfer of data and the use of the data sent to or received from the interfacing system. Interoperability T&E may include the T&E of APIs (see the API Technical Guidance for more details). The TEMP or TES documentation should further identify the environments in which the interoperability testing will occur and, if not on the production environment, how the simulated interfaces will be verified, validated, and accredited for operational evaluation. See DoDI 8330.01 for additional information on interoperability requirements and testing guidance for information technology (IT).
- Scalability. Before system deployment, tests should simulate high user load on the system in a representative network. Scalability evaluations may also include the continued software deployment strategy and the potential of the system to support the entire user base. Additionally, scalability testing should include technical considerations of the system, including databases and data size.
- Safety. The test team should plan safety assessments and testing as applicable to the software system including considerations for the platforms on which the software runs. For testing embedded software, safety considerations for the platforms on which the software runs should be included.
- Adequacy of Infrastructure. Software cannot perform without adequate computing processing and networks, including bandwidth. Whether the software system uses dedicated hardware or resides in an existing server facility, the objective hardware and representative network conditions (including bandwidth) need to be tested to ensure they are adequate to run the software. If the software is on a cloud environment, then the cloud contract should include adequate services to run and monitor the software. Test teams

should further evaluate the software development and deployment process accounting for supply chain risk management (SCRM) for updating the system as needed by the user community.

4.2.2 Integrated Decision Support Key

In DevSecOps, decisions can be delegated to lower levels and made iteratively and more frequently. Each DA needs different information depending on the environment and circumstances. The information is an essential part of DT&E planning and should include the IDSK.

The IDSK aligns the component- or system-level planned DT&E activities that generate data from contractor, developmental, integrated, and operational tests and M&S events to the capability evaluation and technical, programmatic, and operational decisions. The capability delivery community stakeholders (e.g., S&T, P&E, acquisition, and capability portfolio management) should use the IDSK's decision–evaluation–data critical thought process to develop a decision support strategy for the full capability development and delivery life cycle.

Programs using MBSE during system development or new start programs as directed by DoDI 5000.97, "Digital Engineering," should develop a model-based IDSK. More information on IDSK will be provided in the forthcoming DoDI 5000.DT.

4.2.3 Product Roadmap

A stable product roadmap in Agile is the goal but is flexible to change with warfighter needs. A 1-year plan can change with warfighter needs, and the Agile framework allows that change to happen in rapid iterations.

After receiving the product vision (from the CNS or other requirements document), the PM, in collaboration with the PrjMgr and product owner, determines what capabilities and features need to be built and when. These capabilities and features are the foundation of the product roadmap.

The product roadmap defines the planned evolution of the solution capabilities and architecture; is updated periodically by DoD program management and the product team; communicates the capabilities/feature sets targeted for delivery at discrete times in an iterative fashion; and aligns with warfighter/end user needs. This input will define which key DT&E activities are required. See the DAU Product Roadmap Website for additional information.

4.2.4 Software Acquisition Pathway Programs

For programs using the software acquisition pathway, the following documents help test teams recognize the importance of maintaining schedules and delivering software to the user quickly.

- In accordance with Section 800(a)(2)(B) of Public Law 116-92, the product team will "demonstrate the viability and effectiveness of such capabilities for operational use not later than one year after the date on which funds are first obligated to acquire or develop software."
- DoDI 5000.87, "Operation of the Software Acquisition Pathway," states:
  - "Programs using the software acquisition pathway will demonstrate the viability and effectiveness of capabilities for operational use not later than 1 year after the date on which funds are first obligated to develop the new software capability. New capabilities will be delivered to operations at least annually to iteratively meet requirements." (See Paragraph 1.2.e. of DoDI 5000.87.)
  - "The MVCR for applications programs must be deployed to an operational environment within 1 year after the date on which funds are first obligated to acquire or develop new software capability including appropriate operational test." (See Paragraph 3.3.b.(5) of DoDI 5000.87.)

The 1-year constraint imposed by both Public Law 116-92 and DoDI 5000.87 can be extremely challenging. Programs that cannot meet this challenge may want to consider another acquisition pathway or should at least ensure that the MVCR is scoped properly to enable early delivery of a fielded capability.

Test teams are an integral part of meeting this challenge and will require a paradigm shift in thinking to plan and execute their test strategy within this 1-year constraint. This paradigm shift extends to the provisioning of resources by the program office for the test teams and tests. The following items offer test teams some points to consider in developing a test strategy:

- Organize the Test Teams. The PM should charter the test teams (e.g., T&E WIPT, SW ITT, CyWG, and STAT Working Group) as early in the planning phase as possible. The test teams should include all relevant stakeholders involved with the planning, resourcing, execution, and reporting of T&E as well as other impacted entities (e.g., SE, cybersecurity, safety engineering, the product team, and users).
- Develop the Test Strategy. Ideally, the test teams should collaborate to produce an integrated test strategy that efficiently incorporates all QA equities. The test strategy development team should include appropriate personnel to address both the functional and nonfunctional requirements. The CDT, when satisfied that the strategy is complete

and executable, presents the test strategy to program management. The test strategy should be developed well before the execution stage.

- Support Development of an Integrated Master Schedule (IMS). The test strategy presented to program management must be sufficiently detailed to support development of a resource-loaded IMS. This IMS will look much like a traditional IMS from project initiation through delivery of the MVCR, though with different milestones. From that point on, the IMS should be based on the anticipated rhythm of epics; an estimated project duration; and planned staffing levels during iterative development.
    - High-level milestones on the product roadmap, introduced in Section 4.2.3, are a summary of these IMS activities.
    - If a trained program scheduler is not available, the CDT should reference the guidelines in the Agile and Earned Value Management: A Program Manager's Desk Guide to develop and track progress against the T&E schedule.

4.3 Test Automation Tool Selection and Training

DevSecOps seeks to compress the time to delivery without sacrificing quality. Test tools facilitate this schedule compression by automating the process and removing sources of manual error.

As previously addressed in Section 1.3, a foundational strength of a DevSecOps pipeline is the set of automated and integrated tools that allow for rapid development, testing, cyber T&E, and deployment processes. Automated test tools in this pipeline will be a significant source of data to support the overall government evaluation of the system, thus supporting an Agile, iterative, data-driven framework such as dTEaaC. Government test teams and cyber test teams must be fully trained in these tools to understand and have confidence in the efficacy of test planning, execution, and results that they produce. Cyber test teams should be able to write and understand test scripts. Government test teams, part of the T&E WIPT, should participate in the various planning activities with developers and other key stakeholders when decisions are being made to purchase and integrate automated test tools into the DevSecOps pipeline. Again, formal training, arranged by the PM, will be necessary, and this training should be funded and scheduled early in the program to prevent future delays. It is critical that test teams are familiar not only with the test management and test execution tools but also with the various details necessary to verify and validate test results. Various test tool training is included in available STAT COE courses (see course information in Section 6.1.3) and DAU DevSecOps courses.

4.3.1 Tool Selection Criteria

As tools are being acquired to support test planning, management, and execution, the following *characteristics are critical to good tool selection*.

Trust and Transparency

To enable testing at the speed of relevance desired in DevSecOps, the testing tool ecosystem detailed in Section 3.1 must be both trusted and transparent; the tool, process(es), and data.

At a high level, trust and transparency can be measured by how effectively a given process or tool is understood, operated, and evaluated by all stakeholders. Trust is enhanced by using proven tools or conducting verification, validation, and accreditation.

For automated tests executed in a CI/CD environment, transparency means that all stakeholders should be able to easily view and verify test results by examining the test scripts as well as the output (data, test history) of an automated test execution tool in the CI/CD pipeline. Transparency is enhanced by mutual understanding of how results will be used by all parties to improve the process and product.

Data Format Standardization

For tests results to be properly verified or validated, the output data, their formats, and storage should be in accordance with common standards that are well-known and accepted by all stakeholders.

Integration With Other Tools

An attribute of a high-quality tool is its integration with other tools utilized throughout the pipeline. These tools will often document and utilize common APIs and data formats that are easily queried and ingested by other tools, often requiring little effort to integrate into existing pipelines.

Cyber Risk Capabilities

Although CI/CD brings many benefits, it also increases an organization’s attack surface. People, processes, and technology are all required for CI/CD, and all can be avenues of attack. The specific methods attackers use to exploit CI/CD environments are diverse; however, certain risks are more prominent than others. Understanding the most prominent risks to CI/CD environments can help organizations allocate security resources more efficiently. Obtaining tools with these capabilities to offset the security risks should be a top priority. The OWASP CI/CD Security Cheat Sheet Website discusses the security capabilities needed for good tool selection.

4.3.2 Government Software Factory Considerations

When developing software in an established software factory, the test automation tools and many other tools in a DevSecOps pipeline will likely already be selected and established. In these instances, training selection will be straightforward, as it should be based on the current tooling. Regardless, even established DevSecOps pipelines will undergo changes and updates that will require collaboration and integration with the stakeholders in the testing community. The CDT and cyber test team should consider whether the security testing automated tools are appropriate or if the software development requires additional or alternate tools. Some considerations for security automation include development language, application type, compilers, effectiveness, false positive rates, ease of deployment, compatibility, integration, scalability, UI, playback, data capture, and cost.

4.3.3 Digital Engineering and Model-Based Systems Engineering Test Considerations

- Leveraging DE and MBSE to capture the test cases improves the test threads and verification of the requirements.
- MBSE methods using validated model-based T&E may reduce the need for comprehensive T&E including manual cyber T&E.

DoDI 5000.97 specifies that DoD will iteratively develop a DE capability that enables DevSecOps, automated testing, and software deployment throughout the system life cycle. This capability will be accessible to DT&E personnel and will support DT&E activities related to verification and validation of requirements, cybersecurity, and software distribution as well as deployment to the operational environment.

DT&E personnel should collaborate with the development team and with PMO DE activities to define the digital threads—data, models, software, and processes—needed to support the automation of as many DT&E and cybersecurity functions as possible. These digital threads should be documented in the IDSK (discussed in Section 4.2.2) as they are identified, linked to specific decision support functions.

DE and MBSE initiatives are growing throughout DoD, and the test community will require training in the full spectrum of M&S products that may be utilized in conjunction with DevSecOps pipelines.

4.4 Understanding the Cadence and Roadmap

The software development cadence is a fixed, repetitive sequence of sprints and releases that the program adopts. Once the cadence is established, all test teams, DT&E and OT&E, must determine how they will fit into the schedule while still conducting robust testing. Some programs have embedded government test teams full-time in the development teams. In the event of resource limitations, the CDT must determine whether sufficient government resources exist or whether the government test teams should focus more on the release integration and acceptance testing.

Development team decisions on methodology influence the cadence and roadmap. For example, if a development team uses test-driven development, it generates test cases early in the SDLC, affecting the government test teams' test case development and review activities. Some test cases derive from user stories, so their development roadmap depends on the user story creation roadmap. Some test cases assume realistic data as inputs, so their development roadmap depends on obtaining or generating that data. These and other considerations influence the project cadence, particularly regarding how the development team interacts with the SW ITT.

During the execution phase of a program on the software acquisition pathway (see the DAU AAF Website), the roadmap becomes the primary schedule guide. The schedule of the test events is fixed by the cadence. The content of the test events is established by the roadmap. To help maintain the schedule, the CDT should do three things: strive to collect as much valid test data from the sprint demos as possible; understand the contractor automated test scripts for potential reuse during release integration and acceptance testing; and ensure the scope of test activities is commensurate with the content of the release.

4.4.1 Software Testing Stages

Software DT&E typically follows a structured order to ensure comprehensive T&E of software products. Although this sequence generally represents the order of testing, the testing stages often overlap and may not strictly follow this sequence in Agile or iterative development methodologies where testing is integrated into the development process.

The order of software testing may vary slightly depending on the specific development methodology, but generally the stages include:

1. Unit Testing. The first stage is where individual components or units of the software are tested in isolation. Developers typically perform these tests to check whether each unit works as intended.

2. Integration Testing. After unit testing, the units/modules are combined and tested together to ensure they function correctly when integrated. This stage focuses on interactions between different modules.
3. System Testing. This stage involves testing the entire system to verify that it meets the specified requirements. It includes functional and nonfunctional testing to validate the system's behavior under different conditions.
4. Acceptance Testing. Also known as UAT, this stage involves testing the software with end users to ensure that it meets their needs and works in the intended environment.
5. Regression Testing. Whenever changes are made to the software, regression testing is performed to ensure that the new code has not adversely affected the existing functionalities.
6. Performance Testing. This stage involves testing the software's performance under various conditions to assess its responsiveness, stability, scalability, and reliability.
7. Security or Software Assurance Testing. Automated security testing involves identifying vulnerabilities in the software. The evaluation and prioritization of those identified vulnerabilities to understand mission impacts are not usually automated. If the vulnerabilities are discovered during early unit testing, the developers may be able to fix the vulnerabilities without a rigorous evaluation. This testing is integrated into the pipeline by the test teams.
8. Cyber DT&E Testing. Hands-on security testing, or cyber DT&E, involves penetration testing to assess exploitability and the performance, usability, and maintenance impacts of identified vulnerabilities, and verifying the functional cyber performance requirements of the integrated software into the larger subsystem or system. This testing is not directly integrated in the pipeline and requires a test environment.
9. Usability Testing. Also known as HSI testing, this phase ensures that the software is user-friendly by assessing factors such as the UI, navigation, and overall UX.
10. Maintenance Testing. Even after deployment, periodic testing is essential to ensure the software's continued functionality and effectiveness. This maintenance activity involves testing patches, updates, and modifications.
11. Documentation and Reporting. Throughout the testing process, documentation is crucial to record test cases, results, and any issues found. Reports are created to summarize the testing process and outcomes.

4.4.2 Test Execution

Test execution begins at the first sprint demonstration. Automation is necessary for efficiency and repeatability. All test data are potentially useful, and the government test teams should try to avoid repeating tests that have already verified functionality at the sprint level (unless it is regression testing). Government DT&E teams maximize their impact by conducting integration testing to verify that inputs from multiple development teams work together to satisfy mission requirements through end-to-end mission thread testing; government test teams should plan to include end users in this testing. DT&E teams conducts cyber penetration testing in the developmental, test, or pre-production/staging environments. DT&E also addresses cyber testing gaps identified in automated testing.

Software assurance testing, a component of security testing, is specifically focused on investigation and mitigation of fault/failure cases within the software, where a particular computational fault could result in a mission failure (or mission-essential function/subfunction failure). Software assurance testing is often done by planning negative testing—the intentional stimulation of abuse/misuse cases to determine the effects within the SUT.

The necessary test cases for positive testing (everything works like it should) increase to the square of the number of functional lines of instructions, at a minimum. Therefore, the negative test cases (proving the nonexistence of a fault case) must grow even faster because each functional case may have more than one negative case to prove no impact when placed in an abuse/misuse configuration or input set. To borrow a car metaphor, if the lug nuts are redesigned, the radiator does not need a new series of pressure tests because it can be reasonably asserted that those subsystems are relatively independent. Programs must have an equivalent (and validated) understanding of the interaction space between software functional modules.

Without a way to refactor the test cases and remove those that are not relevant for the code that was recently merged in, programs are forced to rerun every test case, every update cycle—even those that are not relevant. The rerun of every test case, every update cycle increases demand on testing time and computing resources. Software programs should provide to the test team (and the test team should help them formulate) software models that define the interaction space between components, modules, and/or microservices/API calls. Test teams should have available the information they need to make test case refactoring decisions and determine which test cases are relevant to run against a newly proposed code or module.

4.4.3 Test Results Evaluation

In software DT&E, test results evaluation involves assessing the outcomes of various testing stages to determine the software's quality; identify issues; and make informed decisions regarding its readiness for deployment. Test results evaluation includes the analysis of security and cyber DT&E tests. Test results evaluation not only identifies issues but also drives improvements in the software development and testing processes, aiming to enhance the quality and reliability of the software product.

Test results evaluation includes:

1. Review of Test Results. The first step in test results evaluation involves reviewing the results obtained from different testing phases, including unit tests, integration tests, system tests, and more. Test results are analyzed to understand the software's behavior, its performance, and any identified issues or bugs.
2. Defect Analysis. Identified defects or issues are analyzed to understand their severity, impact on the software, and potential causes. This analysis helps prioritize which issues need immediate attention and which can be addressed later.
3. Root Cause Analysis. Understanding the root causes of defects is crucial and involves delving deeper into why particular issues occurred. This analysis helps prevent similar problems in the future and improves overall software quality.
4. Severity and Priority Assessment. Each identified issue is assessed based on its severity (how critical it is) and priority (how urgently it needs to be fixed). This assessment aids in determining which issues should be resolved first.
5. Decision Making. Based on the evaluation of test results, including the severity and priority of issues found, decisions are made regarding the software's readiness for the next phase, such as deployment or further development and testing cycles.
6. Reporting. Comprehensive reports are generated to document the test results, including the identified issues, their severity, the steps to reproduce them (in DevSecOps, usually expressed in a test script), and any other relevant details. These reports serve as references for developers, stakeholders, and decision makers.
7. Feedback Loop. The evaluation process often includes a feedback loop, where insights gained from test result evaluations are used to improve future testing strategies, development practices, and overall software quality. In DevSecOps, improvement often comes in the form of increased automation.
8. Validation of Fixes. Once issues are addressed, a reevaluation might occur to validate that the fixes have effectively resolved the identified problems without introducing new issues.

9. Continuous Improvement. Continuous improvement is a key aspect of test results evaluation. It involves refining testing processes, enhancing test cases, and implementing best practices based on the lessons learned during evaluation.

4.4.4 Test Results Reporting Considerations

Test results reporting should keep pace with the cadence of the test execution in the pipeline. A recommended approach is to write the report concurrently with test execution and minimize the need for high-level staffing of the report. Within the Agile framework, decision making is allocated to the lowest possible level, so test reporting can be tailored to the needs of those decision makers. A project should use tools to automatically develop the report and provide interim reports throughout the test execution period.

4.5 Agile Framework and Software DT&E Integration

In Agile methodologies, DT&E activities are integrated into the development process to ensure that testing and evaluation occur iteratively and concurrently with development. This iterative approach allows for continuous improvement, early issue detection, and faster adaptation to changes, ultimately leading to the delivery of higher-quality software.

The following subsections describe the software development activities that DT&E teams should participate.

4.5.1 Sprint/Iteration Planning

Sprint/iteration planning includes:

- User Story Refinement: Collaboratively refining user stories to ensure clear acceptance criteria, including testing requirements.
- Task Estimation: Estimating testing efforts and defining testing tasks for each user story.
- Backlog Integration: Selecting the proper set of product backlog items to include during the sprint.
- Threat Modeling: Defining the scope of the threat model and identifying cyber T&E requirements using MBCRAs (preferred) or some other type of assessment.

4.5.2 Development Phase

The development phase includes:

- Test-Driven Development: Writing tests before writing code to increase the testability and quality of the code.
- Continuous Integration: Continuously integrating code changes into the main branch, triggering automated builds and tests.
- Unit Testing: Writing and running unit tests to ensure individual components/modules function as expected.

4.5.3 Testing Phase

The testing phase includes:

- Acceptance Testing: Conducting tests against user stories to validate whether they meet acceptance criteria.
- Regression Testing: Ensuring new code changes do not adversely affect existing functionalities.
- Exploratory Testing: Conducting ad hoc testing to discover unforeseen issues or edge cases.

4.5.4 Evaluation and Review

Evaluation and review activities include:

- Sprint Review/Demo: Demonstrating completed user stories to stakeholders for feedback and validation.
- Retrospective: Reflecting on the sprint, including testing and evaluation aspects, to identify improvements for the next sprint.

4.5.5 Automation and Tools

Automation and tools include:

- Automated Testing: Implementing automated tests for regression, integration, and functional testing to speed up testing processes.
- Process Automation: Automatically distributing test results to all interested parties. Test failure automatically inhibits progression to subsequent phases.

4.5.6 Collaboration and Communication

Collaboration and communication activities include:

- Cross-Functional Collaboration: Ensuring close collaboration between development, testing, and other stakeholders throughout the sprint.
- Daily Stand-ups: Conducting daily meetings to discuss progress, issues, and impediments, including testing-related concerns.

4.5.7 Continuous Improvement

Continuous improvement includes:

- Feedback Loop: Utilizing feedback from sprint reviews and retrospectives to continuously improve testing strategies and practices.
- Adaptation and Flexibility: Embracing changes and adapting testing approaches according to evolving project requirements.

4.5.8 Acceptance Criteria Test Case Example

Table 4-2 (which will be used in the DAU Agile Test and Evaluation course that is in development) illustrates how the developer and test teams can decompose a feature user story to scenarios for test cases in test design. The scenarios express different conditions. The "Given" and "When" statements describe factors that can vary during testing. In these two scenarios, only the account balance is varied. However, tests could be developed for cases where the card is not valid, or the machine does not have enough money. The "Then" statements describe the desired response variable. If the response variables are not achieved, the test fails. The test teams can use these scenarios to support test-driven development and refine the definition of the overall user story.

Table 4-2. Acceptance Criteria Test Case Example

Acceptance Criteria Test Case Example	
Feature: Account holder withdraws cash **As an** Account Holder **I want to** withdraw cash from an ATM **So that I** can get money when the bank is closed	
Scenario: Account has sufficient funds	**Scenario**: Account has insufficient funds
Given the account balance is $100 **And** the card is valid **And** the machine contains enough money	**Given** the account balance is $10 **And** the card is valid **And** the machine contains enough money

Acceptance Criteria Test Case Example	
When the Account Holder requests $20	**When** the Account Holder requests $20
Then the ATM should dispense $20 **And** the account balance should be $80 **And** the card should be returned	**Then** the ATM should not dispense any money **And** the ATM should display there are not sufficient funds **And** the account balance should be $10 **And** the card should be returned

4.6 DevSecOps Architecture

4.6.1 DevSecOps Architecture Security Features

The software developer and program should use DevSecOps architectures to help manage the continuous software development risks associated with cyberspace threat actors. Cyberspace threats target developer environments and tools; infect release pipelines with malicious scripts; and gain access to production data via test environments to establish persistent presence. This section provides best practices, as defined in the DoD Enterprise DevSecOps Reference Design, for implementing DevSecOps. These practices include implementing (1) specific architecture and design features to enhance security, including using microservices; (2) a service mesh; and (3) the security sidecar pattern of deploying components of the application into a separate process or container to provide isolation and encapsulation. Programs should determine whether implementation or adoption of these practices is appropriate to meet the assurance level of the program. Programs should remediate any residual risk through custom implementation or application of additional assurance methods and practices.

Established gates within the software development pipeline and continuous monitoring enable programs to monitor risk thresholds for assurance methods and practices applied throughout the life cycle of the software application.

4.6.2 Software Development Platform Security Considerations and Zero Trust

Baseline Security Considerations

Planning for security in the development environment in DevSecOps is crucial for protecting code and data. Best practices to establish baseline security protections for development environments include the following:

- Understand the Risks. The Development team should be aware of the potential risks associated with their development environment, including understanding common vulnerabilities, attack vectors, and security threats.
- Implement Technical Controls. The Development team should apply technical controls to mitigate risks. These controls include:
  - Access Controls: Limit access to authorized personnel only. Use strong authentication mechanisms and enforce the principle of least privilege.
  - Network Segmentation: Isolate development environments from production systems to prevent lateral movement in case of a breach.
  - Encryption: Encrypt sensitive data at rest and in transit.
  - Code Scanning Tools: Use static code analysis tools to identify security vulnerabilities early in the development process.
  - Secure Coding Practices: Train developers on secure coding practices to avoid common pitfalls. Ensure the contract enables verification of these practices.
  - Trust and Verify Methods. Regularly review logs, monitor access, and perform security assessments.

Zero Trust Principles

In addition to the above considerations, the software developer (government or contractor) should implement the principles of zero trust throughout the entire DevSecOps platform and all environments. Zero trust principles include the following:

- Verify Explicitly: Always authenticate and authorize. When verifying explicitly, examine all pertinent aspects of access requests. Instead of assuming trust based on weak assurances (such as network location), consider multiple data points.
- Use Least-Privilege Access. Grant users or processes only the minimum necessary access or permissions required to perform their intended tasks.
- Assume Breach. Operate under the assumption that a breach could occur. This mindset helps minimize the impact of potential security incidents.
- Verify End-to-End Encryption and Use Analytics. Verify end-to-end encryption and use analytics to gain visibility, detect threats, and improve defenses.

Implementing zero trust principles and securing code and data in the DevSecOps environment will assist in preventing hackers from compromising developer environments; infecting release pipelines with malicious scripts; and gaining access to production data via test environments.

Assessment of Effectiveness

The development environment in DevSecOps is a target for cyberspace threats to modify code and inject malware. The above considerations are best practices for reducing the cyber risks to development environments. Programs should also strive to understand the effectiveness of the practices:

- Use MBCRAs and follow-on penetration testing of the software development platform's people, processes, tools, networks, environments, defensive capabilities, and architecture to verify effectiveness.
- Exercise incident response and continuity of operations plans during the MBCRA or conduct a simulation of attacks against the platform.

4.6.3 Infrastructure as Code

IaC provides confidence that the deployment across all application environments is consistent and under strict configuration management. IaC scripts should include secure configuration across all environments as well as the provisioning of roles and privileges. Test teams should assess the exploitability of session tokens, software keys, and automated scripts. Test teams should also assess the effects or detection of attempts to alter data collection, logging, and telemetry requirements in the configuration on deployment.

4.6.4 Security as Code

Test teams should verify all security elements managed by SaC and verify proper configuration management. The government cyber testers should identify how the contractor will make changes to code and infrastructure and where they are adding security checks, tests, and gates. Test teams should verify that the security testing and scans implemented into the pipeline enable automatic and continuous detection of vulnerabilities and security bugs. Government test teams should validate that developers codify access policy decisions into source code and use test results to remediate findings during subsequent coding. The cyber test teams should validate that SaC:

- Automates security scans and tests (such as static analysis, dynamic analysis, and penetration testing) within the pipeline.
- Refines SAST/DAST rules and templates based on feedback from previous tests.
- Builds in continuous feedback loops providing results to developers for remediation while coding.
- Validates the means to evaluate and monitor automated security policies.

- Automates manual tests via custom scripts, with human sign-off on results if necessary.
- Validates the accuracy and efficiency of test scripts.
- Tests new code within a staging environment to allow for thorough, manual, and failure tests on every code commit.
- Automates the creation of logs (or noted anomalies) within a review dashboard to enable continuous monitoring.

By implementing zero trust principles, IaC, SaC, and the other considerations discussed in this section, the software developer helps to reduce the risks associated with cyberspace threats.

4.7 Cyber and Software Assurance Test and Evaluation

Software vulnerabilities are inevitable without significant investment in formal methods. Software developers, testers, and program office stakeholders should apply a multipronged approach to reduce vulnerabilities and maximize the resilience of a software-enabled system and its components. This approach should include managing the cyberspace risks to DevSecOps architecture and software development as well as continuous automated security and software assurance testing in the pipelines; analysis and evaluation of the automated results; and hands-on or manual cyber DT&E.

Program offices adopting DevSecOps processes to develop and deliver applications should incorporate cyber developmental testing activities described in the forthcoming Version 3.0 of the DoD Cyber DT&E Guidebook. Cyber DT&E includes automated security, the evaluation of scan results, and manual testing, and it informs software assurance determinations. Automation alone is not sufficient to find critical vulnerabilities. The flexibility and creativity of the human brain enables manual testers to find clever vulnerabilities and attacks that automation may miss. It takes human intelligence to instinctively sense when to dig deeper and when to move on. Manual, hands-on testing enables additional focus on the performance aspects of the delivered application in the intended operating environment.

The DevSecOps ecosystem provides a platform, an infrastructure, and capabilities to reduce the burden of full-stack software security and testing. Although trust is maintained through security analysis and the hardening of platform services, test teams must verify, through automated and manual assessment methods, that all platform, infrastructure, and application security requirements are implemented by the development team or inherited by supporting services.

The CDT and the developmental and operational test teams including the security champion have key roles within the CyWG to guide and participate with DevSecOps teams to build security into their pipelines and Agile processes. These roles include:

- Understanding the security expectations of the authorizing official at the start of development.
- Assessing the effectiveness of the developers' and their subcontractors' security processes, procedures, and technology during development.
- Soliciting timely and relevant security information to effectively assess program risk.
- Participating in the initial MBCRA and updating the MBCRA with new information as needed for each increment.
- Performing out-of-band monitoring of software inventory lists and the third-party software bill of materials (SBOM) as well as the manifests for direct and transitive (component of components) code and containers, especially if the application is invoking microservices at need time.
- Ensuring automated monitoring of the development, test, and production environments and tools using up-to-date threat intelligence feeds.
- Enabling continuous software composition analysis (SCA) to inventory and assess known vulnerabilities in hardened containers, software packages, libraries, and open-source software.

4.7.1 Continuous Security Practices

The government developmental and operational test teams should be involved in ensuring that the software factory incorporates the following security/assurance methods continuously throughout the testing activities and across the phases of DevSecOps. Section 5 discusses these topics in more detail.

- Mission-Based Cyber Risk Assessments. MBCRAs are initially performed before DevSecOps iterations commence and are reconsidered and updated as required in any phase. MBCRAs are performed mostly during the plan phase in conjunction with threat modeling.
- Continuous Monitoring. Continuous monitoring is performed in every phase as a critical feeder of information to the cyber test teams.
- Threat Modeling as Code. To evaluate threats continuously along with the development process, threat models must be maintained and updated in synchronous code with every development at each sprint.
- Application Programming Interface Security Tests. API security tests are conducted primarily during the test and deliver/deploy phases.

- Security Verification. Security verification is automated to the maximum extent possible with every build and test, at any stage, and includes:
    - SAST
    - DAST
    - Interactive application security testing (IAST)
- Cooperative and Adversarial Tests. Manual (hands-on) cyber testing is conducted primarily during the test/release, deliver, and deploy phases.
- Persistent Cyber Operations Tests. As outlined in the DOT&E FY 2021 Annual Report, persistent cyber operations provide cyber Red Teams longer dwell time to probe deeper into network and system vulnerabilities. This approach results in assessments that are more thorough and more threat representative.
- Review or Selection of Security Testing Methods. The review or selection of security testing methods should be informed by systems security engineering methods and practices employed in accordance with the DoD Technology and Program Protection Guidebook. To mitigate risks commensurate with technology, program, system, and mission objectives, SAST, DAST, and IAST tool selection should consider software language, software standards, access to source code, and common weaknesses based on system characteristics. Tool selection may require procuring multiple tools to analyze a single software build or establishing multiple pipelines to accommodate different languages or configurations.

4.7.2 Software Assurance Testing/Software Composition Analysis

Software assurance testing (post-build) ensures that security is included at all levels of testing. The automation of test suite execution enables comprehensive testing with full coverage of software execution paths, providing test teams with assurance that the software functions only as intended. The test teams verify that the SaC approach using SAST, DAST, IAST, and fuzz testing is functioning correctly. The test teams are responsible for helping to prioritize the findings and for ensuring that developers are responding to the feedback. The PMO and government lead test organizations should ensure software assurance testing includes:

- Prior SAST and secure code reviews; evaluating the code structure before compiling enables developers to fix errors before testing.
- Negative testing at the unit level to exercise negative inputs at the lowest functional level.

- Abuse cases (McDermott and Fox 1999) and associated acceptance criteria to inform the creation of integration tests that can be executed along with automated test suites on a set cadence and before delivery.
- Automated penetration testing:
  - DAST: Identifies runtime vulnerabilities before deploying to operations as a form of automated black box testing (post-compiling). DAST can identify runtime vulnerabilities and environment-related issues. DAST provides the security teams with external attack vectors.
  - IAST: Finds vulnerabilities that appear when interacting with the application. IAST identifies the line in the source code with weaknesses post-build. IAST is most effective when combined with functional testing.
- Setting control points in the CI/CD orchestrator for the review of test results.
- Cooperative manual penetration testing that focuses on verifying DAST/IAST findings and the discovery of other vulnerabilities.
- Adversarial manual testing conducted by cyber test teams to evaluate the security of the system (software, integrated software, etc.) by attempting to gain access and exploit built-in features. Teams evaluate the mission impacts of unmitigated vulnerabilities and search for other unknown weaknesses.

Automated Penetration Testing

DAST and IAST can be time-consuming but result in fewer false positives than SAST. The contractor and PMO should ensure the flexibility and customization of scan times and CI/CD integration to optimize the use of DAST and IAST. Automated penetration testing provides the security test teams with insights into functionality to target during manual testing.

Penetration Testing as a Service

Penetration testing as a service (PTaaS) is an option to save time where automated vulnerability scanners as well as manual cooperative test teams scan and test the system at scheduled and/or unscheduled intervals. If the development team uses PTaaS, the government cyber test teams should receive the reports and monitor the metrics associated with the PTaaS efforts. PTaaS can be a useful precursor to government cooperative and adversarial testing.

Software Composition Analysis

In addition to vulnerability management as required by DoDI 8531.01, “DoD Vulnerability Management,” test teams should ensure continuous assessment for known vulnerabilities within commercial off-the-shelf/open-source code used in environments, tools, and software. IaaS, PaaS, and SaaS used by the program should provide evidence that software is free from known vulnerabilities and should have a defined process for patching. The test teams should use SCA tools and continuous out-of-band monitoring throughout development to assess known vulnerabilities in hardened containers, software packages, libraries, and open-source software, enabling rapid mitigation and remediation response.

4.7.3 Software Assurance for Continuous Integration/Continuous Delivery

Once a program has established an Agile process and identified software assurance engineering methods, CI/CD supports the automated generation of engineering artifacts for test teams to verify that the applied assurance practices are maintaining the appropriate assurance level for the system. Where possible, test teams should verify that the CI/CD pipeline integrates custom software assurance activities and tools, with automated execution of tests as appropriate to support verification and feedback to the developer.

Test teams should use data produced by the automated execution of assurance tools to generate findings and artifacts to inform development and the test teams. Test teams should identify how automation is used to correlate, score, and prioritize tool findings to verify the adequacy of generated artifacts and reports. Utilization of enterprise CI/CD SaaS provides built-in security through the inclusion of assurance capabilities. Built-in capabilities are compared with system requirements to determine whether additional assurance practices are needed and whether automated correlation, scoring, and prioritization are effective to determine whether the system assurance level is maintained.

4.7.4 DevSecOps Cybersecurity Data and Reporting

An underlying goal of DevSecOps is preventing vulnerable applications from reaching production. The PM, information systems security manager, DT&E team, security champion, OT&E team, and Development team need to be able to make an informed decision at the end of each sprint to recommend a release go/no-go decision, based on confidence that the application is resilient to adversary exploits. This decision is risk-informed, as no application can be 100 percent resilient. To make these risk-based decisions, the stakeholders require timely security data of various types and collected in specific ways. Automated testing produces some of the required data. Manual testing and other nonautomated methods produce more of the required data, especially when performed in conjunction with instrumented systems and on instrumented platforms. All this data should be available to all stakeholders at any point in time. Test teams should collaborate with the test lead to define the evaluation data needed by each stakeholder.

4.7.5 Cybersecurity Deliver, Deploy, Operate and Monitor Considerations

Cybersecurity needs to be baked in during software development to promote cybersecurity during operation. The development team, security champion, and SW ITT, including representatives from the developmental and operational testing communities, will work together to determine how to deliver a software system securely; how to determine that the system is safe to deploy; how to deploy it; how to operate it securely; how to monitor its operation; and how to provide feedback. These activities occur outside the software factory on which a system is developed, but they draw on knowledge gained during development, especially how to automate tasks related to verifying security and how to document processes that cannot be automated.

4.8 Testing Systems that Incorporate Artificial Intelligence and Machine Learning

Agile and DevSecOps iterations emphasize developing and testing code. Many modern systems incorporate AI, particularly machine learning (ML). The life cycle for developing an ML-based system necessarily differs from a traditional system in that an ML-based system requires training data and cannot be properly tested until it has been trained. The amount of training data can be orders of magnitude larger than the system trained on that data. A DevSecOps process for developing a system that includes an ML component must consider some issues related to testing that are nonexistent, or at least significantly less significant, in a traditional system. These issues include:

- Verifying and validating training data.
- Testing the software environment used to conduct training.
- Testing system components that implement ML. The functional requirements for these components have little to do with the system's functional requirements. They concern matters such as whether a neural network is properly connected, as opposed to what capabilities the trained neural network delivers to a user.
- Defining statistically based tests. Testing does not involve right or wrong answers so much as whether the percentage of time a system delivers a satisfactory answer is above some threshold.

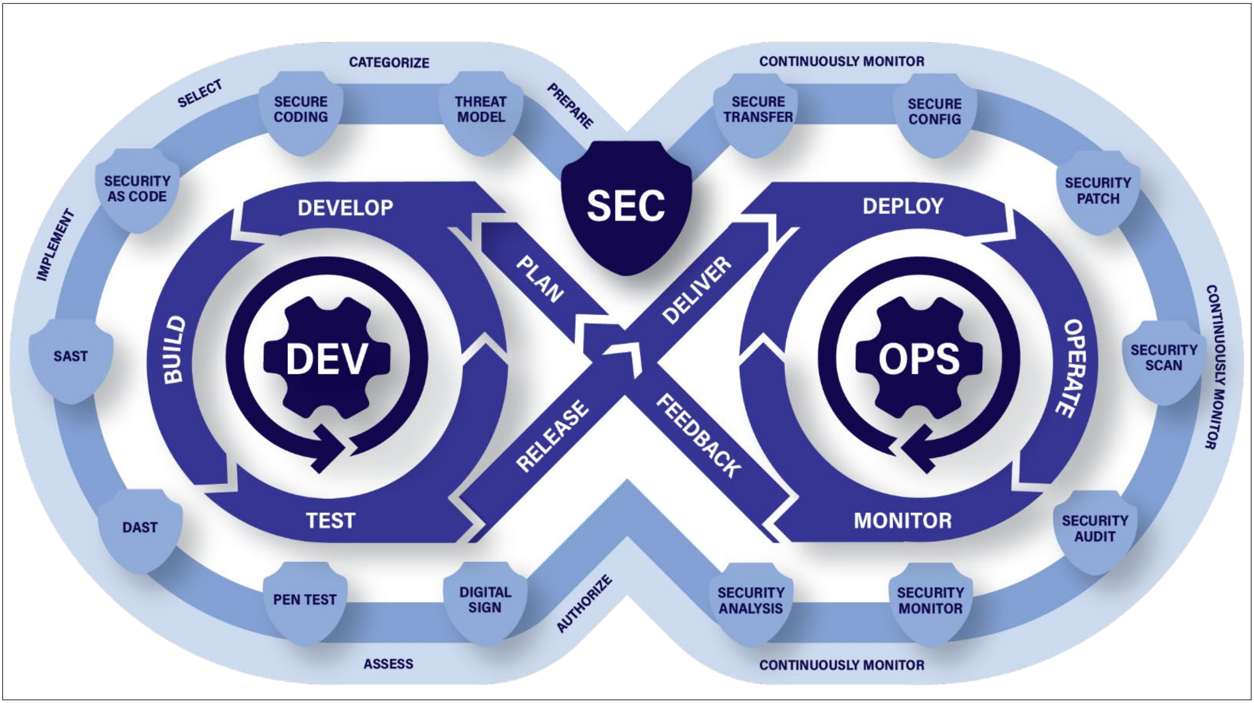
Conversely, there has been research on AI-enabled software testing, in which AI is used to generate test cases, execute test cases, and evaluate results. This is a nascent field, and its use within DevSecOps is unclear.

An in-depth discussion of these topics is beyond the scope of this guidebook. DTE&A staff is preparing a guidebook on AI T&E which is scheduled to be published in FY 2025.

5 DT&E in DevSecOps – DT&E Activity Considerations within the DevSecOps Phases

5.1 Introduction

T&E considerations during DevSecOps phases revolve around integrating robust T&E practices with a strong focus on security across the entire SDLC. The T&E activities supporting each DevSecOps phase illustrated in Figure 5-1 are delineated in Table 5-1 and the following subsections.



Source: DoD Enterprise DevSecOps Fundamentals, Version 2.5, p. 20

Figure 5-1. DevSecOps Distinct Life Cycle Phases

DevSecOps software development does not reduce the critical need for mission-based/end-to-end testing. Especially for tightly coupled SoS networks, communication, coordination, and empirical demonstration and verification are essential to avoid the introduction of mission-impeding discrepancies.

DevSecOps government software development test responsibility hinges on an appropriate level of independence, effective test execution, and the adequacy of reporting. Working closely with the program office, the test team must, at a minimum, determine the following:

- Traceability of requirements/user stories to acceptance criteria and test cases.
- Test capability needs and gaps.

- Critical areas to test.
- Adequacy and coverage of planned testing.
- Opportunities for developmental and operational test integration.
- Level of contractor testing.
- Test frequency.
- Test reporting methods/adequacy.
- Level of automation.

In addition to the major milestones defined in various DoD pathway issuances, Table 5-1 identifies control-gate-based decisions enabling movement from one DevSecOps phase to the next and the notional supporting test activities that feed those decisions. Note that decision points within the Agile/DevSecOps construct occur more frequently than in traditional acquisition programs. Table 5-1 addresses DT&E objectives; there is ongoing DOT&E development of comparable software OT&E guidance to be captured as a companion guide to the DoDM 5000.96.

Table 5-1. DevSecOps Test Activities, Objectives, and Control Gate Decisions

Phase	Test Activities/Data	Objective	Affirmative Control Gate Decision
Plan	Definition of Done	Ensure the definition of done captures the evaluation of DT&E objectives and user acceptance criteria	Ready to code in development environment?
	Software Test Plans	Create test plans to capture DT&E objectives (integrated test plans are preferred)	
	Threat Modeling	Identify threats to the system	
	Mission-Based Cyber Risk Assessment (MBCRA)	Determine cyber T&E priorities based on mission risk	
Develop/Build	Unit Testing	Capture code-level functionality; trace to higher-level requirements	Ready to test in test environment?
	Static Application Security Testing (SAST)	Ensure source code quality	
	Functional Testing	Capture functionality at the system level; trace	

Phase	Test Activities/Data	Objective	Affirmative Control Gate Decision
		to higher-level requirements	
	Regression Testing	Ensure integrity of previously completed capabilities	
	User Story Demonstrations	Capture and provide feedback	
	Software Composition Analysis	Identify open-source vulnerabilities and license issues	
	Software Bill of Materials (SBOM) Monitoring	Capture software versions to identify vulnerabilities	
	Compliance Verification	Ensure compliance standards are met	
Test/Release	Static Application Security Testing (SAST)	Ensure source code quality	Ready to release to pre-production environment (operations team)?
	System Integration Testing	Ensure code segments combine to provide capability	
	Regression Testing	Ensure integrity of previously completed capabilities	
	Interoperability Testing	Verify the compatibility and communication of different systems, applications, or devices and ensure the data exchange and functionality are consistent and reliable across various platforms and environments	
	Dynamic Application Security Testing (DAST)	Evaluate code quality during execution	
	Application Programming Interface (API) Testing	Evaluate software interfaces for functionality and security	
	Interactive Application Security Testing (IAST)	Evaluate application quality through software instrumentation	
	Human Systems Integration (HSI) Testing	Perform initial evaluation of usability	

Phase	Test Activities/Data	Objective	Affirmative Control Gate Decision
	Mission-Oriented DT&E	Ensure user end-to-end capability	
	Cyber DT&E	Identify vulnerabilities for future mitigation	
	Penetration Testing	Identify exploitable vulnerabilities	
Deliver	Regression Testing	Ensure integrity of previously completed capabilities	Ready to release to production environment?
	Interoperability Testing	Verify the compatibility and communication of different systems, applications, or devices and ensure the data exchange and functionality are consistent and reliable across various platforms and environments	
	Performance Testing	Gather system-level usage metrics	
Deploy	OT&E (Initial Operational Test and Evaluation / Follow-on Operational Test and Evaluation)	Determine operational effectiveness, suitability, survivability, and lethality (as applicable)	Ready to deploy to users?
	Interoperability Certification	Certify the system for interoperability	
Operate/ Monitor/ Feedback	System Monitoring	Capture operational performance	N/A, sustainment
	Value Assessment	Provide data to the PM and user community	
	Fault Detection	Provide data to the development community	
	Log Auditing	Provide data to the T&E community	

5.2 Plan Phase

5.2.1 Definition of Done

The definition of done is used to determine when a product is ready to be accepted by the product's user community. It does not mean the product is "done" as in "finalized." In DevSecOps, products continually evolve in response to changing user needs. The definition of done identifies criteria that the user communities will use to decide whether to accept a product for their use. These criteria include T&E, so the definition of done is usually written to assume a product increment has passed through the test phase of an iteration. The definition of done therefore establishes criteria for a control gate between the test and release phases.

The definition of done is sometimes written as a checklist. Determining whether an iteration is done is then a matter of seeing whether all items on the checklist are marked as complete.

The definition of done is written mainly by the development team, with input from developmental testers. Developers describe how to determine whether the software test plan (discussed in Section 5.2.2) has been carried out. DT&E team members review developers' work and assess its adequacy, guided by the TES developed by the T&E WIPT. DT&E team members may amend checklist items they consider difficult to interpret unambiguously. Especially for system tests, developmental testers may opt to add their own criteria because developers sometimes focus too narrowly on modules and overlook system-level interactions.

The definition of done is updated during each DevSecOps iteration. Each iteration delivers a new set of capabilities. The definition of done must describe how to test those capabilities as well as how to conduct regression tests of previously delivered capabilities. The SW ITT is involved in each update. SW ITT members review tests and operational results from previous iterations and consider how those results should affect testing in subsequent iterations.

The test strategy guides preparation of the definition of done, in the sense that an iteration is not done until the testing strategy has been adequately achieved. Furthermore, the definition of done is tied to preparing the software test plan (see Section 5.2.2), which will describe actions that can be included as checklist items.

Objective

From the testers' perspective, the definition of done must ensure that the evaluation of DT&E objectives is captured. Capturing DT&E objectives is simplest if the definition of done is prepared as a checklist, where items in the list relate to whether an objective has met the necessary criteria through testing and evaluation. For each objective, there should be a set of items, each describing a test or evaluation to be performed. If, at the end of the test phase, all items have been checked, then the capabilities for the iteration can be considered to have been vetted, and the release phase can begin. Figure 1-1 illustrates nominal control gates in the DevSecOps life cycle. The gate following the test phase remains closed until all testing-related activities in the definition of done have been completed.

Why It Matters to the DT&E Community

The definition of done is a contract between the test community and the PM. The contract prescribes the actions that, once performed, are considered sufficient to assure the PM that a quality product is ready for delivery, deployment, and operation.

During the writing of this contract, the test community needs to create a comprehensive T&E plan. In DevSecOps, much of the unit and integration testing is performed by the developmental organization. Developers are often responsible for creating unit tests for their modules. System architects work with developers to create integration tests. The DT&E will want to review these tests. The necessity of this review requires visibility into the software factory. The DT&E team will also want to review how the software factory implements automation—whether and how tests are executed, how regression tests are performed, etc. This visibility should occur early. The developer cannot simply deliver software and its tests at the end of the test phase; the subsequent time to review would delay an iteration. The definition of done should include statements of when visibility is needed and how it will be implemented.

The developmental organization defines system tests too, and the test community will review them. The DT&E team, acting as an independent T&E organization, may create its own system tests. The definition of done should include these expected system tests.

Because the definition of done is a contract of sorts, all program stakeholders need to review and approve it. The test community should expect to prepare a report on its contribution to the definition of done. The report should demonstrate that T&E will yield a product that is reliable and functionally useful. If a program has developed user stories, the report should demonstrate that the system provides the mission capabilities needed in each story. If a program is using risk management, the report should describe how testing and evaluation addresses risks; for example, the risk that network latency will lead to system failure can be addressed through performance testing. The report should discuss the items in the definition of done that pertain to data collection or generation.

Table 5-2 describes scope of the Definition of Done DT role by T&E activity.

Table 5-2. Definition of Done - T&E Activities and DT Roles

T&E Activity	Description of Developmental Testing Role
Unit Test	Determine adequacy of: • Developer's unit tests • DevSecOps ecosystem for unit testing
Integration Test	Determine adequacy of: • Developer's integration tests • DevSecOps ecosystem for integration testing
System Test	Determine adequacy of: • Developer's system tests • DevSecOps ecosystem for system testing • System tests
Cyber Test	Determine adequacy of: • Developer's cybersecurity evaluations • Cybersecurity evaluations

Lessons Learned

Information will be added to the guidebook based on stakeholder feedback.

5.2.2 Software Test Plans

Software test plans are documents describing plans to test and evaluate a system, including:

- The types of tests (and evaluations) to be performed.
- How tests will be performed (automated, manual, or a combination).
- The DevSecOps phases during which tests will be performed.
- The reasons that performing the specified tests increases confidence that a system will execute in accordance with its requirements and satisfy mission needs.
- The metrics and associated planned analysis used to evaluate test objectives/ requirements.

Test types can be categorized in several ways. One approach identifies tests based on system architecture and has three categories: unit tests, integration tests, and system tests. The developmental organization is usually responsible for unit tests and integration tests, although government members of the SW ITT will want to review unit and integration tests for integrity and completeness, and integration tests can require specialized testing skills (discussed in Section 5.4.2). The software test plans should include descriptions of when and how these reviews will occur. The T&E WIPT and the SW ITT play a larger role in system tests, sometimes developing their own tests independently of the developmental organization. The software test plan should describe the system tests that both the T&E WIPT and the SW ITT plan to develop.

Tests can also be categorized by DT&E objective. Example objectives include functional tests (including cyber tests), nonfunctional tests, threat-based cyber tests, and operational evaluations. Software test plans must reflect the infrastructure necessary to achieve different objectives. Functional tests can usually be carried out in the development environment. Other tests may require an environment with characteristics closer to the operational environment. Software test plans should describe how to set up these environments in time to perform testing and evaluation.

Methods for generating input data also categorize T&E. Input data can be random; generated according to methodologies such as STAT and/or fuzzing; generated according to domain models; or intuitively generated, such as might be done by a human operator interacting with a system. Software test plans should specify the kinds of input data that will be used, how the data will be obtained, and the identification of automated tools if needed to acquire this data.

Objective

The objective of writing software test plans is to define testing approaches, schedules, and resources that capture DT&E objectives. In other words, a properly written software test plan lays out testing objectives to achieve by the end of an iteration and explains why achieving these objectives will offer convincing evidence that the iteration's planned capabilities have been delivered. The test plan's author(s) must write a document that articulates to the reader that the system will, after passing all tests, behave according to operational expectations within acceptable limits of risk. A software test plan:

- Informs developers and testers when testing activities will occur. For example, in DevSecOps, where unit tests are (usually) designed and implemented during the develop phase and executed during the build phase, the software test plan's objective is to make the development team aware of the need to allocate resources for unit test development and execution before the test phase.
- Assigns responsibilities for designing and implementing tests. The software test plan should clarify responsibilities for the development team and the SW ITT during all phases of a DevSecOps iteration.
- Assures the PM that carrying out the specified tests achieves DT&E objectives.

The DT&E objectives emphasize the capabilities delivered by a DevSecOps iteration. The objectives therefore change from iteration to iteration, and the software test plans must be updated accordingly. The updates do not usually replace existing tests with new ones. Existing tests become regression tests to verify that new capabilities do not alter existing behavior. One test plan objective is to identify clearly what is new and what already exists for an iteration.

Test planning also entails describing how to perform tests. A test plan should cover the infrastructure needed to execute tests. This infrastructure exists in part within a software factory. A test plan must describe how testing infrastructure integrates into a software factory. A test plan must also state who is responsible for implementing the infrastructure. The part of the infrastructure implementing pipelines usually requires some custom development. The software test plan should treat this development as a small software project and describe expected manpower, resources, and schedules. The parties responsible for the infrastructure's implementation then have the objective of meeting the schedules. The description of the infrastructure depends on the kind of testing occurring. Infrastructure for most unit tests is simple: the test runs in a container, perhaps simulating some input/output devices. Infrastructure grows in complexity during integration testing. Eventually, the system will be tested in a pre-production environment that mirrors the production environment. The software test plan should describe how to construct the pre-production environment.

Why It Matters to the DT&E Community

The test community bears the primary responsibility for writing software test plans. Based on requirements and government policy, the test community must produce a planning document that, when followed, helps ensure implementation of a system that can perform mission capabilities.

Software test plans are partly written by the developmental organization, especially those parts for unit and integration tests. The SW ITT contributes to software test plans by describing unit and integration tests in the pre-production environment, which is an environment closer to the operational environment than to the development environment (and therefore more suited to certain kinds of tests, especially nonfunctional tests such as performance testing and penetration testing). The SW ITT also reviews the developmental organization's test plans for completeness and correctness.

The entire test community works together to create an integrated test matrix for a system. This matrix correlates requirements with test cases. During the writing of software test plans, the matrix is used to determine whether all requirements are tested. If certain tests are to be automated, the matrix can also be used to identify the set of tests that must be executed manually, which in turn can be used to predict the effort required to perform testing.

The DT organization, with input from the SW ITT, specifies the software factory infrastructure needed for testing. This input will include selecting SAST and DAST tools. If commercial software is to be used, the test community needs to prepare cost estimates for acquisition, installation, and maintenance.

The test community should estimate the costs of testing as part of writing software test plans. These costs include:

- Procurement and maintenance of testing tools, including software license fees.
- The staffing and effort required to implement testing in a software factory.
- The staffing and effort for tests and evaluations that cannot be automated: Human users must be allocated to operate a system, and analysts must verify that results conform to expectations.

The test community is responsible for ensuring that the software test plans describe how to test adherence to standards and best practices. Software tools can verify certain standards, and software test plans should list the standards to use. Manual code reviews may be required, and software test plans should state when they will occur. The test community is also responsible for ensuring that the software test plans contain the testing standards and best practices that will be used. Listing these can justify testing approaches.

Finally, the test community is responsible for scheduling testing-related events. The test community must decide when the tests they design are to be executed. This decision is based on when software factory testing infrastructure must be in place, which cannot happen until the overall software factory exists.

Software test plans result from all these activities. They are used to produce the definition of done (discussed in Section 5.2.1) and are also reviewed by the PM. The plans should explain what will be tested, using a product such as an integrated test matrix. For each requirement, the plans should also describe how the requirement is tested.

Table 5-3 describes scope of the Software Test Plans DT role by T&E activity.

Table 5-3. Software Test Plans - T&E Activities and DT Roles

T&E Activity	Description of Developmental Testing Role
Unit Test	Determine standards and best practices for unit testing, in anticipation of using them during reviews
Integration Test	Determine standards and best practices for integration testing, in anticipation of using them during reviews

T&E Activity	Description of Developmental Testing Role
System Test	• Plan system testing approach • Determine standards and best practices for system testing, in anticipation of using them during reviews
Cyber Test	• Plan cybersecurity evaluation • Determine standards and best practices for cybersecurity, in anticipation of using them during reviews

Lessons Learned

Information will be added to the guidebook based on stakeholder feedback.

5.2.3 Threat Modeling

Threat modeling takes place within the plan phase of the T&E DevSecOps pipeline and is a proactive, holistic approach to analyzing potential threats and risks in a system or application to identify and address them. Threat modeling involves analyzing how an attacker might try to exploit weaknesses in the system and then taking steps to mitigate those risks. It enables informed decision making about application security risks. In addition to producing a model diagram, the process also produces a prioritized list of security improvements to the conception, requirements gathering, design, or implementation of an application. Threat modeling should be integrated with MBCRAs (further discussed in Section 5.2.4) *and required by* the forthcoming DoDM 5000.UY.

One popular threat model is the spoofing, tampering, repudiation, information disclosure, denial of service, and elevation of privilege (STRIDE) framework. This systematic approach is used to identify and analyze potential security threats and vulnerabilities in software systems. It provides a structured methodology for understanding and addressing security risks during the design and development stages of a system.

The acronym STRIDE stands for the six types of threats that the framework helps to identify as defined in Table 5-4.

Table 5-4. STRIDE Framework

Threat Type	Property Violated	Threat Definition
Spoofing	Authentication	Pretending to be something or someone other than yourself
Tampering	Integrity	Modifying something on disk, network, memory, or elsewhere

Threat Type	Property Violated	Threat Definition
Repudiation	Non-repudiation	Claiming that you did not do something or were not responsible; can be honest or false
Information Disclosure	Confidentiality	Providing information to someone not authorized to access it
Denial of Service	Availability	Exhausting resources needed to provide service
Elevation of Privilege	Authorization	Allowing someone to do something they are not authorized to do

STRIDE is a highly flexible approach, and getting started is not complex. Simple techniques such as brainstorming and whiteboarding may be used. STRIDE is also incorporated into popular threat modeling tools such as the OWASP Threat Dragon and the Microsoft Threat Modeling Tool. STRIDE combines with more tactical approaches such as kill chains or MITRE ATT&CK.

After possible threats have been identified, they are typically ranked. Ranking should be based on the mathematical product of an identified threat's likelihood and its impact. A threat that is likely to occur and would result in serious damage would be prioritized much higher than a threat that is unlikely to occur and would have only a moderate impact. However, both factors can be challenging to calculate, and they ignore the work to fix a problem.

During the mitigation activity, each threat identified earlier must have a response. Threat responses are similar, but not identical, to risk responses. Some typical responses include the following:

- Mitigate: Take action to reduce the likelihood that the threat will materialize.
- Eliminate: Simply remove the feature or component that is causing the threat.
- Transfer: Shift responsibility to another entity such as the customer.
- Accept: Do not mitigate, eliminate, or transfer the risk because none of these options is acceptable given mission requirements or constraints.

If a threat needs to be mitigated, mitigation strategies must be formulated and documented as requirements. Depending on the complexity of the system, nature of threats identified, and process used for identifying threats (STRIDE or another method), mitigation responses may be applied at either the category or individual threat level. Mitigations strategies should be actionable and be able to be built into the system. The strategies must be tailored to the application; resources such as the OWASP Application Security Verification Standard (ASVS) (see OWASP Application Security Verification Standard (ASVS) Website) and MITRE Common Weakness Enumeration (CWE) list (see MITRE Common Weakness Enumeration Website) can prove valuable when formulating these responses.

Throughout the validation and review process, the threat model must be reviewed by all stakeholders. Focus areas include the following:

- Does the data flow diagram (or comparable) accurately reflect the system?
- Have all threats been identified?
- For each identified threat, has a response strategy been agreed upon?
- For identified threats for which mitigation is the desired response, have mitigation strategies been developed that reduce risk to an acceptable level?
- Has the threat model been formally documented? Are artifacts from the threat model process stored in such a way that the model can be accessed by those with "need-to-know"?
- Can the agreed-upon mitigations be tested? Can success or failure of the requirements and recommendations from the threat model be measured?

For more information on threat modeling and frameworks, see the MITRE ATT&CK Website the OWASP Threat Modeling Website.

Objective

The objective of threat modeling during the plan phase of T&E within the DevSecOps life cycle is to identify security requirements; pinpoint security threats and potential vulnerabilities; quantify threat and vulnerability criticality; and prioritize remediation methods. During the plan phase, T&E threat modeling seeks to identify potential security issues allowing security to be integrated into a system rather than implementing it as an afterthought. This approach is far more efficient than having to identify and resolve security flaws after a system is in production. Because threat modeling is a design-time activity, it occurs during the plan phase of T&E before code review, code analysis (static or dynamic), and penetration testing; these activities all come later in the security life cycle.

Why It Matters to the DT&E Community

The test community uses threat modeling because it has emerged to proactively identify vulnerabilities and anticipate threats before they become significant issues. This approach allows organizations to go beyond simply responding to incidents as they happen and enables them to create more comprehensive security strategies that anticipate and prevent future attacks. Specifically, during penetration testing, strategies and scenarios can be developed that anticipate and prevent future attacks. By using the power of predictive risk-based methodologies, developing robust security controls, and conducting regular assessments, risks can be mitigated before they become a reality.

Table 5-5 describes scope of the Threat Modeling DT role by T&E activity.

Table 5-5. Threat Modeling - T&E Activities and DT Roles

T&E Activity	Description of Developmental Testing Role
Unit Test	N/A
Integration Test	N/A
System Test	Determine criteria for assessing developer’s threat modeling approach
Cyber Test	Determine threat modeling approach

Lessons Learned

Information will be added to the guidebook based on stakeholder feedback.

5.2.4 Mission-Based Cyber Risk Assessment

MBCRAs are the analytical process of identifying, estimating, assessing, and prioritizing risks based on the impacts on DoD operational missions resulting from cyber effects on the system(s) being employed. MBCRAs exercise threat modeling in DevSecOps and throughout the DevSecOps life cycle. These methodologies identify the risks associated with cyber threats and other system risks resulting from the failures in cyberspace of digital components in a system to focus test planning and execution.

During the MBCRA, the assessment team identifies potential risk areas, nodes of interest, critical data, critical functions, and the potential mission impacts that could compromise a data element’s confidentiality, integrity, or availability. The MBCRA informs test planning, preparation, and execution.

The MBCRA will identify specific areas of assessment for cyber DT&E such as cyberspace attack interfaces, key targets, and critical functions. MBCRA results provide insight into anticipated DT&E resources and methodologies. This information helps develop a comprehensive cyber DT&E strategy and informs other documentation such as the statement of work (SOW) or statement of objectives (SOO), program protection plan (PPP), TEMP or test strategy, and cybersecurity strategy (CSS).

The forthcoming DoDM 5000.UY will provide additional information on MBCRA minimum requirements and procedures.

Objective

The objective of an MBCRA is to identify potential cyber vulnerabilities during design and throughout the life cycle and generate cyber requirements that will mitigate those vulnerabilities. MBCRAs should inform decisions associated with engineering, transition (to warfighter, program of record), knowledge points or milestones, risk management, T&E, defensive TTPs, and operational utilization activities. Each assessment iteration updates previously generated assessment products to incorporate test results and changes in design, environment, mission, and threats, providing internal and external stakeholders with actionable information. The MBCRA should be consistent with DoDI 5000.89 and the forthcoming DoDM 5000.UY policy.

Why It Matters to the DT&E Community

MBCRAs inform automated and manual testing strategies, optimize limited resources, and enable software development planning activities. This information helps develop a comprehensive cyber DT&E strategy and informs other documentation such as the SOW/SOO, PPP, TEMP or TES, and CSS.

Test personnel should demonstrate and build upon the MBCRA cyberspace-attack scenarios through automated scripts or penetration tests and hands-on test activities to provide a complete analysis of anticipated cyber resilience in an operational environment.

The CyWG should update the MBCRA products to include mitigations and actions discovered during test events. More detailed information will be provided in Appendix G in the forthcoming Version 3.0 of the DoD Cyber DT&E Guidebook.

Table 5-6 describes scope of the MBCRA DT role by T&E activity.

Table 5-6. MBCRA - T&E Activities and DT Roles

T&E Activity	Description of Developmental Testing Role
Unit Test	N/A
Integration Test	N/A
System Test	N/A
Cyber Test	Plan and conduct MBCRA

Lessons Learned

DoD has applied years of lessons learned and MBCRA best practices and independent MBCRA analysis of DoD Component efforts toward the current policy and guidance. The DoD Cyber Table Top Guide, Version 2.0, provides a lightweight, tailorable approach as an MBCRA methodology that multiple software pathway programs have executed successfully.

5.3 Develop/Build Phase

5.3.1 Unit Testing

During the develop phase, developers design and implement modules in accordance with the project's system and software architectures. As the developers implement modules, they build a suite of unit tests for each module. Each unit test verifies that a module behaves according to one or more of its requirements.

A project should maintain a traceability matrix that describes, for each system and software requirement, the module(s) that implement that requirement. The traceability matrix provides an estimate of the amount of functionality verified during the develop and build phases.

Oftentimes a requirement is implemented through the interactions of multiple modules. Verifying the implementation of these requirements must wait until integration or system testing (discussed in Section 5.4.2).

Unit testing is ultimately driven by software test plans (discussed in Section 5.2.2) describing the nature of unit testing (objectives, standards, guidelines, and best practices) as well as dates and milestones. Software test plans do not usually describe test suites for every module because much of the plans' content is written before the software architecture is finalized. If certain critical modules are identified early, software test plans may include information on performing the module's unit tests.

Objective

The objective of unit testing is to ascertain the correct implementation of software modules. Unit testing is performed according to directives and guidance in the software test plans, developed during the plan phase (discussed in Section 5.2.2). The development team develops most unit tests. DT&E teams are responsible for ensuring that the development team has implemented a set of unit test suites that sufficiently exercise module functionality and verify nonfunctional characteristics such as performance and robustness. Work products produced during the plan phase will describe sufficiency. Sufficiency might entail using methods such as risk management or STAT. The DT&E teams assess unit test suites on whatever criteria the software test plans prescribe. These assessments should consider questions such as the following:

- Suppose a module implements a requirement: What type of data does the requirement describe as input? How can that data be obtained? How can the correct output be determined for an input?

- What requirements are not being tested during unit testing? A traceability matrix is a useful tool to answer this question.
- Are boundary cases being tested? If a module is required to store up to n values, does its unit test suite include tests that provide n values? What about $n + 1$ values?
- Are incorrect inputs being tested? If a module implements a protocol, what happens when it receives an input that is semantically or syntactically invalid? If a module accepts a Unicode string as input, what happens if it is given a string containing a non-Unicode character?

The DT&E teams should determine whether unit test suites address these questions.

Paragraph 5.2.a. of DoDI 5000.89 states that technical requirements must be measurable, testable, and achievable. Developing unit tests is the definitive determination of testability. If a module implements a requirement but no unit test can be devised for that requirement, either the requirement is not testable and should be rewritten or the module is a poor expression of the requirement, and the software architecture needs to be reconsidered.

DoDI 5000.89 provides other guidance that should be used during unit testing:

- In accordance with Paragraph 5.2.b.(1), DT&E verifies achievement of critical technical parameters (CTPs). CTPs should be traced to unit tests where possible. If a unit test verifies a CTP, or some aspect of a CTP, that verification should be noted. The percentage of CTPs achieved is a useful metric to track progress.
- Paragraph 5.2.b.(10) states that DT&E should assess compatibility with legacy systems. During unit testing, it is important to note the standards used, the protocols introduced or deprecated, and similar considerations that influence compatibility.
- Paragraph 5.2.b.(18) states the need to support RMF security controls. The DT&E team should assess whether unit test suites consider security controls.

A best practice in DevSecOps is to automate unit testing as much as possible. The DT&E team should review the software factory artifacts that implement automation to verify that unit tests are performed as expected.

Why It Matters to the DT&E Community

The test community is responsible for ensuring that test plans are properly implemented on schedule. In DevSecOps, unit tests must be implemented during the develop phase. The absence of unit tests should be a warning the project may not be ready to begin the build phase.

The development team writes most unit tests. The DT&E team conducts reviews of unit tests, preferably as tests are written. The DT&E team should include members with experience in unit testing. This experience should include unit testing technologies, tools, and infrastructure. For example, software factories often include tools to generate unit test frameworks. The DT&E team should investigate the tools proposed by the development team and make recommendations on their suitability. DT&E team members should also be able to assess whether unit tests properly implement testing methodologies such as STAT and whether test results provide data necessary to manage risk.

DT&E team members should possess domain knowledge about the system being tested. They should know the kinds of data that are realistic as module inputs and outputs and should be able to propose (and possibly implement if the development team cannot do so) strategies for obtaining realistic data.

In waterfall SDLC frameworks, the development team presents the DT&E team with testing-related work products and results at the end of the testing phase. The DT&E team reviews these materials and provides feedback, potentially requiring the development team to redo and repeat tests. This workflow is too slow for DevSecOps. Unit testing and review should occur module by module and not for all modules in an entire system. DT&E team members should have access to the development team's work products as soon as they are developed. This access can be achieved in at least two ways. First, the development team's software factory can be configured to send work products to the government team whenever a developer places a work product in the software factory's repository (or changes an existing work product) and to send unit test results to the DT&E team immediately after completing the execution of a unit test suite. Second, the DT&E team can be granted access to the software factory. DT&E team members can inspect work products, including unit tests, as they are developed.

The first access approach requires more effort to configure, but it alerts the DT&E team whenever an event of interest to testing occurs. The second approach is easier to implement and gives DT&E team members more immediate access, but it relies on the initiative of the DT&E team to seek out work products relevant to testing. The approaches are not mutually exclusive, and a program may wish to implement both.

No matter which approach is chosen, the DT&E team must review the software factory's configuration. Software test plans will describe which parts of testing are to be automated. The DT&E team should ensure automation is implemented properly.

The DT&E team is also responsible for creating unit tests specified in the software test plans that cannot be effectively implemented by the development team in the software factory. Common reasons for having to test a module outside a software factory include:

- The requirement to use classified data as input to a module that is developed on an unclassified system.
- The need to evaluate a module's performance by executing it on specialized hardware.

The cyber test team should ensure that unit testing accounts for cybersecurity. Many security flaws, it should be noted, stem from system design errors (e.g., unencrypted network traffic, improper startup configuration, or use of a library that has a security flaw) and reveal themselves during system tests rather than unit tests. Of course, some modules are intended to support security, and unit test suites for these modules should reflect guidelines, best practices, and standards for cyber testing. For example, buffer overflows are a notorious exploit. Any module with a buffer should have unit tests to determine its buffer overflow susceptibility. SAST (discussed in Section 5.3.2) should also be used to detect coding errors that might lead to security flaws. Some cybersecurity flaws are caused by a lack of robustness (e.g., failure to verify maximum string length has been used to introduce malware), so unit tests for robustness are important in building secure systems.

At the end of the build phase, the DT&E team and the development team are jointly responsible for deciding whether the modules are robust enough to proceed with integration and system testing. Best practice dictates that every test in a unit test suite should succeed before a module can be considered ready for integration. Depending on the system architecture, it may be possible to proceed partially with integration testing, testing only those modules whose unit test suites execute without errors. System testing cannot begin until all modules pass their unit tests. The DT&E team should keep the T&E WIPT and PM apprised of the state of unit testing.

Table 5-7 describes scope of the Unit Testing DT role by T&E activity.

Table 5-7. Unit Testing T&E Activities and DT Roles

T&E Activity	Description of Developmental Testing Role
Unit Test	• Review unit tests for comprehensiveness and sufficiency • Review unit testing framework in the software factory • Develop and execute unit tests that cannot be implemented in the software factory • Execute any nonautomated unit tests and respond to results
Integration Test	N/A
System Test	N/A
Cyber Test	Inspect unit tests for evaluation of module cybersecurity

Lessons Learned

Information will be added to the guidebook based on stakeholder feedback.

5.3.2 Static Application Security Testing in Develop/Build Phase

SAST is a type of security testing for source and object code. It examines the system in a non-executing state. SAST is often one of the first lines of defense in identifying security vulnerabilities through testing.

Most SAST is performed using automated tools, although manual reviews can be considered a type of SAST. In DevSecOps, the developmental organization chooses a set of SAST tools and integrates them into the software factory. The DT&E team should review and approve the choices.

Successful completion of SAST is required for promotion from the development environment to the test environment. The meaning of "successful completion" should be defined during the plan phase. It usually means that every module has been scanned by the SAST tool suite, and the development team has addressed every issue the tools have found. Projects sometimes opt to ignore certain errors, but this practice is discouraged.

During the develop and build phases, SAST can be performed within the software factory. During the test and release phases, some SAST may need to take place outside the software factory (discussed in Section 5.4.1).

Every module in the software repository should undergo SAST. Ideally, every check-in triggers SAST. Executing a SAST tool does not generally have a high overhead. Many modern compilers can perform some SAST.[3] The development team should specify the options that developers are required to use when compiling their own code and, as a backup, require those options when compiling code in the software repository. The DT&E team's responsibility extends to reviewing how the development team invokes compilers.

Objective

The goal of SAST is to find and correct security issues at the level of software source code and object code (compiled code that is not executing) as quickly as possible. For custom code (i.e., not open source or other reused code), vulnerabilities are identified and categorized based on known common weaknesses in code such as those in the MITRE CWE catalog. Because of its "white box" nature, SAST can find code issues (e.g., improper input validation[4]), but emergent

[3] For example, recent versions of the GNU Compiler Collection (GCC), which includes front ends for C/C++, have a static analysis option. See Static analysis in GCC 10 at https://developers.redhat.com/blog/2020/03/26/static-analysis-in-gcc-10.
[4] See https://cwe.mitre.org/top25/archive/2023/2023_kev_list.html.

properties of software such as performance issues may not be apparent until other forms of testing are conducted.

SAST tooling is provided by developers and integrated into the software factory. Testers provide oversight on the choice of tooling. SAST execution begins when code is checked in. The automated build and test process triggered by check-in starts the SAST. SAST should be automated as much as possible, but human review of source code may be required for specific sections of code in which failure or compromise could result in greater mission impact.

Why It Matters to the DT&E Community

SAST can find issues in source and object code so they can be corrected quickly and with minimal impact. The MVP must have SAST tooling as part of the software factory.

At the develop and build phase, testers should already have a SAST test strategy (defined during the plan phase). They should document SAST test sources (e.g., drawn from CWE), and developers and testers should agree where and how the tests and results should be stored and shared with the government. Additionally, processes should be in place to notify developers of any test failures in an issue tracking system. The SAST test strategy should clearly define which types of test failures would result in immediate notification to government testers (e.g., a vulnerability found that has an impact on a system already in production). Testers should ensure the SAST automated tooling executes as expected and that manual SAST is conducted properly during the develop/build phase and the test/release phase. Processes should ensure that failures found later in the development process—that could have been identified through a static test—result in the creation of a regression test case if possible. SAST output includes a static code scan report with recommended mitigations.

Reductions of specific CWE findings over time may indicate increased code quality. The DT&E team may want to ask the development team to instrument the software factory to capture CWE findings. The resulting reports can yield metrics that indicate the state of code quality over time. The DT&E team can present these metrics to the PM, who can use them when estimating whether a project is on track to achieve its milestones.

Table 5-8 describes scope of the SAST Develop/Build Phase DT role by T&E activity.

Table 5-8. SAST Develop/Build Phase - T&E Activities and DT Roles

T&E Activity	Description of Developmental Testing Role
Unit Test	• Determine adequacy of: o SAST tooling o SAST coverage

T&E Activity	Description of Developmental Testing Role
	• Agree on triggers for SAST • Agree upon details of what SAST output is reported as well as how, when, and in what format • Review SAST reports for areas of concern
Integration Test	N/A
System Test	N/A
Cyber Test	• Determine adequacy of: o SAST tooling o SAST coverage • Agree on triggers for SAST • Agree upon details of what SAST output is reported as well as how, when, and in what format • Review SAST reports for areas of concern

Lessons Learned

Information will be added to the guidebook based on stakeholder feedback.

5.3.3 Functional Testing

Functional testing evaluates one portion of a system at a time to ensure that specific functions perform as expected. These functions may involve multiple modules or subsystems that support proper functionality. Functional testing differs from integration testing where the focus is on top-level functionality vice lower-level system interactions. In describing the different types of software testing, the Atlassian Website states, "There is sometimes a confusion between integration tests and functional tests as they both require multiple components to interact with each other. The difference is that an integration test may simply verify that you can query the database while a functional test would expect to get a specific value from the database as defined by the product requirements."[5]

Objective

Functional testing is meant to test system functionality one function at a time at a system level. It can be traced to higher-level functional requirements, ensuring that functionality meets those specific requirements. Functional testing includes dependencies of the specific function.

[5] https://www.atlassian.com/continuous-delivery/software-testing/types-of-software-testing

Why It Matters to the DT&E Community

Functional testing isolates one function at a time at the system level to ensure that the SUT meets its functional requirements. By isolating one function at a time, emergent system-level issues can be detected, localized, and mitigated before system-wide testing.

Top-level functional requirements should be clearly documented and measurable in the plan phase. These requirements form a traceability matrix of subsequent lower-level functional requirements and associated tests that are performed later. Testers should ensure that key functions (i.e., those critical to system functionality and security) go through the functional testing process. For each function tested, developers create and execute test cases, which are traceable to functional requirements. Testers ensure that functional tests are conducted as expected and are sufficient to detect and mitigate risks of function failure.

Table 5-9 describes scope of the Functional Testing DT role by T&E activity.

Table 5-9. Functional Testing - T&E Activities and DT Roles

T&E Activity	Description of Developmental Testing Role
Unit Test	N/A
Integration Test	N/A
System Test	• Developers create tests to verify that functions are executed such that they meet functional requirements; functional issues are promptly mitigated and communicated to key stakeholders if they apply to fielded systems. • Testers validate that the tests are sufficient to appropriately mitigate system risk and that any remaining functional risks are clearly communicated to stakeholders.
Cyber Test	• Developers create tests to verify that functions are executed securely; security issues are promptly mitigated and communicated to key stakeholders if they apply to fielded systems. • Testers validate that the tests are sufficient to appropriately mitigate system risk and that any remaining security risks are clearly communicated to stakeholders.

Lessons Learned

Information will be added to the guidebook based on stakeholder feedback.

5.3.4 Regression Testing in the Develop/Build Phase

Regression tests help to prevent the introduction of errors into software that has already undergone testing and was subsequently changed. Developers are responsible for writing, selecting, and executing the specific tests. Government testers oversee the regression test process to ensure that the tests are created as needed, selected as appropriate, and executed on time.

Government testers also evaluate the test results to determine and communicate the software quality.

Regression tests do not occur in the first DevSecOps iteration. A regression test is conducted to ensure that a change in a subsequent iteration does not cause unwanted behavior. In each iteration, a project needs to plan for future regression testing.

A regression test can focus on a module, an integrated set of modules, a subsystem, or a system. Bug fixes whose changes are confined to a single module require regression tests for that module. However, it is not uncommon for software to depend, knowingly or unknowingly, on a module's erroneous behavior, so changes to one module mean that regression tests should be executed on all modules making use of that module; ultimately, regression tests should be applied to the entire system. With that caveat, regression testing in the develop and build phases focuses more on modules than on the system. (Here it is appropriate to channel the attitude of the signers of the Agile Manifesto: We are not saying that system regression testing during the develop and build phases is unimportant, but rather that module regression testing is more important.)

Objective

Regression testing is the execution of tests to ensure that changes to software ensure the integrity of previously completed capabilities while not introducing errors. Regression test cases can be functional or nonfunctional (sometimes also referred to as "quality attributes"). The goal is to avoid breaking existing code and functionality. Regression testing should be integrated into the software factory and automated as much as possible to catch bugs quickly.

Regression test suites, by nature, grow with time. To ensure they do not unnecessarily burden software development, developers may need to prune the test suites to ensure that duplicates and tests of deprecated code are removed. It still may not be feasible to run all regression tests at once, so a selection of regression tests may be necessary. In this case, regression tests for the changed code and adjacent modules (i.e., modules that interact at runtime with the changed module) are the minimum that should be selected and executed. If the change is architecturally cross-cutting (e.g., upgrading the cryptography used by all networked modules' communications), then more comprehensive testing is necessary. Ultimately, the selection of regression test cases should be based on mission risk. Government testers are responsible for reviewing the test suite to ensure that the test coverage and depth remain sufficient to properly manage risk.

Why It Matters to the DT&E Community

Regression testing reduces the reintroduction of software bugs that were addressed earlier. By automating regression tests, developers receive feedback quickly in the development process. Regression testing minimizes the test failures that must be addressed later in the development life cycle when it becomes more costly and complicated to fix code.

In the develop and build phases, developers and testers should agree upon the general approach to regression testing, and the approach should be clearly documented. The regression testing approach includes determining the scope of regression testing; processes for adding new tests to the existing regression test suite; which regression tests are executed under which circumstances; and how and where the regression tests and test results will be stored. The approach should clarify which tests will not be automated and why. The use of issue tracking software is also a key part of the approach to regression testing; developers and testers should agree which regression test failures result in automatic issue creation and how the issues are assigned. Test case failures at the very least should result in alerts or notifications to relevant stakeholders. Key outputs here include the test tool suite and test information that will be shared with the government. An initial set of regression test cases created by developers will be part of the develop and build phases.

In the test and release phases (section 5.4.3), the developer tests new capabilities using new tests. The selection of tests should be based on factors (e.g., number of tests run, prior test coverage, history of failures) including mission risk and whether the code changes could conceivably have affected the results of the test consideration. Automated selection of test cases should be explainable to testers. Testing should prioritize risk to the mission in addition to issues that could be costly and areas of the system that are prone to error. Additionally, new test cases are added to the regression test suite as new issues are encountered. Test failures result in alerts or notifications. Successful completion of regression testing is required for a system to be promoted via CI to the next environment. The key output here is a test report produced by developers. The report should provide sufficient detail (e.g., coverage, failed tests, remediations) for testers to adequately judge the quality of the software under test. Testers should review the results and report any discrepancies to the developers and project management staff.

A key measurement to understand the timeliness of the DevSecOps process is the time between the identification of a new requirement and the completion of regression and cyber testing. Additionally, time spent on automated and manual regression testing is key to understanding the extent to which testing might be unnecessarily slow or may be slowing development with minimal value added.

Table 5-10 describes scope of the Regression Testing Develop/Build Phase DT role by T&E activity.

Table 5-10. Regression Testing Develop/Build Phase - T&E Activities and DT Roles

T&E Activity	Description of Developmental Testing Role
Unit Test	Develop initial sets of regression tests and a process for when and how to create new tests as issues are found. Subsequently plan for archiving, maintaining, selecting, and reusing tests.
Integration Test	Develop initial sets of regression tests and a process for creating new tests as issues are found. Subsequently plan for archiving, maintaining, selecting, and reusing tests.
System Test	Develop initial sets of regression tests and a process for creating new tests as issues are found. Subsequently plan for archiving, maintaining, selecting, and reusing tests.
Cyber Test	Ensure that regression tests address cybersecurity.

Lessons Learned

Information will be added to the guidebook based on stakeholder feedback.

5.3.5 User Story Demonstrations

During the plan phase, customers, operators, and developers work together to create user stories. A user story shows how a system supports a mission capability. The development team uses a user story demonstration to prove it has implemented that part of a system needed for a mission capability.

The scope of a user story depends on the mission capability it describes. The capability may require several requirements to express it. Conversely, it may emphasize a single, narrowly focused requirement that performs a specific, easily described piece of functionality. In the develop/build phases, the latter kind of user story is the one of interest. The developmental organization will want to demonstrate certain pieces of system functionality to let system users see how the pieces will appear in an implementation. The system users play the role of the eventual system operators and judge the demonstration not only on whether the capability is correctly implemented but on the ease of using the system.

Objective

The DT&E team's objective in a user story demonstration is to help make the demonstration a success. Success means two things. First, success means that the demonstration proceeds flawlessly—that it occurs without the system crashing; with the system executing according to performance expectations; and with the expected results easily visible and accessible to the audience. Second, success means that the demonstration yields feedback. System users should have the opportunity to respond to the demonstration, requesting changes they think will improve usefulness. System users should be working users, not stakeholders; this does not mean that stakeholders should be excluded from user story demonstrations, just that their feedback on day-to-day use is of less importance than that of the operators. This focus on operators is part of the "Ops" phase in DevSecOps.

The DT&E team has several roles to play in improving the prospects for success. The DT&E team should:

- Determine whether the user story is worth demonstrating: Will a demonstration provide useful feedback?
- Review the demonstration to decide the degree to which it applies to the user story.
- Decide whether the data used as input to a demonstration is realistic enough to be convincing.
- Ensure that the system components used in the demonstration are tested enough to be used with a minimal prospect of failure.
- Decide whether outputs generated are easily accessible and understandable—in other words, whether the users will believe the system produced the expected results.
- Determine how to evaluate a user story demonstration.
- Evaluate the results of user story demonstrations to determine the degree to which the system as currently implemented provides a satisfactory demonstration of the user story and how the system could be improved.

The DT&E team's objectives also include determining whether tests cover nonfunctional requirements adequately. A user story may describe a nonfunctional system quality, such as a system's resiliency in the face of a cyberattack or behavior under high load. Configuring and operating an environment in which to perform a corresponding user story demonstration can be complicated; the scenario may assume more actors (e.g., each operating on a separate platform, even if virtual) and their corresponding inputs than a user story focused on normal system operation.

Why It Matters to the DT&E Community

The DT&E team has multiple obligations to see that DT objectives for user story demonstrations are satisfied. These obligations include:

- Knowing the user stories being demonstrated. User stories are developed during the plan phase. The DT&E team should study them as they are released, giving the team time to plan for conducting and evaluating demonstrations during the build phase.
- Understanding each user story's purpose. A user story should be traceable to one or more requirements. If requirements are testable, measurable, and achievable (discussed in Section 5.3.1), then understanding the requirements underlying a user story can guide the DT&E team in preparing the evaluation criteria for a demonstration. (The development team may wish to cooperate in this activity.)
- Ensuring that the code to be used in the demonstration has been tested enough to make failure unlikely. The DT&E team will review relevant tests developed to date that pertain to the modules being used in a demonstration. These can be any kind of test: unit, integration, system, or regression. They test not only functional behavior but also such nonfunctional properties as the ability to handle invalid inputs gracefully as well as satisfactory runtime performance. The DT&E team may wish to write its own test suite to support a demonstration or may request that the development team write one. The decision on whether a user story demonstration is ready should be informed by the assessments of the DT&E team.
- Evaluating a user story demonstration. Some demonstrations do not require formal evaluation. They only seek to convince users and stakeholders that system development is progressing as expected. Other demonstrations may ask for user input: preferred interface style, most useful data visualization format, etc. In these cases, the minimum effort is to ask users for their opinions; but especially for a user story derived from a testable and measurable requirement, the DT&E team should design, or assist in the design of, an environment in which users' capability to operate a system can be measured and quantitatively assessed. After the demonstration, the DT&E team is responsible for analyzing the results.
- Giving feedback on the success of a demonstration and any changes suggested. The DT&E team is not responsible for revising requirements, design, or code based on demonstration results, but it is responsible for providing the inputs that guide and influence the decision makers who authorize revisions.

Table 5-11 describes scope of the User Story Demonstrations DT role by T&E activity.

Table 5-11. User Story Demonstrations - T&E Activities and DT Roles

T&E Activity	Description of Developmental Testing Role
Unit Test	Ensure that modules to be used in a user story demonstration have undergone enough unit testing to be reliable for the demonstration

T&E Activity	Description of Developmental Testing Role
Integration Test	Ensure that modules to be used in a user story demonstration have undergone enough integration testing to be reliable for the demonstration
System Test	Ensure that modules to be used in a user story demonstration have undergone enough system testing to be reliable for the demonstration
Cyber Test	Ensure that modules to be used in a user story demonstration have undergone enough cybersecurity testing to withstand attacks that are part of the demonstration

Lessons Learned

Information will be added to the guidebook based on stakeholder feedback.

5.3.6 Software Composition Analysis

SCA is an activity in which the development team studies open-source software components used to develop a software system. Many projects can utilize the vast resources of open-source software to provide functionality they would otherwise have to develop. A DevSecOps project, with its emphasis on rapid iterations, should expect to integrate open-source software rather than developing functionality in-house.

The use of open-source software has become standard practice. Once developers learn to use one library, investing the time and effort to learn another is of dubious benefit unless the new library offers new capabilities. This probably explains why, according to one study (Vailshery, Lionel Sujay), in 2021, 60 percent of data scientists programming in Python used the open-source NumPy library,[6] and 55 percent used pandas.[7]

A program using DevSecOps must keep in mind that open-source software is not limited to the software libraries imported and the infrastructure used to execute a system. A software factory likely contains open-source software; therefore, it too must be analyzed.

Using open-source software has risks. An organization must accept that it will not have time to review open-source software's source code and must trust the open-source software developers' competence. Because programming is complicated and attackers are ingenious and persistent, complete trust is not justified, as a search of public vulnerability databases will reveal.[8] The trust issue is somewhat ameliorated by these databases, which exist to record known vulnerabilities and can be searched to check whether a particular component has vulnerabilities—or better yet, whether a particular version of a particular component has vulnerabilities. Vulnerability

[6] https://numpy.org/.
[7] https://pandas.pydata.org/.
[8] See, for instance, the NIST National Vulnerability Database at https://nvd.nist.gov/.

databases have an obvious drawback: Because they record only publicly known vulnerabilities, they do not guarantee that a component is safe. They are, however, the best way to conduct a low-effort check for vulnerabilities.

Programs that develop software in an unclassified environment but test and deploy it in a classified environment must consider the delays involved in approving software for use on a classified system. A component version that fixes a vulnerability may not appear on a classified system for some time. When performing SCA, an organization must be sensitive to problems in previous versions and aware of how using a fixed version during development might delay deployment in a classified environment.

SCA shares objectives with SCRM. Both SCA and SCRM are concerned with obtaining a product of sufficient quality; however, SCA has a narrower focus. SCA concerns open-source software and assumes that the software is available if one wants it; but unlike SCRM, SCA is not concerned with politics, terrorist threats to shipping routes, or transportation costs. SCA is also related to SBOM monitoring (discussed in Section 5.3.7). According to some perspectives, SCA generates an SBOM (Paliwal 2023). The accuracy of this claim depends on what is expected of an SBOM, which may contain elements that are irrelevant to SCA, and vice versa.

Many tools automatically perform SCA. A program using DevSecOps to implement software should be expected to select a suite of SCA tools; identify points in DevSecOps phases when these tools will be executed; and agree on the outputs of these tools and how to use them.

Objectives

SCA is, or should be, performed mainly by the development team. The DT&E team's role is to determine whether developers have done their due diligence in selecting open-source software. The DT&E team decides whether the development team has taken adequate measures to ensure the following about open-source software components:[9]

- Components do not have vulnerabilities (or the risk of their vulnerabilities can be managed). Because components evolve, it is important to know whether the development team has selected versions of components that do not have vulnerabilities.
- Component licenses do not violate intellectual property agreements. Intellectual property concerns are usually outside the scope of testing and evaluation, but the DT&E team should verify that the development team has prepared a report listing all software licenses

[9] These items are adapted from "Risk Management in Projects Based on Open-Source Software" (Nguyen Duc Linh et al. 2019, 178–183).

of open-source software used. This report will be approved by the PM before the release phase ends.

- Components are compatible with the software being developed; addressing potential problems such as an open-source software component using single-precision floating-point values when computations require double precision. The DT&E team's objective is to make sure testing is comprehensive enough to catch these kinds of incompatibilities.
- Components are not deprecated or obsolete. The DT&E team should investigate whether the development team has used open-source software that is no longer actively maintained or supported. An open-source library with an active user community provides an opportunity to ask questions about how to use the library. SCA objectives extend to indirectly used components: open-source software used by other open-source software. The DT&E team will want to determine whether SCA has examined vulnerabilities in every open-source software component used. This kind of analysis is usually limited; examining differences between versions of indirectly used components, or attempting to discover incompatibilities, is effort intensive. Components that are deprecated or obsolete must be rejected.

Why It Matters to the DT&E Community

Because the DT&E team's role in SCA mainly involves review, testers should have a review process to follow. This process will be defined during the plan phase and will describe:

- Work products that the development team is to provide to the DT&E team. These work products should document how the development team performs SCA as well as its results.
- Points in the develop and build phases where the development team will provide these work products.
- Analysis methods, assessment methods, and tools that the DT&E team will use to conduct its reviews.
- The expected time and effort needed to conduct reviews.
- Work products delivered as the output of a review.
- Recipients of a review.

Normally, if the development team identifies a problematic component through SCA, it must decide whether to not use the component, implement some work-around, or accept the risk of using the component. The development team should document its work as part of its evolving software design, and the DT&E team should be able to review these design changes. If the

DT&E team, in its review of SCA work products, decides the development team has overlooked a vulnerability, or has not adequately mitigated the risk of using a vulnerable component, the DT&E team should report its findings to the T&E WIPT, software lead, and PM, who may then decide whether the development team can continue.

Table 5-12 describes scope of the Software Composition Analysis DT role by T&E activity.

Table 5-12. Software Composition Analysis - T&E Activities and DT Roles

T&E Activity	Description of Developmental Testing Role
Unit Test	• Determine whether all open-source software used by a module being unit tested has been identified • Determine whether all open-source software has been subjected to SCA
Integration Test	Determine whether all open-source software used in a set of modules has been: • Identified • Subjected to SCA
System Test	Determine whether all open-source software used by a system has been: • Identified • Subjected to SCA
Cyber Test	• Assess whether the set of vulnerability databases used in SCA is adequate • Review the development team's assessments of cybersecurity risks revealed by SCA

Lessons Learned

Information will be added to the guidebook based on stakeholder feedback.

5.3.7 Software Bill of Materials Monitoring

The SBOM is a detailed description of the components and supply chain relationships used to build a software work product. Pairing an SBOM with a database of known vulnerabilities and exploits can reveal cybersecurity weaknesses and attack vectors. Preparing an SBOM and monitoring vulnerability databases are an important part of designing a secure system.

Executive Order 14028, "Executive Order on Improving the Nation's Cybersecurity," tasked NIST to publish guidance on providing software purchasers with an SBOM. As part of a larger effort pertaining to Executive Order 14028, NIST has published material on SBOMs addressing what they are; the problems they address; and regulations, standards, tools, and best practices for using them. For further information, see the NIST Guidance, "Software Security in Supply Chains: Software Bill of Materials (SBOM)." Although DoD does not have a single approach to using SBOMs (and neither do DevSecOps practitioners), the NIST material contains information that the DT community should use to establish the adequacy of cybersecurity reviews.

Monitoring an SBOM requires planning. The development team must decide on, and the DT&E team must accept, the SBOM standards and tools that will be used. The development team must also decide the process for creating and maintaining SBOM entries. Ideally, SBOM creation will be automatic; tools can create an SBOM by examining software build instructions. If any components cannot be identified automatically, the SBOM entry process should include control gates for verifying that the SBOM is up to date. The teams must keep in mind that an SBOM includes not only the software modules and the libraries they reference but every item in the DevSecOps ecosystem. The SolarWinds breaches, revealed in 2020, are a real-world example of how everyday tools in a development environment can have destructive effects; for further information, see The New York Times article, "Scope of Russian Hacking Becomes Clear: Multiple U.S. Agencies Were Hit" (Sanger, Perlroth, and Schmitt 2020).

Objective

The objective of SBOM monitoring is to capture all the software components used to build a product; determine whether any of those components are known to have vulnerabilities; and assess the effect of those vulnerabilities on the product and its operational use. Knowing a component's identifier is not enough. Flaws are introduced in specific component versions and can be fixed in later versions. To be useful in determining cybersecurity risk, an SBOM must identify both a component and a version of that component.

The development team's objective is to automate as much of the SBOM monitoring as possible by using tools that can:

- Produce an SBOM for the software being developed.
- Produce an SBOM for the development environment.
- Compare these SBOMs against a database of known vulnerabilities.

If these tools exist, then a report on vulnerabilities can be generated automatically each time a change is committed to the software repository during the develop phase and each time the DevSecOps ecosystem changes, which might occur at any time during an iteration.

The simplest approach to handling an identified vulnerability is to disallow the use of any flawed component. It is more likely that the DT&E team will define criteria to determine whether using a flawed component causes unacceptable risk. Currently, SBOM standards are not mature enough to permit the sort of automated analysis that would address risk; see The Minimum Elements for an SBOM for further elaboration, or even to answer whether a particular cybersecurity flaw can be exploited in a particular system. The DT&E team's objective is to ensure that all identified flaws are thoroughly analyzed for effects.

Why It Matters to the DT&E Community

The DT community has several responsibilities regarding SBOM monitoring, including:

- Approving the SBOM monitoring objectives and plans created during the plan phase.
- Approving the selection of tools, repositories, standards, processes, and practices to be used in SBOM monitoring.
- Approving the process of SBOM creation.
- Approving the process for checking software components in an SBOM against vulnerability databases.
- Assisting in vulnerability analysis and decisions on component approval or rejection.

In DevSecOps, process approval generally involves investigating the process's implementation in a software factory. The DT&E team will need to receive all the tools involved in creating and analyzing SBOMs or will need access to the development team's software factory. A software factory should log SBOM-related events. The DT&E team should expect to review log entries to verify that SBOMs are created, updated, and analyzed.

Standard SBOM formats such as defined on the CycloneDX Website and SPDX Website are not syntactically complex, but the data in an SBOM is extensive and the meanings of entries can be subtle. The DT&E team should include a member with expertise in the chosen format. A team member should also be able to interpret the content of the chosen vulnerability databases.

The most complex task that DT&E team members face in this activity is analyzing the SBOM and performing vulnerability analysis. Sometimes analysis is easy. For example, in 2019, the Federal Acquisition Regulations were amended to prohibit the use of any software products from Kaspersky Lab (see the Federal Acquisition Regulation: Use of Products and Services of Kaspersky Lab); therefore, if an SBOM lists any component from Kaspersky Lab, then DoD, GSA, and the National Aeronautics and Space Administration (NASA) cannot use the product. More often, analysis is more subtle. Suppose a vulnerability database identifies a flaw in a library component. The DT&E team must determine whether the product being tested uses that library component, as opposed to some other component in the library. In theory, the library's owners will publish an SBOM for the library listing dependencies within the library, which helps determine whether one component transitively uses another. But transitive use may depend on how an intermediate component is invoked. Furthermore, the DT&E team is trusting the accuracy and completeness of the library's SBOM. The context in which a component is used is also important. Some cybersecurity flaws are obviated by using a zero-trust architecture. Testers may want to accept the risk of a flawed component if they believe the flaw cannot be exploited.

The DT&E team is responsible for keeping the T&E WIPT and the PM informed of vulnerabilities as revealed by SBOM monitoring. Passing the control gate into operations requires preparing a report of any vulnerabilities and the reasons why the risk of deploying software with those vulnerabilities is acceptable.

Table 5-13 describes scope of the SBOM Monitoring DT role by T&E activity.

Table 5-13. SBOM Monitoring - T&E Activities and DT Roles

T&E Activity	Description of Developmental Testing Role
Unit Test	• Develop SBOM for module • Assess module SBOM based on: o Standards and regulations o Vulnerability databases
Integration Test	• Develop SBOM for integrated modules • Assess module set SBOM based on: o Standards and regulations o Vulnerability databases
System Test	• Develop SBOM for system • Assess system SBOM based on: o Standards and regulations o Vulnerability databases
Cyber Test	• Develop SBOM for: o Software factory o DevSecOps ecosystem • Assess SBOMs based on: o Standards and regulations o Vulnerability databases • If appropriate, publish SBOM

Lessons Learned

Information will be added to the guidebook based on stakeholder feedback.

5.3.8 Compliance Verification

Compliance verification is the process of ensuring that a system adheres to applicable rules and regulations. Rules and regulations include laws, such as accessibility as defined in Section 508 of the Rehabilitation Act (see GSA Section 508 website). They also include program-specific, contractually dictated requirements, such as complying with NIST Special Publication 800-53, "Security and Privacy Controls for Information Systems and Organizations, or obtaining and maintaining International Organization for Standardization (ISO)/International Electrotechnical

Commission (IEC) 27001 certification as defined in ISO/IEC 27001:2022, "Information security, cybersecurity and privacy protection — Information security management systems — Requirements." Rules and regulations may include compliance with industry-accepted best practices for quality and security.

A program establishes the rules, regulations, controls, and practices with which it must comply by the end of the plan phase. Some are codified in processes and activities developed in support of adherence. Activities may include information gathering in support of verification as well as reviews of reports prepared from this information. Processes may have milestones and control gates centered on achieving compliance.

Organizations recognized the importance of compliance verification and responded by developing software tools to automate portions of the compliance verification process. Version 2.2 of the DevSecOps Fundamentals Guidebook: DevSecOps Activities and Tools has tables that describe the activities during which compliance tools should be used. It splits these tools into two categories: security compliance and non-security compliance. Security compliance tools tend to emphasize testing for adherence to controls. Some focus on controls defined by standardization bodies such as ISO and NIST. Others are more granular, such as determining whether a system uses operating-system-specific mechanisms to limit access rights. An organization that uses the latter kind of tool needs to map the tool's verifications to its higher-level objectives.

Compliance verification overlaps with SAST (discussed in Section 5.3.2). For example, the tools that verify security compliance can be run as part of the processes that perform SAST. However, the rules and regulations that are legal documents can defy the sort of formalization needed to encode them for automated processing. Verifying compliance with those documents will have to be performed manually.

Compliance verification extends to the software factory. Tools used in the software factory are subject to the same rules and regulations as the system being built, and so is custom code written to help automate DevSecOps. Auditing regulations may require preserving supporting work products, including input data sets, outputs, and log files. For the same reasons, compliance verification also extends to the pre-production environment and the production environment.

Objective

The DT&E team is responsible for determining whether the development team has implemented a system that complies with applicable rules and regulations. Because the development team has the primary responsibility to put in place processes, activities, and tools to assess compliance, the DT&E team's objectives are threefold:

- Review the development team's decisions on how to verify compliance.
- Ensure the development team carries out compliance verification.
- Review compliance-verification-related work products that the development team produces.

The development team's software factory may not be an adequate environment in which to perform compliance verification. The DT&E team's review of compliance verification processes, activities, and tools should consider whether an additional environment needs to be used. For example, compliance verification tools related to security classification may not be available in an unclassified environment. The DT&E team must establish the adequacy of compliance verification in the software factory and, if inadequate, determine whether the development team has accounted for compliance verification in the production environment.

Why It Matters to the DT&E Community

The DT&E team's role in compliance verification is to assess the quality of compliance verification as performed by the development team and assist the development team as necessary. The PM and the software lead will use the DT&E team's results to determine whether a project can proceed to the next DevSecOps phase.

As part of compliance verification, the DT&E team should engage in the following activities:

- Ensure during the plan phase that the development team has:
  - Identified all applicable rules and regulations.
  - Defined processes to ascertain compliance verification. Compliance can include using specific tools and technologies that may be needed immediately upon entering the develop phase.
- Ensure that the development team follows its compliance verification processes:
  - The DT&E team will verify that the software factory properly automates those parts of the processes that can be automated. The DT&E team will need access to logs that record tool execution as well as to the outputs of automated tests.
  - For those parts of the processes that must be performed manually, the DT&E team will expect regular reports from the development team. The DT&E team may wish to participate in some compliance verification activities. For example, adherence to Section 508 may require walk-throughs of user interfaces that the DT&E team may wish to observe.

- Ensure that the development team has addressed compliance failures. The development team should be expected to provide reports to the DT&E team describing compliance verification successes and failures, along with plans and timelines to address the failures.
- Provide the T&E WIPT, the PrjMgr, and the software lead with compliance verification information.

Most software tools that perform compliance verification can be used during the develop and build phases. Compliance verification in an environment other than a software factory cannot occur until the environment is available, which may not happen until the test and release phases.

Table 5-14 describes scope of the Compliance Verification DT role by T&E activity.

Table 5-14. Compliance Verification - T&E Activities and DT Roles

T&E Activity	Description of Developmental Testing Role
Unit Test	• Ensure that compliance verification has been performed on modules • Verify that a module complies with rules and regulations
Integration Test	• Ensure that compliance verification has been performed on module sets • Determine whether module sets have passed compliance verification
System Test	• Ensure that compliance verification has been performed on a system • Verify that a system complies with rules and regulations
Cyber Test	• Ensure that processes, procedures, and tools properly implement tests for SaC • Verify that a system complies with cybersecurity rules and regulations

Lessons Learned

Information will be added to the guidebook based on stakeholder feedback.

5.4 Test/Release Phase

5.4.1 Static Application Security Testing in the Test/Release Phase

SAST during the test and release phases is like SAST during the develop and build phases (discussed in Section 5.3.2) but (theoretically) broadened in scope. During the develop and build phases, the focus is primarily on individual modules; during the test and release phases, the focus extends to subsystems and eventually to an entire system. The distinction can be irrelevant. Some SAST tools examine individual modules one at a time. A project using those tools will not gain any extra information by running SAST tools during the test and release phases. Nevertheless, the tools serve as a final quality-control step before release and being able to package their outputs along with all other testing outputs makes for a satisfying report to present to the PrjMgr.

The importance of SAST during the test and release phases also depends on the programming language. Strongly typed languages that require formal definitions of UIs, such as C++ and Java, catch many errors at compile time by their nature. Languages such as Python, which do not allow a function argument's datatype to be specified, are subject to runtime type conversion errors. A SAST tool that operates on multiple modules can detect some of these errors.

Performing SAST on an entire system, however, can be time consuming and resource intensive. The development team, with input from the DT&E team, must determine (1) how costly system-wide SAST will be; (2) whether the resources are available to perform system-wide SAST; (3) whether performing system-wide SAST will slow down development; and (4) whether strategies exist for selecting when to perform system-wide SAST that will reasonably achieve SAST objectives at acceptable risk levels.

Objective

The objective of SAST during the test and release phases is to ensure code quality of an entire system. To the extent possible, SAST is done using automated tools in the software factory so that the steps to discover problems are mechanical and repeatable, and the results are traceable. SAST will identify problems that must be corrected before a system may be delivered. Because these are inter-module problems, they can require conversation and coordination between module developers, possibly necessitating interface changes. They can be expensive (at least, relative to SAST-uncovered problems during the develop and build phases). The DT&E team will want to review the development team's results and plans, with the objective of verifying that the development team is adequately managing its resources to fix pre-execution defects.

Why It Matters to the DT&E Community

The DT&E team's role in this activity depends on whether the SAST tools provide more information than was generated during their use in the develop and build phases. If the tools do not provide more information, the DT&E team is simply responsible for ensuring that the PM sees a complete report of SAST activities—a report that could just as easily have been generated in the build phase. If the tools do provide more information, the DT&E team is responsible for ensuring that the information is acted upon. Except in unusual circumstances, a project should not be allowed to progress beyond the test or deliver phases if a SAST tool uncovers a vulnerability.

The same sorts of SAST-generated metrics that can be collected during the develop and build phases should be collected during SAST in the test and release phases. The ratio of the incidence of inter-module SAST violations to the incidence of per-module violations can be revealing. A high ratio may indicate that developers are not communicating how their modules are intended to be used. This in turn may suggest a poorly defined architecture or inadequate module documentation standards. Whatever the reason, the cause needs to be identified and addressed. If these metrics are being collected, the DT&E team is ultimately responsible for ensuring that they are used.

Table 5-15 describes scope of the SAST Test/Release Phase DT role by T&E activity.

Table 5-15. SAST Test/Release Phase - T&E Activities and DT Roles

T&E Activity	Description of Developmental Testing Role
Unit Test	N/A
Integration Test	• Determine adequacy of: ○ SAST tooling ○ SAST coverage • Agree on triggers for SAST • Agree upon details of what SAST output is reported as well as how, when, and in what format • Review SAST reports for areas of concern
System Test	• Determine adequacy of: ○ SAST tooling ○ SAST coverage • Agree on triggers for SAST • Agree upon details of what SAST output is reported as well as how, when, and in what format • Review SAST reports for areas of concern
Cyber Test	• Determine adequacy of: ○ SAST tooling ○ SAST coverage • Agree on triggers for SAST • Agree upon details of what SAST output is reported as well as how, when, and in what format • Review SAST reports for areas of concern

Lessons Learned

Information will be added to the guidebook based on stakeholder feedback.

5.4.2 System Integration Testing

In the system integration testing activity, the development team tests sets of modules. System integration testing follows unit testing, where individual modules are tested. System integration testing determines whether modules interact and cooperate as defined in the system architecture.

Note: System integration testing is sometimes used to mean the testing of all subsystems integrated into a functioning system.

Because system integration testing requires knowledge of the system architecture, it demands a skill set beyond that of a module developer. An integrated set of modules usually emphasizes some aspect of an architecture. It might implement a layer of a software stack, such as the network layer of the Open Systems Interconnection model as defined in ISO/IEC 7498-1:1994, "Information technology — Open Systems Interconnection — Basic Reference Model: The Basic Model." It might implement some mission-specific function or application-specific protocol. Testing the set requires knowledge of some domain. This in turn requires knowledge of domain-specific testing techniques. Although developers often contribute to system integration testing, they need testing skills.

The system test plan, developed during the plan phase, should specify how to perform system integration testing. It describes increasingly larger sets of modules to test. This successive approach is based on the belief that debugging a small set of modules is easier than debugging a large set because a small set has fewer inter-module interactions. Once a small set has been debugged and the development team has confidence in the interactions among modules in that set, more modules can be included in the set; testers then only must focus on the new interactions. This limiting approach isolates the places where bugs are likely to be found. The consequent reduction in overall effort helps minimize the time spent during testing, which is important in keeping DevSecOps iterations short.

Objective

In system integration testing, the DT&E team's objective is to supply evidence that module combinations provide expected capabilities. A unit test focuses on an individual module and assumes that other modules with which the module communicates behave as advertised. A system integration test confirms or rejects this assumption. It replaces module stubs with calls to, or communications with, actual modules. Now a test will determine whether a called module provides the expected results; whether it reports errors correctly; and whether it recovers gracefully from failure.

System integration testing incrementally bridges between unit tests and tests of the complete system. The DT&E team typically creates an integration test suite in stages. The initial stages focus on a few modules. These modules do not need to be the most critical. The objective of system integration testing is to combine modules in succession to simplify debugging by concentrating on a few capabilities at a time. The DT&E team's objective is to determine module combinations that are easy to analyze. Once the DT&E team has tested small sets of modules together, the team can add more and more modules, growing the set until it constitutes an entire subsystem.

As testers increase the size and complexity of integration tests, they need a test environment that comes closer to the production environment. Their objective is to document and implement this environment. Such an environment helps testers demonstrate that the entire system will behave as expected once it reaches production. The environment will also be useful for other types of testing described in this section.

System integration testing has some overlap with SAST (discussed in Section 5.4.1) and DAST (discussed in Section 5.4.5). Repeating SAST is advisable. It may detect differences between the interfaces of stubs used in unit testing and those of the actual modules. SAST and DAST should be applied to the test environment as well, if for no other reason than to dissuade bad actors from interfering with tests. The likelihood of cybersecurity incidents increases with system complexity, so early scanning for problems allows earlier remediation.

Why It Matters to the DT&E Community

During system integration testing, the DT&E team works collaboratively. Unit testing is more of an individual activity; each developer concentrates on creating a test suite for a particular module and often works independently of other developers. In system integration testing, the team makes decisions jointly. One of the first decisions, likely made during the plan or develop phase, pertains to the architecture of the environment in which tests are to be conducted. The test environment is more complex than implementing drivers and stubs. Depending on the modules' functionality, the DT&E team may have to simulate a network; stand up a web server; provide a primitive implementation of both ends of a software stack; or emulate some aspect of the physical world. Creating a realistic test environment for a system integration test can be an act of design. In DevSecOps, the DT&E team must consider whether the test environment's design can be implemented in time to deliver the system according to the project's schedule. The difficulty of implementing an overly complex design may require sacrificing some amount of realism.

The DT&E team must then implement this design, ideally before the test phase begins. Part of the implementation will involve writing driver and stub components—the integrated set of modules is still not a complete system, and the environment must simulate absent modules. These components must interact to create an environment that mimics some aspect of the production environment. Although some not-insignificant effort may be necessary to stand up this environment, it can generally be reused in larger system integration tests, as well as in subsequent DevSecOps iterations.

The test environment will likely be implemented using container technology. Containers help guarantee that each test begins in the same environment, with the same set of starting conditions. Containers also isolate tests and prevent adverse effects, such as overwriting or removing files in the software factory. Containers can be used to control what processes are competing for resources and are therefore useful when measuring performance. The DT&E team will need members who are proficient in configuring and using containers.

The DT&E team will need to specify instructions to build the modules into an executable image; transfer that image into a container; and execute the image within the container. These instructions will be written using formal, executable languages that can be executed automatically by the software factory. The DT&E team must also define the triggers for executing these instructions. The simplest, most straightforward trigger is to execute the instructions whenever a module that is part of the integrated set is checked into the software repository. If too many test suite executions are triggered, the DT&E team should consider other strategies: periodically (e.g., nightly); check-in of some subset; or on request.

The DT&E team is responsible for defining the test data inputs. The methods for defining the inputs depend on the modules being tested. Real-world data sets may be available, other data sets will require data to be generated. The DT&E team must select an appropriate method, which may require domain expertise. Sometimes data from a unit test suite can be reused in system integration testing. If the set of modules has a “driver” module, the data for that module’s unit test demonstrates that it correctly interacts with all other modules in the set. If the data turns out to be insufficient for a complete test, the DT&E team should consider adding tests using any new data to the driver module’s unit test suite.

The DT&E team usually configures a software factory to automate system integration test execution. Some tests that involve UI modules may require humans to enter values or observe results. The DT&E team must identify these tests and determine how they will be conducted.

The DT&E team is responsible for determining what happens when a test finishes. During the plan phase, project management staff must determine how much visibility project members should have into test results; i.e., who is to be notified of success and failure. If a test succeeds, the simplest action is to notify the persons responsible for triggering the test. A broader action is to notify the persons who contribute to the test: persons who developed any of the integrated modules; persons who develop the testing environment; and persons who develop the test execution scripts. An even broader action is to notify project management staff, giving them information on testing progress.

If a test fails, the results of a system integration test do not generally indicate why the test failed beyond the immediate fault: an incorrect output, an exception, a deadlock. These results say nothing about what led to the fault, so all developers of modules involved in the test should be notified. Because implementing the test environment can be nontrivial, the test environment may have flaws of its own, and until these flaws are discovered and corrected, the test environment's developers should be notified too.

Some tests do not simply succeed or fail. For example, performance-based tests generate measurement data. Part of DT&E involves evaluating this kind of data to provide insights into the production environment—for example, whether predictions on hardware capabilities are accurate. The DT&E team must identify who needs to review tests that generate analytical results.

Table 5-16 describes scope of the System Integration Testing DT role by T&E activity.

Table 5-16. System Integration Testing - T&E Activities and DT Roles

T&E Activity	Description of Developmental Testing Role
Unit Test	N/A
Integration Test	• Verify proper implementation of the test environment • Perform manual system integration tests • Review test results
System Test	N/A
Cyber Test	• Ensure the test suite includes cyber-oriented tests • Run SAST tools on modules and the test environment • Run DAST tools on modules executing in the test environment

Lessons Learned

Information will be added to the guidebook based on stakeholder feedback.

5.4.3 Regression Testing in the Test/Release Phase

Regression testing, as discussed in Section 5.3.4, is an activity that verifies that an iteration's changes do not affect existing capabilities. Most changes are made to add capabilities or to fix bugs in existing capabilities. Re-executing tests from previous iterations is an accepted practice to ensure two things: that existing capabilities still behave as expected and that changes did not reintroduce old bugs (assuming tests were introduced in response to detected bugs, a best practice of developmental testing).

In the test and release phases, regression testing focuses more on system tests than on module tests. Regression tests on modules should be conducted during the develop and build phases (discussed in Section 5.3.4) and do not have to be repeated. Regression tests on sets of modules and on subsystems need to be executed in the test and release phases only insofar as they exercise capabilities not covered in system regression tests.

As in the develop and build phases, the development team should strive to automate regression tests to the maximum extent practical. Automation is even more important in the test and release phases because the amount of time and effort expended to build a system and execute tests is greater.

A new version of a system may remove a capability or may change system input and output formats. These types of changes can make existing regression tests obsolete. Running some regression tests may not only be unnecessary but counterproductive, falsely indicating failure.

Objective

The objective of the regression testing activity is to ensure the integrity of previously completed capabilities. From a practical standpoint, the objective is to spend the least amount of time and effort doing so; automating as much of the regression testing as practical so developers and testers do not have to prepare or initiate tests. The development team is responsible for automating regression testing by programing the software factory to initiate regression tests in response to predetermined triggers. The nature of these triggers is specified as part of the overall testing process. If the software factory has sufficient resources, it could execute multiple types of testing simultaneously.

The DT&E team is responsible for reviewing the development team's work. The DT&E team should verify that the regression testing strategy makes sense; the automated regression testing is properly implemented; the regression testing occurs during the test and release phases; and the development team responds to the results of regression tests. The DT&E team uses the results of regression testing to inform the PM as to whether a system is ready to be delivered.

Why It Matters to the DT&E Community

The DT&E team's role during the regression testing activity is mainly one of review. The team reviews system logs to verify that regression testing has taken place; ensures that issues are created in response to failed tests; and informs the PM about how the results of regression testing affect the remainder of the current DevSecOps iteration. Depending on how regression testing fits into the overall testing process, the effect may be to delay subsequent testing activities or to combine regression test results with other test results. The latter may provide more information on the time and effort needed to complete the test phase.

Regression testing planning should address which regression tests are important based on the capabilities added, modified, or removed. The DT&E team works with the development team to ensure that the software factory properly automates regression tests the next time they are executed.

Table 5-17 describes scope of the Regression Testing Test/Release Phase DT role by T&E activity.

Table 5-17. Regression Testing Test/Release Phase - T&E Activities and DT Roles

T&E Activity	Description of Developmental Testing Role
Unit Test	N/A
Integration Test	• Verify that integration regression tests have successfully completed • Maintain measurements on integration regression test results to use for planning time and effort during subsequent iterations • Review integration tests developed during the current iteration and plan to incorporate them as regression tests in subsequent iterations
System Test	• Verify that system regression tests have successfully completed • Maintain measurements on system regression test results to use for planning time and effort during subsequent iterations • Review system tests developed during the current iteration and plan to incorporate them as regression tests in subsequent iterations
Cyber Test	Ensure that cyber-related regression tests have been executed

Lessons Learned

Information will be added to the guidebook based on stakeholder feedback.

5.4.4 Interoperability Testing in the Test/Release Phase

DoDI 8330.01 defines interoperability as "The ability of systems, units, or forces to provide data, information, materiel, and services to, and accept the same from, other systems, units, or forces, and to use the data, information, materiel, and services exchanged to enable them to operate effectively together." During the interoperability testing activity, the DT&E team works to verify that the system under development has been adequately tested regarding interoperability.

The systems, units, and forces with which a system must interact (and hence be interoperable), along with the data, information, and services it must exchange, should be understood by the end of the plan phase. The requirements document should specify a system's input and output formats, along with its APIs. Formats and APIs can be justified based on how they promote interoperability. The need to conform to an existing system's formats is one justification. A system that might serve as a data broker should be designed to maximize interoperability with future systems.

Interoperability includes the platforms on which a system might operate or with which it might communicate. If a system is to be hosted on a Microsoft Windows platform, its requirements should state which version of Windows is to be employed and tests required to ensure correct operation. If a system has a web-based interface, its requirements should specify the browsers to be supported (including, possibly, mobile devices), and interoperability tests should verify that a given page displays properly on each browser. This kind of testing, it should be noted, requires visual inspection and may not be fully automatable.

Interoperability can be specified through user stories. An interoperability-focused user story should describe a scenario in which the system being tested needs to communicate with another system to obtain the data to answer a query or to perform some tasks. From the operator's perspective, the use of another system may be explicit or transparent. If an operator must submit credentials and authorizations to the other system, the operator will need to know which credentials to supply. An interoperability test should determine whether the operator is prepared to provide credentials and authorization. Or, if access to another system is entirely in the background, the operator may be completely unaware that the other system is being used. A user story should focus on an operator's unimpeded experience (normal operations) of using the SUT, and the interoperability test should guarantee it. In stress testing, it may be necessary to assess a system's behavior when another system is inaccessible and whether the message it presents to the user is appropriately informative, which might mean revealing a failure to communicate.

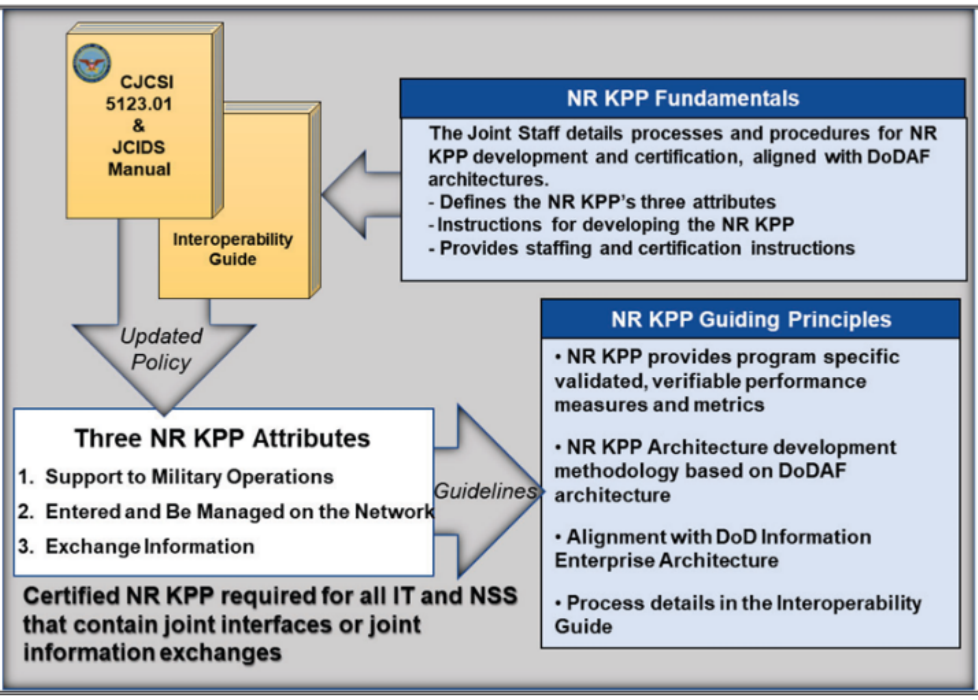
Objective

The objective of the DoDI 8330.01 interoperability testing activity is to ensure that developmental software systems work as part of a network of interfacing systems. The DT&E team may work with an interoperability test team devoted to interoperability testing as well as representatives from the development team. The interoperability test team's objective is to ensure that interoperability tests are properly conducted during the test phase.

Before interoperability T&E during the test phase, the interoperability test team works with the T&E WIPT and the PM to plan interoperability testing. The DT&E team also provides input to the TES to provide performance metrics and a test methodology to address evaluation of data transfer and use among the interfaced elements of the system and external systems. The interoperability test team works to confirm that the environment in which interoperability testing will occur, if distinct from the production environment, is functionally representative enough of the operational environment to adequately allow the interfaces to be tested and evaluated for joint interoperation. During the initial DevSecOps develop and build phases, the DT&E team is responsible for providing input on the selection, integration, and sufficiency of automated or manual interoperability, as well as software test tools required to provide data through the iterative testing incorporated in the DevSecOps software factory pipeline.

If this integration is left until OT&E, the touchpoints will not be there to produce the required supporting data for evaluation. This data gap puts an onus on the T&E WIPT and the PM during the plan phase to ensure development of Net Ready Key Performance Parameters (NR KPPs),[10] through interaction with the Joint Staff and JITC, and inclusion of the NR KPPs in the planning for design of the development environment and the system's TES. These requirements and performance metrics must then feed into development of the automated testing built into the pipeline. Additionally, given the frequency of DevSecOps iterations, no practical way exists to tie up an interoperability T&E team continuously to revalidate each backlog as it moves into production.

Figure 5-2 provides a chart of relevant source documents, NR KPP attributes, NR KPP fundamentals, and guidelines for the interoperability certification process. Interoperability test team members should refer to these and any additional Service- or agency-specific policies and guidance.



NR KPP Fundamentals

The Joint Staff details processes and procedures for NR KPP development and certification, aligned with DoDAF architectures.
- Defines the NR KPP's three attributes
- Instructions for developing the NR KPP
- Provides staffing and certification instructions

NR KPP Guiding Principles

- NR KPP provides program specific validated, verifiable performance measures and metrics
- NR KPP Architecture development methodology based on DoDAF architecture
- Alignment with DoD Information Enterprise Architecture
- Process details in the Interoperability Guide

Three NR KPP Attributes

1. Support to Military Operations
2. Entered and Be Managed on the Network
3. Exchange Information

Certified NR KPP required for all IT and NSS that contain joint interfaces or joint information exchanges

LEGEND:

&	and	JCIDS	Joint Capabilities Integration Development System
CJCSI	Chairman of the Joint Chiefs of Staff Instruction	KPP	Key Performance Parameter
DoD	Department of Defense	NR	Net-Ready
DoDAF	Department of Defense Architecture Framework	NR KPP	Net-Ready Key Performance Parameter
IT	Information Technology	NSS	National Security Systems

Department of Defense Interoperability Process Guide, Version 3.0, p. 10

Figure 5-2. Policies Governing Requirements Preparation

[10] See the NR KPP definition in the DAU Glossary at https://www.dau.edu/glossary/net-ready-key-performance-parameter.

Technically, interoperability T&E is considered operational testing; however, in a CI/CD construct, where the development environment is functionally equivalent to the operational environment using virtualization and containerization, interoperability testing should be accomplished during developmental testing if possible, where course correction can still be made before deployment.

Why It Matters to the DT&E Community

The complexity and challenges of meeting joint interoperability testing and certification requirements while employing a DevSecOps development methodology largely mirror those of cyber testing and the RMF ATO. Under the classic phases of acquisition processes, these test events traditionally provide a static view of operationally deployed software rather than a continuously evolving baseline.

The interoperability T&E process is addressed in Section 5.5.2, which discusses interoperability testing in the deliver phase. In the test phase, the DT&E team's important consideration is the necessary amount of interoperability testing required considering the objectives of a DevSecOps cycle. In early cycles, when the objective is to produce an MVP, the number of systems with which the system must interact and the number of platforms on which it must operate may be fewer than those of the MVCR. The DT&E team should ensure that the test plan has been achieved insofar as actual interoperability concerns apply. In a later cycle whose objective is to deliver a small number of changes, or even a single change, interoperability may not be of concern at all: The DT&E team may determine that a change fixing a core functionality bug but not affecting external interfaces does not merit the time and effort to retest interoperability.

Table 5-18 describes scope of the Interoperability Testing Test/Release Phase DT role by T&E activity.

Table 5-18. Interoperability Testing Test/Release Phase - T&E Activities and DT Activities

T&E Activity	Description of Developmental Testing Activities
Unit Test	N/A
Integration Test	Integration tests can be used as a proxy for interoperability testing when the development environment is sufficiently representative of the operational environments and any interfaced systems requiring data exchange.
System Test	The DT&E team needs to assess whether the system test environment is sufficiently representative of the production environment to determine whether testing can verify that interoperability requirements are satisfied.
Cyber Test	Selected cyber tests can be combined with interoperability tests to provide artifacts usable for both the RMF and joint interoperability certification when the development environment is sufficiently representative of the operational environments and any interfaced systems requiring data exchange.

Lessons Learned

Information will be added to the guidebook based on stakeholder feedback.

5.4.5 Dynamic Application Security Testing

DAST evaluates software (not source code) as it is executing for security issues that could affect the system. Like SAST, it should be automated to the extent possible and triggered automatically after code commits. DAST, however, uses a different set of tools and skillsets to assess system quality from a different perspective—in the dynamic case, from the perspective of live, running modules in a "black box" (i.e., without source code access) paradigm.

Objective

DAST can determine issues relevant to running software (e.g., CPU utilization, memory utilization, performance) that are not covered by SAST. DAST can be performed at unit, integration, and system levels of testing. Emergent properties of systems are often easier to identify via DAST than via SAST.

Why It Matters to the DT&E Community

DAST can find issues in compiled modules while they are executing so issues can be identified and corrected quickly and at lower cost than if they are found later. The MVP must have DAST tooling as part of the software factory.

Testers should already have a DAST test strategy that they developed during the plan phase. Testers should provide oversight and agree with developers on the test strategy. Testing should prioritize sections of the application with greater mission impact or cost of failure. An analysis of the system architecture can be used to determine these areas for enhanced testing. Developers should document the DAST tooling and sources of DAST test cases (e.g., libraries of DAST scripts), as well as where and how the test cases and results should be stored and shared with the government. Testers should provide oversight to ensure that tooling and test sources are sufficient to assess software quality. Additionally, processes should be in place to notify developers of any test failures in an issue tracking system. Testers should ensure that the automated DAST tool is executing when expected (e.g., during code check-in, upon completion of SAST) during the test and release phase and that manual DAST is also performed when expected. Processes should ensure that failures found later in the development process—that could have been identified through a dynamic test—result in the creation of a regression test case if possible. Testers should ensure that DAST or IAST is performed successfully because system changes must be promoted from the test environment to the pre-production environment. DAST output includes a weakness report with recommended mitigations.

Reductions over time of specific CWE findings may indicate increased code quality.

Successful completion of DAST is required for promotion from the test environment to the pre-production environment.

Table 5-19 describes scope of the DAST DT role by T&E activity.

Table 5-19. DAST - T&E Activities and DT Roles

T&E Activity	Description of Developmental Testing Role
Unit Test	N/A
Integration Test	N/A
System Test	N/A
Cyber Test	• Determine adequacy of: ○ DAST tooling ○ DAST coverage • Agree on triggers for DAST • Agree upon details of what DAST output is reported as well as how, when, and in what format • Review DAST reports for areas of concern

Lessons Learned

Information will be added to the guidebook based on stakeholder feedback.

5.4.6 Application Programming Interface Testing

APIs are the glue that holds systems together. These interfaces are critical to system functionality and security, and problems with an API can have disastrous consequences for system behavior. During the API testing activity, test cases that exercise APIs are executed, and their results are reviewed.

Objective

API testing ensures that system intraconnectivity (connectivity of subsystems) and interconnectivity behave as expected. API issues can be architecturally cross-cutting; a change to an API can require subsequent changes across large portions of system architectures, so it is critical to get APIs right from the onset of the system life cycle.

Why It Matters to the DT&E Community

API testing covers much of the non-human machine interface portion of a system's attack surface. APIs can be entry points for attackers and targets for distributed denial-of-service attacks, and they are make-or-break for ensuring system interoperability. Therefore, APIs should be given extra scrutiny throughout the system life cycle. Testing should be thorough and performed with respect to the impact of API failure on the ability of the system to perform its mission or the impact on the mission of other connected systems that may be exposed by a failure of the SUT.

Early in the development process, APIs should be standardized with documented behavior so developers can independently develop subsystem components that will interact properly and securely with each other and with external systems. Security attributes should be part of this documentation and should be built in during development rather than "bolted on" later in the development process. This documentation of expected API behavior should include preconditions and postconditions for proper API behavior. That is, it should document attributes such as assumed input ranges and formats, as well as the guaranteed output ranges and formats. Where applicable, invariants of the subsystem should also be documented alongside the preconditions and postconditions. Testers should ensure documentation completeness because it is an input to the proper testing of APIs.

Testing of APIs involves the testing of the interfaces as well as how those APIs were implemented by the developers. The testing strategy should use equivalence partitioning and boundary analysis to effectively cover the state space of possible tests. Although formal verification may not be necessary, the proper and complete documentation of preconditions, postconditions, and invariants is critical to formal proofs of correctness; serving as a "contract" by which developers produce code; see Meyer, Bertrand Website. OpenAPI is an example of how to write such a specification, though it does not fully cover invariants; see OpenAPI Initiative Website. Particularly early in the development process, subsystems critical to security or functionality of high-impact missions should be formally verified for correctness at the API level. Testers should ensure that key subsystems go through a formal verification process where warranted.

Modeling makes use of APIs to assess system behavior without the need for a fully constructed system. Modeling is helpful to identify issues before system development. Because a model is simpler than the deployed system, it can also help to understand aspects of the deployed system without having to interact with it or develop a complete duplicate. The accuracy of the model is dependent on the realism of the modeled system and environment. The model should be kept current during the DevSecOps development process.

Although some types of API issues can be tested statically, others are best tested dynamically. At the very least, testing at the integration and system levels helps to detect and mitigate any emergent issues related to APIs. Testing should be automated to the extent possible, built into the CI/CD pipeline to be automatically triggered—particularly when code is integrated and promoted to the next environment. Several security tests for APIs can be found in open sources of best practices such as OWASP; e.g., those identified on the OWASP Top 10 API Security Risks – 2023 Website. The entire catalog of API security risks should be considered, and government testers should ensure that relevant API tests are performed, reported, and mitigated.

Government testers should ensure that API tests, as well as security tests, are sufficient to not impair functionality.

Table 5-20 describes scope of the API Testing DT role by T&E activity.

Table 5-20. API Testing - T&E Activities and DT Roles

T&E Activity	Description of Developmental Testing Role
Unit Test	• Developers document API preconditions, postconditions, and invariants. • Developers ensure—and testers confirm—that unit tests are sufficient for system functionality and security, validating input and guaranteeing proper output format. • Testers also ensure that documentation is accurate and sufficient to promote quality code.
Integration Test	• Developers agree with testers on when and which tests will be conducted for system integration; they also agree on requirements for the realism of stubbed services used in testing. • Testers ensure proper execution and reporting of tests, and they ensure that issues are appropriately mitigated.
System Test	• Developers agree with testers on areas that require formal verification. • Developers agree with testers on dynamic tests of system APIs. • Post-testing, testers ensure that tests were conducted as expected; results ensure confidence in system functionality and security; and any issues are communicated to stakeholders.
Cyber Test	• Developers agree with testers on when and which tests will be conducted for evaluating functional cyber performance and identifying vulnerabilities needing to be fixed. • Testers ensure that tests verify that APIs abide by zero trust architecture requirements. • Testers ensure the proper execution and reporting of tests, and they ensure that issues are appropriately mitigated and communicated to stakeholders.

Lessons Learned

Information will be added to the guidebook based on stakeholder feedback.

5.4.7 Interactive Application Security Testing

IAST is a type of dynamic security testing that requires interacting with running software for the purpose of evaluating security. It is more sophisticated than DAST testing. IAST combines elements of DAST and SAST to instrument sensors into a running application in a way that enables identification and understanding of weaknesses.

Objective

IAST tests a running system like DAST, but its integration with sensors makes IAST a good choice for situations in which determining root causes of issues is not straightforward. However, the overhead of installing and monitoring sensors is a drawback compared with traditional DAST. IAST may use human testers or automated interactivity for sophisticated levels of application interaction. IAST coverage includes only parts of the application that are interactively tested by a human or automated interface; it does not cover entire codebases or modules.

Why It Matters to the DT&E Community

IAST is interactive and can test for issues that may not be easily discoverable through other forms of SAST and DAST. Through instrumenting code with sensors, IAST can assist in identifying and locating code issues in a way that other test methods cannot. However, IAST does not have the same level of coverage, so the scope of IAST must be focused on the elements of system behavior that could have the greatest mission impact. IAST may also require human involvement for dynamic interaction with the running system, further limiting how and where it is applied. Testers should ensure that DAST or IAST is completed as expected and that tests are passed because system changes must be promoted from the test environment to the pre-production environment. Testers should assess whether the coverage of the IAST is sufficient to identify security risks for risk management decisions.

Table 5-21 describes scope of the IAST DT role by T&E activity.

Table 5-21. IAST - T&E Activities and DT Roles

T&E Activity	Description of Developmental Testing Role
Unit Test	N/A
Integration Test	N/A
System Test	N/A
Cyber Test	• Determine adequacy of: o IAST tooling o IAST coverage

T&E Activity	Description of Developmental Testing Role
	• Agree on triggers for IAST • Agree upon details of what IAST output is reported as well as how, when, and in what format • Review IAST reports for areas of concern

Lessons Learned

Information will be added to the guidebook based on stakeholder feedback.

5.4.8 Human Systems Integration Testing

As defined on page 2 of the HSI Guidebook, "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 TSP [total system performance] and minimizes TOC [total ownership costs]. The PM staff, specifically the Lead Systems Engineer with HSI practitioner support, analyze requirements to optimize TSP and determine the most effective, efficient, and affordable design. The PM staff should use the analysis of the HSI domains to help determine and investigate the S&T gaps to address all aspects of the system (hardware, software, and human)."

As outlined in DoDI 5000.95, "Human Systems Integration in Defense Acquisition," the DoD Component capability developer or PM will formulate a comprehensive HSI program using an appropriate strategy to ensure HSI-related and human performance requirements are achieved in accordance with DoDI 5000.02 and including, but not limited to:

- Usability and other user testing to support and inform human and machine interface analysis under operational conditions.
- HSI risk management maintained through testing.

Objective

The objective of HSI testing is to ensure that human performance is optimized to increase total system performance and minimize total ownership cost. Incorporating HSI early in system design promotes a more successful and effective transition of capability to the warfighter.

T&E should use findings, analyses, evaluations, design reviews, modeling, simulations, demonstrations, and other early engineering tests when planning and conducting later tests in the life cycle.

Incorporating HSI as a functional competency within the product team will ensure that user needs are included in design, development, and integration. HSI user-centered design activities should be integrated with DevSecOps development and align user-centered design with iterative development and test.

Why It Matters to the DT&E Community

Section 4.4.2.5 of the HSI Guidebook summarizes the HSI impacts on DT&E:

- Human factors engineering (HFE) and the evaluation of all human-machine interfaces and user processes should be integrated into engineering design and development tests, contractor demonstrations, flight tests, acceptance tests, and other development tests.
- T&E and HFE should test compliance with human-centered design requirements as early as possible.
- T&E should use findings, analyses, evaluations, design reviews, modeling, simulations, demonstrations, and other early engineering tests when planning and conducting later tests.
- Test planning should consider DT&E data needed for OT&E.

A primary benefit provided by the HSI community of interest is the importance placed on incorporating user feedback into system design, affecting iterative development and testing.

In Agile/DevSecOps, it would be preferable to have human factors/UX validated designs before developing product features during an Agile development sprint. Correcting usability issues early on costs less than correcting them in later phases of the product development cycle. HSI has the potential to add value to DevSecOps processes by validating designs before deploying them into production, which not only delivers better products but also reduces costs and the amount of rework in future Agile sprints.

Table 5-22 describes scope of the HSI Testing DT role by T&E activity.

Table 5-22. HSI Testing - T&E Activities and DT Roles

T&E Activity	Description of Developmental Testing Role
Unit Test	N/A
Integration Test	N/A
System Test	N/A
Cyber Test	N/A

Lessons Learned

Information will be added to the guidebook based on stakeholder feedback.

5.4.9 Mission-Oriented DT&E

Mission-oriented DT&E is the activity of conducting tests based on mission threads. A mission thread derives from analyzing the steps of a unit's mission, which are described in sources such as the Joint Chiefs of Staff Universal Joint Task List Website. Mission-oriented developmental testing is sometimes referred to as end-to-end testing because it follows each step of a mission from start to finish. Each test focuses on a mission, and not on a system used to support a mission. A test, however, will not succeed if a system malfunctions.

A mission test is driven by the mission's concept of operations (CONOPS). The CONOPS defines the types of data needed as inputs; the types of operators and users involved in the mission; the steps involved in performing a mission; and the expected results of the mission. The test definition, implementation, and performance must conform to the CONOPS.

Some mission testing necessarily occurs during operations rather than in development because of the difficulty of simulating an operational environment. Consider a mission involving target acquisition, projectile aiming and launch, and damage assessment. It uses multiple systems, requires specialized hardware, and necessitates extensive operator training. The time, effort, and cost to build a facsimile of the operational environment may be too large to justify slowing the pace of a DevSecOps development cycle. Now consider a mission that arranges a transfer of funds using a financial management system. It probably requires only a single system, and the environment in which that system operates is akin to an office environment. The mission's efficacy can perhaps be tested and evaluated during development, thereby providing feedback on how the financial management system contributes to mission success.

Mission-oriented developmental testing can capitalize on the iterative nature of DevSecOps. If the effort to produce a realistic environment is judged to be too great during the first iteration, the pre-production environment used to test a system before deployment may be reusable for developmental testing in subsequent iterations.

Even with a suitable environment, mission-oriented developmental testing usually requires the environment to be populated with nonproduction data. Privacy dangers exist in copying production data from one environment to another and on using production data in an environment that has not been hardened. (Hardening is possible but can be costly to verify.) Some projects develop systems on unclassified platforms and deliver them to production environments on classified platforms; this is another reason why production data may not be useful in the mission-oriented developmental testing activity.

Mission-oriented testing and system testing are fundamentally different. In most kinds of system testing, tests either succeed or fail given certain inputs and starting conditions; a system either produces expected outputs or it does not. System tests derive from system requirements, so even tests that do not have binary outcomes, such as performance tests (discussed in Section 5.5.3) and stress tests, are written from a system rather than an operational perspective. A mission-oriented test derives from a list of mission-essential tasks. The test measures an operator's ability to perform each task in the real world. The measures are not generally expressed as success or failure. Ability should be measured based on quantitative criteria; see the DOT&E Quantitative Mission-focused Measures – Guidance, Version 3.0, for additional information. Continuous criteria are better than discrete criteria; measurements of the real world are continuous quantities. The criteria should comprehensively cover the reasons for procuring a system in support of a mission. They should not, however, cover more than is necessary: the fewer measurements to take, the better. The criteria should be used in concert with statistical test design methodologies, which means that devising measurements is intricately tied to designing statistical tests; see the DOT&E Mission Focused Evaluation – Guidance, Version 3.0, for additional information.

A program's testing strategy document should define these criteria and how they can be used to evaluate mission performance. System requirements should, ideally, specify the system characteristics most critical for providing an effective capability to perform a mission, and these characteristics should relate to the contents of the testing strategy; for example, requirements on UI design should be traceable to the measures for an operator to use a system in support of a mission, such as time spent interacting with a system to perform a task.

Mission-oriented developmental testing is not a substitute for T&E during operations. Mission-oriented developmental testing compares with operational testing as laboratory experiments compare with real-world experiences. A laboratory is a confined and controlled space that, in the interests of measuring specific variables, deliberately omits complexity. Only operational use can fully incorporate and account for complexity.

Objective

The objective of the mission-oriented developmental testing activity is to ensure the capability to conduct missions; i.e., to satisfy a CONOPS. Projects may not have the time or resources to stand up and use an environment during development that can adequately simulate a real-world mission. Mission-oriented testing is a complicated endeavor that requires planning. Relevant metrics, measurements techniques and technologies, and evaluation approaches should be defined by the end of the plan phase, and they should be defined by the program office, not by the development team.

A DT&E team objective is to ensure that mission T&E infrastructure is understood by the end of the release phase. Because infrastructure cannot be counted on until it has been exercised, the DT&E team should be able to report on the infrastructure to the PM. The most effective report would be based on a simulated production environment; its use to perform a mission thread; and a demonstration that measurements can be conducted to obtain the metrics necessary for evaluation.

Mission-oriented developmental testing provides an opportunity to analyze mission cybersecurity as well as system cybersecurity. Analysts can identify when sensitive data is in motion versus at rest, for example, or the times an environment is vulnerable to attacks, such as penetration testing, or the consequences of operators in various roles falling victim to phishing attacks. Another DT&E team objective is to understand the cybersecurity consequences of introducing a new system into a mission (or, in later DevSecOps iterations, of modifying a system's functionality).

Why It Matters to the DT&E Community

During the mission-oriented developmental testing activity, the DT&E team follows the test strategy to envision what mission-oriented operational testing requires and, to the degree possible and practical, to simulate it. If the test strategy does not identify tests to perform or does not identify test variables that apply to a system, the DT&E team is responsible for proposing test strategy revisions. By the time the DT&E team completes its work, there should be clear documentation of:

- Tests and evaluations to perform and the purpose of each with respect to missions.
- Test variables and how to measure them. Some variables can be measured automatically; for these, the documentation should describe the necessary infrastructure. Some variables must be measured manually; the documentation should describe measurement techniques and how they fit into the testing process.
- The meaning of a successfully conducted test. "Successfully conducted" does not mean that a test passes or fails but that the data necessary to analyze the test has been collected.
- The meaning of a successful test. For a mission involving a fire-control system, a successful test might be that enemy fire capability has been reduced by 75 percent. For a mission involving an IT system, a successful test might be that funds were transferred to the correct receiver 100 percent of the time and that the transfer, from initiation to reception, occurred in less than one hour 95 percent of the time.

These items should be traceable to a CONOPS.

Documenting these items before system deployments is important. In the same way that a software error becomes increasingly costly to fix the later it is detected, operational evaluation becomes harder the longer it is postponed. One reason is that systems can assist in measurement by being instrumented to collect data. The absence of instrumentation means operational evaluation must be either performed manually or postponed until a subsequent DevSecOps iteration whose objective is to include instrumentation. The DT&E team is responsible for evaluating whether a system is properly instrumented and for testing whether the instrumentation works properly. Once the team has completed these tasks, a system can be advertised as supporting mission-oriented testing.

If the DT&E team has the time and resources, it may be able to conduct a dry run of mission-oriented operational testing during the mission-oriented developmental testing activity. The team must:

- Specify, configure, and provision the environment in which the test will occur. The environment should simulate the production environment to the degree possible. At a minimum, provisioning the environment means including the system being developed.
- Determine reasonable startup configurations to include determining the state of the system and the environment at the start of a test, and addressing what an operator would see when first interacting with a system to complete a task. The state of the environment can include data sets and sensor values. Populating a database may be necessary. The DT&E team should review user stories (discussed in Section 5.3.5), which may include example values.
- Determine how to perform the test, which includes determining the types of users needed; the equipment the users will need; the expected duration of the test; the training the users will need, etc.
- Select users, time, and place to perform the test.
- Train the users.
- Have the users execute the test, using the system to perform their assigned tasks. The documentation on a successfully conducted test should guide this step. As the test is being performed, measurements must be made. The system may be instrumented to make some of these measurements. The test environment may also be instrumented—to monitor memory use or network traffic, for example. The DT&E team should be prepared to measure variables manually if necessary.
- Analyze the results. The analysis is in terms of the definition of a successful test.

Table 5-23 describes scope of the Mission-Oriented DT&E DT role by T&E activity.

Table 5-23. Mission-Oriented DT&E - T&E Activities and DT Roles

T&E Activity	Description of Developmental Testing Role
Unit Test	N/A

T&E Activity	Description of Developmental Testing Role
Integration Test	N/A
System Test	• Determine how a system can support OT&E • Instrument a system to support OT&E • Test the instrumentation to ensure its correct behavior
Cyber Test	• Perform penetration testing on the test environment during the mission test • Identify opportunities for system compromise during the mission test

Lessons Learned

Information will be added to the guidebook based on stakeholder feedback.

5.4.10 Cyber DT&E

Throughout the test and release phases of DevSecOps, cyber DT&E activities should include strategies that use relevant data from all sources including developer and integrated testing. Several activities, artifacts, and data are used to inform and scope cyber DT&E including MBCRAs, RMF security controls assessments, SCRM, SBOM, security verification scans, software assurance test results, network analysis, and component system-level tests. Testing strategies align to technical requirements associated with security standards, scanning, and measuring cyber resilience.

DoDI 5000.89 requires cyber DT&E throughout development. Cyber DT&E should include events that are coordinated and combative on all mission-critical functions, components, and systems through an iterative process that includes implementing remediations, software security patches, and process mitigations and retesting to verify fixes.

In accordance with the forthcoming Version 3.0 of the DoD Cyber DT&E Guidebook, "the cyber DT&E must be integrated early and consistently through system maturation and keep developmental pace with the SUT. The plan, execute, evaluate, and report phases of cyber DT&E should occur at multiple points during a system life cycle, either chronologically or event driven." Because T&E is a continuum, the cyber DT&E process is iterative, as depicted in Figure 5-3. Cyber DT&E is also recursive, meaning that the system developer should continuously reapply the cyber DT&E process throughout the DevSecOps life cycle.

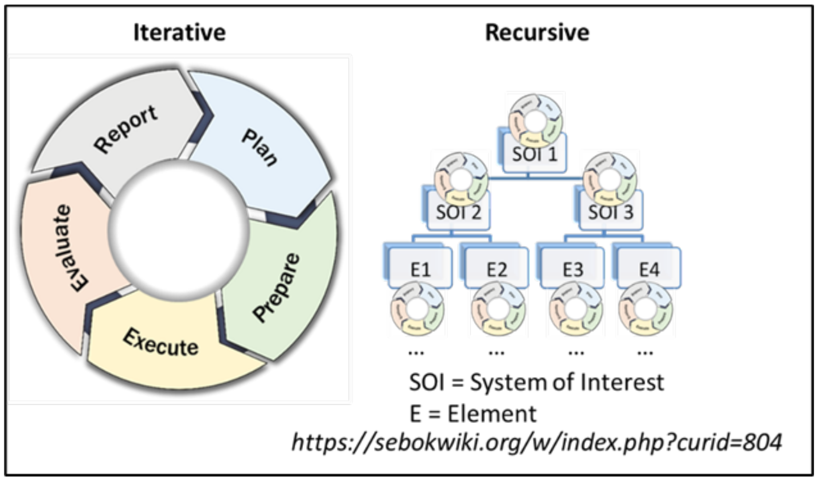


Figure 5-3. Cyber DT&E Continuum

More information on cyber DT&E will be available in Paragraph 3.2. of the forthcoming DODM 5000.UY and in Section 2.1 of the forthcoming Version 3.0 of the DoD Cyber DT&E Guidebook.

Objective

The objective of cyber DT&E is to provide program engineers and decision makers with information to measure progress, identify problems, characterize system capabilities and limitations, and manage technical and programmatic risks. Cyber DT&E supports designing and executing iterative system developer contractor and government cyber DT&E and is an iterative and recursive evaluation of the system's cybersecurity and cyber resilience performance requirements in support of assigned missions. Early detection of cybersecurity and cyber resilience vulnerabilities enables early remediation and reduces the impact on cost, schedule, and performance attributable to late discovery of system performance issues associated with cyber deficiencies. Cyber DT&E should:

- Verify achievement of CTPs, confirm the ability to achieve key performance parameters, and assess progress toward achieving critical operational issues.
- Assess the system's ability to achieve the thresholds prescribed in the capabilities documents for functional cyber performance requirements, if any.
- Provide data to the PM to enable root cause determination and to identify corrective actions.
- Include T&E activities to detect cybersecurity vulnerabilities within custom and commodity hardware and software.
- Stress the system within the intended operationally relevant mission environment.

- Support security control assessment for the RMF assessment and authorization process.

Why It Matters to the DT&E Community

Cyber DT&E provides feedback for designs, notifies system development, and informs on progress toward technical performance and specifications. Cyber DT&E also assists in mitigating system vulnerabilities that have an impact on the operational resilience of capabilities during the life cycle process. The deficiencies should address protection, mitigation, and security. Testing of capabilities such as continuous monitoring, incident management, penetration testing, recoverability, and sustainability during DT&E should be provided as well.

Table 5-24 describes scope of the Cyber DT&E DT role by T&E activity.

Table 5-24. Cyber DT&E - T&E Activities and DT Roles

T&E Activity	Description of Developmental Testing Role
Unit Test	N/A
Integration Test	N/A
System Test	N/A
Cyber Test	Discover issues that are related to the cyber resilience capabilities from cyber threats

Lessons Learned

Information will be added to the guidebook based on stakeholder feedback.

5.5 Deliver Phase

5.5.1 Regression Testing in the Develop Phase

Regression testing, as discussed in Sections 5.3.4 and 5.4.3, is the activity of ensuring that changes made in a DevSecOps iteration do not alter previously completed capabilities. In earlier phases, regression testing usually takes place in a software factory. In the deliver phase, regression testing is done in a pre-production environment before system deployment and operation. Although the development team strives to make the test environment in its software factory as close to the production environment as possible, it seldom achieves perfect fidelity. The pre-production environment may not be perfect either, but it is likely to be closer to that of the production environment. Most obviously, data may be available in the pre-production environment that cannot be used for testing in a software factory. A system developed on an unclassified network might be delivered to secret and top secret networks, and the PM may want to test the system using classified data before approving its operational use. More mundanely, final testing should be performed on a platform that is as close as possible to the hardware and software configuration used in actual operation. This configuration is more likely to exist in the pre-production environment than in the software factory.

Objective

No matter which phase of DevSecOps in which regression testing occurs, the overall objective remains the same: to ensure the integrity of previously completed capabilities. A secondary objective of regression testing is to execute it quickly and efficiently, so it does not interfere with the rapid pace of DevSecOps. Achieving this objective in the deliver phase may not be easy. In previous phases, the software factory automates most, if not all, of regression testing. Replicating the software factory in a pre-production environment may be infeasible because of the licensing restrictions and approvals needed to install software on classified fabrics.

Determining how to perform regression testing in the deliver phase must take place before the phase begins so that regression testing can be completed quickly. Regression testing should still be automated, even without the support of the software factory, and may require a small development project. Plans must also be in place for providing feedback if a regression test fails. This feedback may concern hardware, software, and data not available in the software factory, and the means to transmit it must be carefully considered.

Why It Matters to the DT&E Community

The DT&E team's role in regression testing planning and execution during the deliver phase considers responsibilities, how to automate regression testing; how to set up the pre-production environment for regression testing; how to obtain test data; and how to submit feedback. Some DT&E team members may not have access rights to the pre-production environment. The OT team may need to assume some of the responsibilities.

Some implementation may be necessary as well, recreating those parts of the software factory that had automated the execution of regression tests in previous phases. The DT&E team, which may or may not have access rights to the pre-production environment, can at least provide the requirements.

There is no point in repeating tests. The test teams should reach agreement on which regression tests are necessary and which are redundant. Redundancy may derive from regression tests performed in the test/release phases or from the previous iteration of the deliver phase. The types of changes, and the modules to which they are made, help determine which regression tests to perform and which to omit.

Table 5-25 describes scope of the Regression Testing Deliver Phase DT role by T&E activity.

Table 5-25. Regression Testing Deliver Phase - T&E Activities and DT Roles

T&E Activity	Description of Developmental Testing Role
Unit Test	Execute regression tests of changed modules that interface directly with platform-specific components in the pre-production environment
Integration Test	Execute regression tests of module sets and subsystems with changed modules that interface directly with platform-specific components in the pre-production environment
System Test	Execute system regression tests in the pre-production environment according to plan
Cyber Test	Ensure that cyber-related regression tests have been executed in the pre-production environment

Lessons Learned

Information will be added to the guidebook based on stakeholder feedback.

5.5.2 Interoperability Testing in the Deliver Phase

Interoperability testing during the deliver phase is conceptually like interoperability testing during the test phase. It evaluates whether a system performs correctly on all intended platforms and communicates properly with other systems. The user stories that drive interoperability testing during the test phase drive interoperability during the deliver phase. The difference is that interoperability testing in the test phase takes place in a software factory, whereas in the deliver phase, interoperability testing occurs in a pre-production environment. This difference necessitates testing the details of how the development team has planned for and implemented interoperability. For the test phase, the development team creates a model of the production environment (which the DT&E team is responsible for reviewing). Interoperability testing during the deliver phase validates that model.

Some aspects of validation are mundane. Suppose a system is developed in an unclassified software factory and delivered to a classified network. All external system addresses, whether expressed as URLs or IP addresses, will need to be changed. Good coding practice dictates that these addresses should be stored in configuration files. Interoperability testing in the deliver phase should reveal violations of this practice.

Data exchange considerations are important. The environment in the software factory draws on knowledge of formats and units of measurement in the production environment. Interoperability testing during the deliver phase validates how accurately this knowledge was implemented. Data query mechanisms must also be tested. Suppose it is known that the production environment contains a data broker that can be queried using Structured Query Language (SQL). SQL is well established and standardized. But every vendor that implements the standard offers its own extensions to SQL, and every version of an implementation adds new features or fixes bugs. Unless it is certain that the simulation of the data broker used in the software factory uses the same SQL implementation and version, interoperability testing during the deliver phase should exercise queries to validate the SQL results.

Authentication and authorization are other areas whose implementations may differ between the software factory and the pre-production environment. Modern systems deployed in multiple organizations need to fit into each organization's authentication protocols—using a Kerberos server, for example.[11] Interoperability testing in the pre-production environment validates whether the software factory can model authentication and other access-related issues sufficiently well.

Interoperability testing in the deliver phase should consider nonfunctional issues. The pre-production environment is more representative of the actual platforms, including both hardware and software, that operators will use, along with the networks connecting these platforms. Interoperability testing should evaluate whether platforms are capable of presenting information to operators in acceptable formats. Interoperability testing should evaluate whether the networks can handle the load.

For all these considerations, it is important to realize that the pre-production environment is still not the production environment. Interoperability testing in the deliver phase must not modify production data. It cannot assume that production data sets will be modified to provide expected states needed to run a test. The pre-production environment is containerized, just like the environment in the software factory used for testing. A container might allow access to a production data set for data retrieval but not for updates, which is not necessarily a good practice. It may introduce log entries in a production environment that have nothing to do with operational use. It has the risk of unintentionally allowing an update. Interoperability testing in a pre-production environment requires careful planning and implementation.

The DT&E team does not necessarily perform interoperability testing during the deliver phase. The activity may fall under the jurisdiction of OT&E. The DT&E team should at least be involved in planning. One important aspect of this planning is describing how to determine what the test results say about the fidelity of the test environment in the software factory to the real world and how to provide feedback to the development team on improving the software factory.

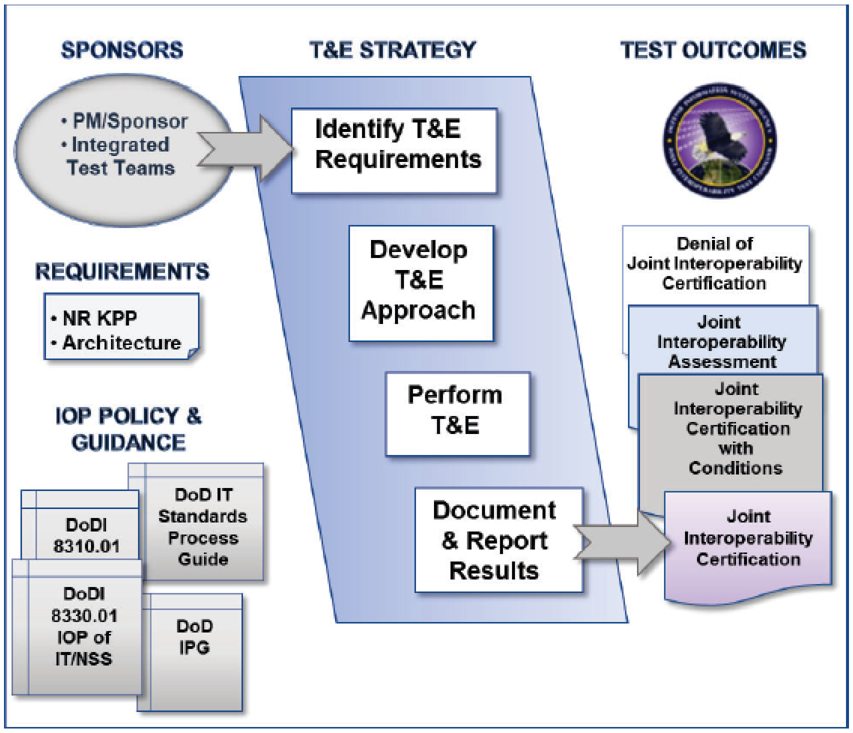
Objective

DoD-managed IT systems must be interoperable in accordance with Paragraph 1.2. of DoDI 8330.01, and the DoD Interoperability Process Guide defines how systems can be certified as

[11] See https://web.mit.edu/kerberos/krb5-1.21/.

interoperable. Figure 5-4 shows the general process flow of interoperability T&E along with notional roles and responsibilities. As noted in previous discussions in Sections 2.4. and 5.4.4, this flow is predicated on a standard waterfall development cycle bound by the phases of an acquisition process. Interoperability T&E was traditionally an OT activity and focused on the operation phase in waterfall development, although there were activities that require interoperability test team participation and coordination throughout the system development life cycle. The DoD Interoperability Process Guide does not require interoperability data collection requirements to be satisfied by OT&E; data may come from DT&E and UAT as well as OT&E. This broader view of acceptable sources accommodates the T&E that occurs during DevSecOps. Indeed, it is essential for a DevSecOps methodology, as noted in Section 5.4.4 on interoperability testing in the test phase.

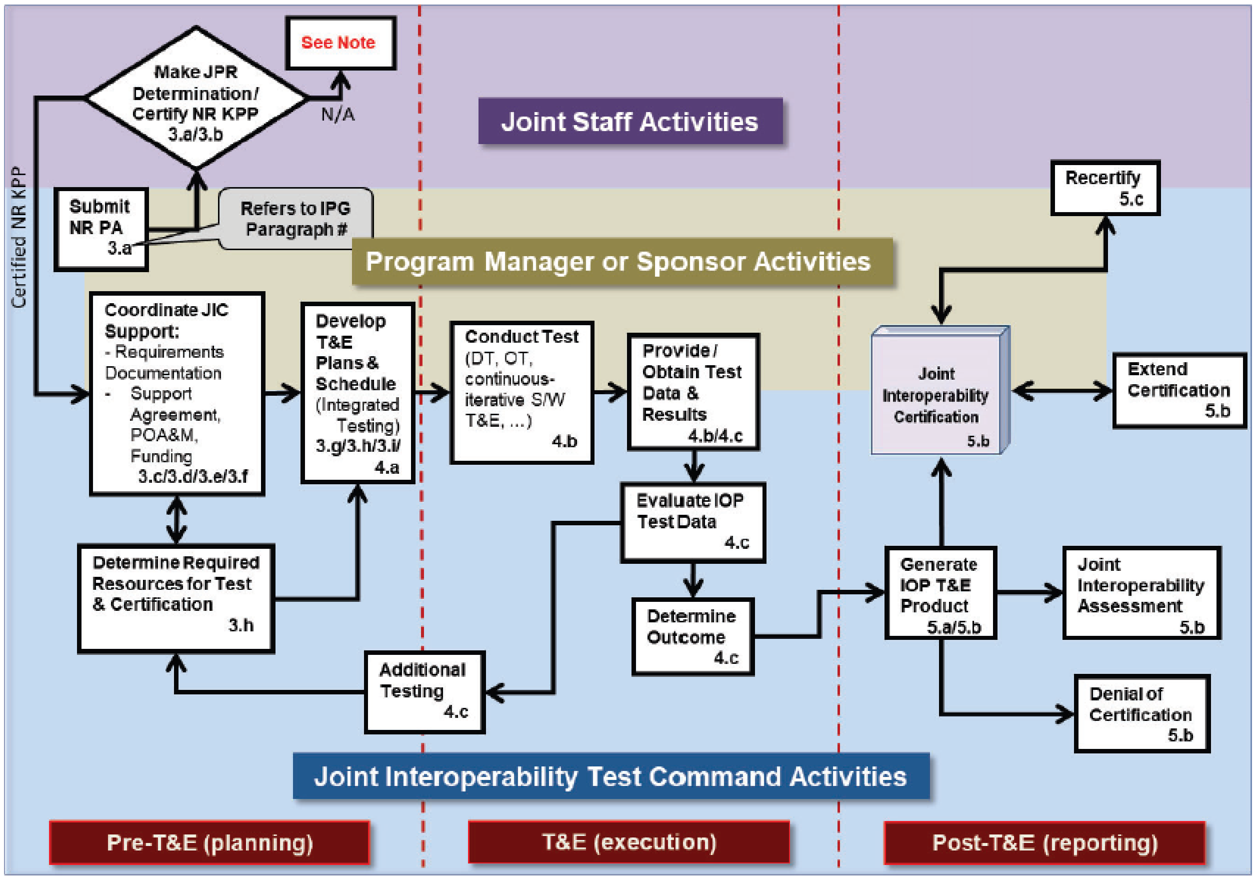
Figure 5-4 illustrates several test outcomes that fall short of full joint interoperability certification, including a denial of joint interoperability certification or a joint interoperability certification with conditions. The former, or failure to progress on the latter, could result in a system's placement on the operating at risk list overseen by the DoD CIO Interoperability Steering Group. Shifting interoperability T&E involvement as far left as possible in the process, especially through participation in the plan phase, provides the most effective risk mitigation strategy against interoperability failure and certification denial. The potential of a waiver also exists for those systems that can demonstrate that they have no interoperability interfaces or requirements, but this is not noted in the chart, as it largely sidesteps the joint interoperability process.


SPONSORS
T&E STRATEGY
TEST OUTCOMES
• PM/Sponsor
• Integrated Test Teams
Identify T&E Requirements
Develop T&E Approach
Perform T&E
Document & Report Results
REQUIREMENTS
• NR KPP
• Architecture
IOP POLICY & GUIDANCE
DoDI 8310.01
DoD IT Standards Process Guide
DoDI 8330.01 IOP of IT/NSS
DoD IPG
Denial of Joint Interoperability Certification
Joint Interoperability Assessment
Joint Interoperability Certification with Conditions
Joint Interoperability Certification

Source: DoD Interoperability Process Guide, Version 3.0, p. 4

Figure 5-4. Joint Interoperability Certification T&E Overview

Figure 5-5 provides an overview of a notional joint interoperability certification process. These process steps must be reconciled with the iterative steps of the DevSecOps methodology to provide required interoperability test data developed in the Information Support Plan (ISP) and TES. If T&E by the government test team is limited to the joint interoperability certification of the initial MVP and then predicated on automated testing to provide the data for maintaining interoperability and joint interoperability certification from that point on, a steady state of continuous monitoring for interoperability compliance can be achieved for the pipeline that frees the government test team to efficiently support multiple system joint interoperability certifications. Continuous monitoring is sustainable if the ongoing automated testing validates program compliance with industry, government, and Service standards and the use of approved virtual objects, containers, and guard rails.


Make JPR Determination/ Certify NR KPP 3.a/3.b
See Note
N/A
Joint Staff Activities
Certified NR KPP
Submit NR PA 3.a
Refers to IPG Paragraph #
Program Manager or Sponsor Activities
Recertify 5.c
Coordinate JIC Support: - Requirements Documentation - Support Agreement, POA&M, Funding 3.c/3.d/3.e/3.f
Develop T&E Plans & Schedule (Integrated Testing) 3.g/3.h/3.i/ 4.a
Conduct Test (DT, OT, continuous-iterative S/W T&E, ...) 4.b
Provide / Obtain Test Data & Results 4.b/4.c
Joint Interoperability Certification 5.b
Extend Certification 5.b
Evaluate IOP Test Data 4.c
Determine Required Resources for Test & Certification 3.h
Determine Outcome 4.c
Generate IOP T&E Product 5.a/5.b
Joint Interoperability Assessment 5.b
Additional Testing 4.c
Denial of Certification 5.b
Joint Interoperability Test Command Activities
Pre-T&E (planning)
T&E (execution)
Post-T&E (reporting)

Source: DoD Interoperability Process Guide, Version 3.0, p. 6

Figure 5-5. Notional Joint Interoperability Certification Process

The Global Information Grid Technical Guidance Federation, a suite of software applications on the Nonclassified Internet Protocol Router Network (NIPRNET) and Secret Internet Protocol Router Network (SIPRNET) maintained by the Defense Information Systems Agency for the DoD CIO, provides technical guidance, tools, and the repository for products of the joint interoperability process including the ISP, test outcomes, and certification artifacts; see DoDI 8330.01 for further information. This application suite provides the platform for interaction between the program and the various interoperability oversight bodies and authorities to achieve and maintain joint interoperability certification. Table 5-26 provides a set of high-level test objectives that can be used to develop the NR KPP performance metrics for the ISP and TES. These metrics, once selected, will help identify the automated or manual test tools that will be used as part of the automated testing and instrumentation built into the system development pipeline and employed during interoperability T&E of the MVP and subsequent versions meeting the interoperability recertification threshold.

Table 5-26. A Template for Net-Readiness Testing Objectives

Objective	Description
Visible	Users and applications can discover the existence of data assets through catalogs, registries, and other search services. All data assets (intelligence, non-intelligence, raw, and processed) are advertised or *made visible* by providing metadata, which describes the asset.
Accessible	Users and applications post data to a *shared space*. Posting data implies that: (1) descriptive information about the asset (metadata) has been provided to a catalog that is visible to the enterprise and (2) the data is stored such that users and applications in the enterprise can access it. Data assets are made available to any user or application except when limited by policy, regulation, or security.
Understandable	Users and applications can comprehend the data, both structurally and semantically, and readily determine how the data may be used for their specific needs.
Trusted	Users and applications can determine and assess the authority of the source because the pedigree, security level, and access control level of each data asset is known and available.
Agile	Many-to-many exchanges of data occur between systems, through interfaces that are sometimes predefined or sometimes unanticipated. Metadata is available to allow mediation or translation of data between interfaces, as needed.
Responsive	Perspectives of users, whether data consumers or data producers, are incorporated into data approaches via continual feedback to ensure satisfaction.

Source: Interoperability Test and Evaluation: A System of Systems Field Study (Colombi 2008)

Why It Matters to the DT&E Community

The final deployment stage into the operational environment is when OT&E would normally occur during the standard acquisition-based waterfall phases. However, under the rubric of the DevSecOps methodology and following the shift-left/continuum of testing construct, the automated tests to produce the necessary data for validating and maintaining interoperability are layered into the CD and CI developmental pipeline. This shift means that early and continuous involvement of the government test team from the plan phase onward is essential to achieving and maintaining interoperability.

Table 5-27 describes scope of the Interoperability Testing DT role by T&E activity.

Table 5-27. Interoperability Testing - T&E Activities and DT Roles

T&E Activity	Description of Developmental Testing Role
Unit Test	N/A
Integration Test	N/A
System Test	Strategies for and results of interoperability testing during the test phase influence interoperability testing during the deliver phase. The influence can extend from simple guidance to automated testing in the pre-production environment.

T&E Activity	Description of Developmental Testing Role
Cyber Test	Selected cyber tests used to evaluate interoperability during the test phase influence interoperability testing during the deliver phase.

Lessons Learned

Information will be added to the guidebook based on stakeholder feedback.

5.5.3 Performance Testing

Performance testing is a software testing process used for testing the speed, response time, stability, reliability, scalability, and resource usage of a software application under a particular workload. Performance testing is a type of nonfunctional testing. It analyzes how a system behaves, not what a system does.

Performance testing is requirements driven. System requirements should state the expected response time, maximum load (e.g., number of simultaneous users to support), minimum processors, memory, disk space, and network throughput needed to operate a system. The DT&E team uses these requirements to define performance tests.

There is no standard definition for the term "performance testing," but many sources agree that it comprises several types of tests, including:

- Load testing: Testing a system under expected real-world loads.
- Stress testing: Testing a system under loads close to, at, and exceeding maximum values as defined in the requirements.
- Soak testing: Testing a system's ability to handle a sustained load over a long period of time.
- Spike testing: Testing a system's ability to handle sudden spikes in traffic.

Setting up and carrying out performance testing can be time-consuming, effort-intensive, and resource-intensive. Soak testing requires a long period of time. Stress testing should consider the effects of a denial-of-service attack, which involves transmitting an abnormally high number of packets and attempts to consume available network bandwidth.

Performance testing does not necessarily yield pass/fail results. Data gathered from performance tests can be used to calculate expected system behavior under various loads and with different platform configurations. An organization can use this information to determine the platforms it requires to operate a system based on its expected usage profile. To the extent that system requirements dictate performance, a performance test can be used to determine whether a system is ready for deployment.

Objective

The objective of performance testing is to evaluate the performance and scalability of a system or application under various loads and conditions. It helps identify bottlenecks, measures system performance, and ensures that the system can handle the expected number of users or transactions.

Gathering system-level usage data and assessing the scalability of delivered capability for software developed using DevSecOps on a DevSecOps ecosystem or software factory means that what can be measured from a developmental testing perspective is a function of the tooling on the platform. Ideally, the tooling on the DevSecOps ecosystem enables the collection of data such as system-level usage metrics and automatically generates testing reports for analysis. The degree to which the ecosystem can produce the needed test results should be understood in the plan phase or when configuring a DevSecOps ecosystem. Adding testing tools to a DevSecOps ecosystem could trigger an ATO event, and additional time and funding may be required before coding and testing can begin.

The definition and scope of performance testing and scalability testing used in the DoD DT community differs from that used by modern software developers and testers in commercial practice. Although performance testing can be conducted in a software factory, it must ultimately occur in an operational environment—either a pre-production or production environment. In a pre-production environment, it is possible to execute tests without interfering with day-to-day operations. In a production environment, the emphasis is not on testing but on data collection and analysis, verifying that reality reflects predictions.

Examples of system-level usage metrics include:

- Processor Usage: The amount of time a processor spends executing non-idle threads.
- Memory Use: The amount of physical memory available to processes on a computer.
- Disk Time: The amount of time the disk is busy executing a read or write request.
- Bandwidth: The bits per second used by a network interface.
- Private Bytes: The number of bytes a process has allocated that cannot be shared among other processes. These are used to measure memory leaks and usage.
- Committed Memory: The amount of virtual memory used.

- Memory Pages/second: The number of pages written to or read from the disk to resolve hard page faults. Hard page faults are when code not from the current working set is called up from elsewhere and retrieved from a disk.
- Page Faults/second: The overall rate in which fault pages are processed by the processor. This again occurs when a process requires code from outside its working set.
- CPU Interrupts/second: The average number of hardware interrupts a processor is receiving and processing each second.
- Disk Queue Length: The average number of read and write requests queued for a selected disk during a sample interval.
- Network Output Queue Length: The length of the output packet queue in packets. Anything more than two means a delay and that bottlenecking needs to be stopped.
- Network Bytes Total/second: The rate at which bytes are sent and received on the interface including framing characters.
- Response Time: The time from when a user enters a request until the first character of the response is received.
- Throughput: The rate a computer or network receives requests per second.
- Amount of Connection Pooling: The number of user requests that are met by pooled connections. The more requests met by connections in the pool, the better the performance will be.
- Maximum Active Sessions: The maximum number of sessions that can be active at once.
- Hit Ratios: The proportion of queries (e.g., SQL statements) that are handled by cached data instead of expensive input/output operations. This is a good place to start for solving bottlenecking issues.
- Hits/second: The number of hits on a web server during each second of a load test.
- Rollback Segment: The amount of data that can rollback at any point in time.
- Database Locks: The locking of tables and databases must be monitored and carefully tuned.
- Top Waits: The wait times must be monitored to determine those that can be reduced when dealing with how fast data is retrieved from memory.
- Thread Counts: The application's health can be measured by the number of threads that are running and currently active.

- Garbage Collection: The unused memory must be monitored to ensure allocation of unused memory back to the system.

Why It Matters to the DT&E Community

Performance testing usually relies on data collection, using metrics such as those identified in Section 3.2. Although data can be collected manually, the process is time-consuming and effort-intensive. Automated data collection requires less human intervention.

Conversely, setting up automated data collection requires planning and implementation. The T&E WIPT and DT&E team are involved in the planning, which takes place during the plan phase. The results of planning, which may be stated in the system requirements, include the metrics to measure. Before the deliver phase, the DT&E team verifies that these metrics can be collected. Metrics collection can involve designing and coding the software being developed for these measures. It can also involve installing measurement tools in the DevSecOps ecosystem. Acquiring and installing tools can require purchasing software and obtaining licenses. The DT&E team should conduct reviews to ensure data can be collected.

Whether the DT&E team collects performance data depends on the environment to which a system is delivered. It may be on a platform to which the DT&E team lacks access. In that case, operational testers are responsible for collecting data and comparing actual performance to required performance. The data collection process follows the procedures from the DT&E team's reviews.

The nature of changes in a DevSecOps iteration determines what performance testing needs to occur for a delivered system. Minor changes, such as UI adjustments and improved error detection, seldom affect performance and do not require repeating performance tests. System architecture changes and changes to algorithms potentially affect performance and require repeating at least some performance tests. The development team is responsible for informing the T&E teams of how changes in an iteration might influence performance. The DT&E team is responsible for reviewing reports and recommendations made by the development team.

Table 5-28 describes scope of the Performance Testing DT role by T&E activity.

Table 5-28. Performance Testing – T&E Activities and DT Roles

T&E Activity	Description of Developmental Testing Role
Unit Test	N/A
Integration Test	N/A
System Test	• Determine how a system can support OT&E • Instrument a system to support OT&E

T&E Activity	Description of Developmental Testing Role
	• Test the instrumentation to ensure its correct behavior
Cyber Test	• Perform penetration testing on the test environment during the mission test • Identify opportunities for system compromise during the mission test

Lessons Learned

During the performance testing of software, testers look for performance symptoms and issues. Slow responses and long load times, for example, are often observed and addressed. Other performance problems can be observed including:

- Bottlenecking. This occurs when data flow is interrupted or halted because there is not enough capacity to handle the workload.
- Poor Scalability. If software cannot handle the desired number of concurrent tasks, results could be delayed, errors could increase, or other unexpected behavior could happen that affects:
    - Disk Usage
    - CPU Usage
    - Memory Leaks
    - Operating System Limitations
    - Poor Network Configuration
- Software Configuration Issues. Oftentimes, settings are not set at a sufficient level to handle the workload.
- Insufficient Hardware Resources. Performance testing may reveal physical memory constraints or low-performing CPUs.

5.6 Deploy Phase

5.6.1 Operational Test and Evaluation (Initial OT&E/Follow-on OT&E)

Objective

As articulated in the DoDM 5000.96, the objective of OT&E is to determine whether the system is operationally effective and suitable in its deployed environment.

Why It Matters to the DT&E Community

The DoDM 5000.96 applies to all DoD software-intensive systems or software embedded in systems employing iterative software development methodologies such as Agile and DevSecOps acquired via the DAS pursuing any AAF pathway in accordance with DoDI 5000.02.

Test data (e.g., measurements and assessments) collected by properly conducted DT&E may help support the objectives of OT&E, particularly activities as identified within the DoDM 5000.96.

Table 5-29 describes scope of the OT&E DT role by T&E activity.

Table 5-29. OT&E - T&E Activities and DT Roles

T&E Activity	Description of Developmental Testing Role
Unit Test Integration Test System Test Cyber Test	• Ensure that the product meets its design requirements, specifications, and performance goals • Identify and help resolve deficiencies in the development process before release and deliver • Reduce risk during the delivery, deployment, and operational phases

Lessons Learned

Information will be added to the guidebook based on stakeholder feedback.

5.6.2 Interoperability Certification

Interoperability certification is "the process of ensuring that a system meets the joint interoperability requirements of its users" (Watson 2010). It involves a review of whether a system conforms to applicable interoperability standards, as well as interoperability demonstrations with other systems. Collecting the data to drive these demonstrations is an important part of interoperability certification, as the data's realism determines whether the system can satisfy real-world interoperability needs.

Objective

The joint interoperability certification process ensures that a system meets the requirements to support joint interoperability across the intended user communities. Testing collects the necessary data to ascertain whether (or not) a given system meets applicable interoperability standards that would allow it to effectively perform information exchange with all relevant interconnected mission systems. This process ensures that combatant command, Service, and agency systems can interoperate in a joint, combined, and or coalition environment. All IT and national security systems that exchange or process information that support these environments must be certified by JITC before they are fielded. The certification process is outlined in DoDI 8330.01, which also requires that systems meeting the definition of interoperability be recertified every 4 years. Systems that do not have joint interfaces or information exchanges may be eligible for exemption.

The DT&E team's objective is to overlay the certification process with the DevSecOps process such that certification does not interfere with rapid DevSecOps iterations and CD. Because certification occurs only every 4 years, it is not likely to interfere with most DevSecOps iterations, which take place in a much shorter time frame. Project management staff is responsible for determining the iteration during which interoperability certification will occur. If full interoperability is not part of the MVCR specification approved by the MDA/DA, initial joint interoperability testing and certification may be deferred to a later iteration. The DT&E team is responsible for ensuring that a plan exists to evaluate interoperability consistent with the DoDI 8330.01 certification process; the data necessary to perform the process is available; and the DevSecOps ecosystem can support the certification test. Some aspects of interoperability can be tested as part of DT&E—those based on automated information exchanges that take place in the same ecosystem in which a system is developed. The DT&E team is responsible for ensuring that these tests are performed and for evaluating the test results. Other aspects of interoperability, such as those that require a production environment or manual interaction with a system, are the responsibility of OT&E. For these aspects, the DT&E team is responsible for reviewing the testing preparations.

Why It Matters to the DT&E Community

Joint interoperability T&E can be a repetitive process because system capabilities and features change frequently, but the interoperability certification process consists of the following four general phases, each of which requires participation from the interoperability test team:

- <u>Identify (Interoperability) Requirements</u>. This phase follows the development of the system NR KPP, as expressed in the ISPs, by the PM and the performance standards that will be tested and evaluated to demonstrate interoperability, derived through the JCIDS process in accordance with Chairman of the Joint Chiefs of Staff Instruction 5123.01, "Charter of the Joint Requirements Oversight Council and Implementation of the Joint Capabilities Integration and Development System." The interoperability test team (JITC, Service, or agency counterpart as relevant) must be involved in this determination. Artifacts from cybersecurity documentation may also meet some of the requirements for documenting the system NR KPP.
- <u>Develop Certification Approach (Planning)</u>. In a standard process, this phase would include coordination with the other governmental test entities and the PM in developing

the TEMP or TES documentation. It is essential, as part of the overall "shift-left/continuum of testing" construct, that as many relevant interoperability data elements as possible be incorporated and satisfied by automated or manual testing during the iterative backlog development for the addition of "features" to the MVP. These data are then captured for use during the operational testing phase for inclusion in the interoperability T&E and certification submission process. Leveraging the test and reuse principle, depending on the comprehensiveness of the developmental system environment during integration testing, these data can also be used to support the determination for recertification.

- Perform Interoperability T&E. Interoperability T&E is accomplished as part of the overall operational testing. Artifacts from cyber testing may also meet some of the requirements for the interoperability T&E.
- Report Certifications and Statuses. Report certifications and statuses are reported through JITC or the appropriate Service or agency interoperability T&E authority to the PM. Although a final interoperability certification is the goal, intermediate certifications must be achieved including certification of the initial NR KPP and ISP.

These general joint interoperability phases, also illustrated in Figure 5-4, require tailoring when they are overlaid with DevSecOps (as applied in a software factory pipeline) because they are predicated on a static view of what constitutes operationally deployed software (under the larger phases of acquisition) rather than a continuously evolving functional baseline. It becomes essential therefore to have a clear set of performance standards during the iterative phases of developmental and operational testing to support maintenance of certification or to determine whether a given system version has drifted far enough out of interoperability compliance to trigger recertification.

Table 5-30 describes scope of the Interoperability Certification DT role by T&E activity.

Table 5-30. Interoperability Certification - T&E Activities and DT Roles

T&E Activity	Description of Developmental Testing Role
Unit Test	N/A
Integration Test	N/A
System Test	N/A
Cyber Test	Selected cyber tests can be combined with interoperability tests to provide artifacts usable for RMF and joint interoperability certifications

Lessons Learned

Information will be added to the guidebook based on stakeholder feedback.

5.7 Operate/Monitor/Feedback Phases

DevSecOps recognizes three distinct post-deployment phases: operate, monitor, and feedback. From the standpoint of DT&E, these phases have the same objectives:

- To provide data to determine if the system is operating according to its functional and nonfunctional requirements. Nonfunctional requirements will include fault tolerance and resilience.
- To provide data that will help determine the ROI.
- To provide data on capability improvement since it was deployed.
- To provide data to determine if the system could be improved; is operating efficiently, and are users able to complete tasks easily?
- To provide data on the ease of incorporating future increments.

In all cases, the purpose of these phases as far as DT&E is concerned is to provide data.

5.7.1 System Monitoring

Once an application is deployed to production, system monitoring provides information about the application's performance and usage patterns so the product team can identify, mitigate, or resolve issues.

Objective

System monitoring after product deployment enhances the DT&E process by providing real-time insights into the software's operation, which leads to improved quality, performance, and security. It is an integral part of the DevSecOps life cycle that extends beyond the materiel release.

Why It Matters to the DT&E Community

DT&E is affected by system monitoring after product deployment in several ways:

- <u>Continuous Feedback</u>. Post-deployment monitoring provides continuous feedback on the software's performance in real-world conditions, allowing the product team to identify issues that may not have been apparent during the product development testing activity.
- <u>Data-Driven Decisions</u>. By analyzing the data collected from system monitoring (fault detection, log auditing), the product team can make informed decisions about necessary improvements or adjustments.

- Issue Detection and Resolution. System monitoring helps in the early detection of bugs, performance bottlenecks, and security vulnerabilities that might have slipped through the product development testing activity; enabling timely resolutions, and maintaining the integrity and performance of the software.
- Reduced Downtime. Effective system monitoring can significantly reduce downtime, as it allows for the quick identification and rectification of issues.
- Enhanced Security. Runtime monitoring is a post-deployment security testing technique that ensures the ongoing security of software systems by observing program behavior and detecting anomalies.

Table 5-31 describes scope of the System Monitoring DT role by T&E activity.

Table 5-31. System Monitoring - T&E Activities and DT Roles

T&E Activity	Description of Developmental Testing Role
Unit Test	N/A
Integration Test	N/A
System Test	N/A
Cyber Test	Continuous monitoring should inform iterative cyber tests throughout the system life cycle to characterize the mission risk associated with detections and newly discovered vulnerabilities.

Lessons Learned

Information will be added to the guidebook based on stakeholder feedback.

5.7.2 Value Assessment

Objective

Paragraph 3.3.b.(12) of DoDI 5000.87 states that "the sponsor and user community will perform a value assessment at least annually on the software delivered. The sponsor will provide feedback on whether the mission improvements or efficiencies realized from the delivered software capabilities are timely and worth the investment. The feedback should be informed by T&E results. The DA, sponsor, and PM will use the value assessments to assess progress on the program, update strategies, designs, and inform resourcing decisions." In the early stages of development, the level of capability delivered may not be sufficient to warrant formal operational testing, but DT results can provide sufficient data to inform the sponsor and the PM on the relative mission improvement achieved to date. Developmental testers should coordinate with their OT counterparts to capture mission-oriented measures and metrics, preferably at the system level, combined with objective performance metrics derived from SE principles to answer the question: What do we mean by value?

Continuous testing within a DevSecOps construct may provide sufficient detail to track mission improvement and its inherent value continuously as well, so that the annual value assessment process is streamlined and efficient.

Why It Matters to the DT&E Community

Value assessments are the sole decision gate evident in the software acquisition pathway post-MVCR. Unless the annual value assessment is informed by objective test data and weighted by operational mission performance impact, it will fall to unspecified members of the user community to make a subjective determination of "value."

Table 5-32 describes scope of the Value Assessment DT role by T&E activity; this role applies across all T&E activities.

Table 5-32. Value Assessment - T&E Activities and DT Roles

T&E Activity	Description of Developmental Testing Role
Unit Test Integration Test System Test Cyber Test	Contributes data (capabilities, limitations identification) to the value assessment determination

Lessons Learned

During development of this guidebook, we encountered a program that had developed a detailed process to identify performance factors that the program could measure during development and have its user community weight relative to mission impact. The program then built an algorithm to calculate objective values for the purpose of the value assessment.

6 DT&E in DevSecOps –Training

6.1 Tester Training

For many individuals in DoD, DevSecOps is a new cultural change different from the traditional way of doing business (e.g., the Waterfall model, with well-defined requirements before software development commences). With DevSecOps changing the way in which DoD develops, deploys, and protects software-intensive systems, this new paradigm requires an understanding of the processes, tools, and techniques for T&E. The first step in tester training is to accept the new culture and the concepts ingrained therein.

6.1.1 DevSecOps Training

PMs who shift to Agile and DevSecOps should ensure that formal training programs are established to help personnel fully understand the concepts and processes of the new methods that the program intends to adopt. The program should have individuals on staff who have experience in DevSecOps to mentor those new to DevSecOps. Several DoD programs have benefited from having a full-time Agile or DevSecOps coach on the team to continue where the formal training classes left off. All government test teams associated with the program should take full advantage of every training opportunity available.

6.1.2 Automated Test Tool Training

Automated test tools offer immense benefits by enhancing testing efficiency and effectiveness, but their successful utilization relies heavily on proper training. Key needs for automated test tool training include:

- Tool Familiarization; Understanding Tool Features. Training should cover the tool's functionalities, interface, capabilities, and limitations. Familiarity with the tool's UI is essential for efficient usage.
- Scripting and Automation Skills; Scripting Languages. Proficiency in scripting languages (such as Python, JavaScript, or specific tool-based languages) is crucial for creating automated test scripts.
- Test Design and Strategy; Creating Test Plans. Training should emphasize designing effective test cases, developing test plans, and creating test suites aligned with testing objectives.
- Framework Understanding; Implementation. Understanding and implementing test automation frameworks (e.g., keyword-driven, data-driven, or behavior-driven frameworks) is required to structure tests effectively.

- Maintenance and Troubleshooting; Maintenance Skills. Training should cover maintaining and updating automated test suites, as well as troubleshooting common issues and debugging scripts.
- Integration and Reporting; Integration Capabilities. Training should focus on utilizing the tool's capabilities to integrate with other tools, version control systems, and reporting mechanisms for comprehensive test reporting.
- Best Practices and Optimization; Optimizing Test Scripts. Training will provide best practices to optimize test scripts for efficiency, speed, and reliability, including reducing script execution time.
- Collaboration and Documentation; Team Collaboration. Training should focus on collaborative practices, version control, and documenting automated test processes for team understanding and knowledge sharing.
- Security and Compliance; Security Considerations. Training should provide awareness of security aspects in test automation, such as handling sensitive data and ensuring compliance with security protocols. Testers must ensure that tools are being patched, being used in the right way, and reducing code reuse security risks.
- Continuous Learning; Staying Updated. PMs should encourage a culture of continuous learning to keep up with tool updates, new features, and emerging trends in automated testing.
- Real-World Use Cases and Exercises; Hands-on Experience. Practical exercises and real-world use cases enable trainees to apply learned concepts in realistic scenarios, reinforcing their understanding.
- Certifications and Assessments; Certification Programs. Optional but valuable, training certifications can validate proficiency and provide structured learning paths for the tool.

6.1.3 Scientific Test and Analysis Techniques Training

DevSecOps and STAT training is recommended for government members of the DevSecOps teams responsible for ensuring that nonfunctional requirements such as performance, reliability, and scalability are rigorously addressed; optimizing test coverage of user stories; or assessing the implementation of automated software testing. The STAT COE provides training in DevSecOps; specific STAT such as design of experiments (DOE) and reliability; and the fundamentals of the STAT process. Each of the STAT COE courses relative to software DT&E in DevSecOps is highlighted below. Additional information on course details is available on the Air Force Institute of Technology, STAT Center of Excellence Website.

- STAT 110: Scientific Test and Analysis Techniques Overview. Designed for any audience, this class demonstrates how STAT is critical in the planning, design, execution, and analysis of test.
- STAT 201: Survey. This course will walk students through the life cycle of a survey from initial creation to interpretation of end results. Includes a discussion on how to decide if a survey is needed and how surveys are developed using the STAT process.
- STAT 410: Scientific Test and Analysis Techniques for Executives. Designed for executives (Senior Executive Service, flag officer, senior PMs), this class provides a high-level overview of STAT with a focus on planning and reporting results.
- STAT-D 251: Design of Experiments for Practitioners. This class introduces DOE basic techniques and processes needed to create a statistically rigorous and defensible test for complex systems.
- STAT-D 252: Design of Experiments for Practitioners. This follow-on course to STAT-D 251 examines (more closely) what a good design entails across a variety of metrics, introduces new classes of designs, and provides advanced modeling and analysis methods.
- STAT-D 320: Design of Experiments for Managers. This class introduces the basic concepts of STAT, with a focus on DOE and how it can be applied to test programs.
- STAT-DSO 220: DevSecOps – T&E Professional. Designed for senior T&E professionals (e.g., test managers, test leads), the class defines the DevSecOps software life cycle goals and activities as applied to DoD software-intensive systems. Software tools and automation used across DevSecOps will be demonstrated.
- STAT-DSO 230: DevSecOps – T&E Technical Practitioner. Designed for T&E technical practitioners (e.g., test analysts, test engineers) and test teams, this class defines the DevSecOps software life cycle goals and activities as applied to DoD software-intensive systems. Software tools and automation used across DevSecOps will be demonstrated. Individuals will participate in live, hands-on exercises that demonstrate the various tools and technology that are used throughout DevSecOps.
- STAT-R 220: Reliability for Practitioners. This class provides an overview of methods and concepts in reliability.
- STAT-R 250: Extended Reliability for Practitioners. This class teaches basic techniques and processes needed to create rigorous reliability tests, assess reliability metrics, analyze reliability data, and explain reliability growth.

6.1.4 System Training

Government test teams may not be familiar with some software-intensive systems such as Defense Business Systems. For example, the business processes are significantly different among hospitals, contracting offices, depot facilities, and investigative services. In such cases, test teams should work closely with the capability sponsors and user representatives in the Agile teams to understand how the system is supposed to work from the user perspective. Government test teams should consider attending user training classes on the new system if they are available.

6.1.5 Cyber T&E Training

DAU offers multiple cyber T&E courses and a cyber T&E workforce credential.

6.2 User Training

User training within the scope of DT&E is a system development component aimed at preparing end users to utilize the software or system being developed effectively and efficiently, and addressing the necessary skills, knowledge, and understanding required for individuals or groups who will interact with the system once it is deployed. Ensuring that users are well-prepared and proficient in using the software/system reduces potential issues, enhances productivity, and maximizes the system's effectiveness within an organization or user base.

The scope of user training in DT&E includes:

- Understanding System Functionality. Users need to comprehend the functionalities and features of the system. Training aims to familiarize users with the capabilities and limitations of the software or system.
- Operation Training. Operation training involves hands-on training to operate the system. Users learn how to perform various tasks, use different modules, and navigate through the interface.
- Troubleshooting and Issue Resolution. Users are trained on identifying common issues, understanding error messages, and implementing basic troubleshooting steps. This training enables users to address minor problems without external assistance.
- Compliance and Security Training. Users must be aware of security protocols, data handling procedures, and compliance requirements associated with the system. Training in this area ensures adherence to regulations and best practices.
- Customization and Advanced Features. For systems with customizable options or advanced features, training covers how users can tailor the system to meet specific needs or leverage more advanced functionalities.

- Change Management. Users need guidance in adapting to system updates or changes. Training programs often include strategies for managing change within the system and its impact on workflows.
- Documentation and Resources. Providing access to user manuals, guides, online resources, or help desks ensures ongoing support beyond the initial training phase.

Streamlining the development, fielding, and conduct of user training to keep pace with the speed of software development will be necessary to maximize the benefits of rapidly developed software. The DOT&E FY 2021 Annual Report noted that many programs adopting Agile practices have not instituted training development practices to keep pace with rapid releases of software.

Appendix: Use Cases

Updates to this guidebook will provide additional information based on stakeholder feedback.

This appendix captures stakeholder feedback on use cases that the DoD enterprise is facing and for which solutions have been found and can be shared across the community. Information will be added to the guidebook based on this stakeholder feedback. The following input has been captured to date.

Feedback on Use Cases	Stakeholder
How to change a user story to generate test and how to write a test/test script to test that requirement.	Not captured
How test teams support the launch of MVP, MVCR, and/or CRs.	Not captured
The utility of functional software models such as N^2 diagrams can also support T&E in a DevSecOps environment. Automated test and re-test (ATRT) are untenable over time without specific software functional models.	Not captured
Attribute the metrics defined in Section 3.2 as measuring the quality of the SUT measuring the quality of the testing. "Audit the list and ensure consistency."	USN, OPTEVFOR
Unclear if there are any nuances when the government owns the software factory versus industry owning and operating the software factory. What is different in roles? If nothing, perhaps explain this somewhere as an introductory stage setter.	OUSD(R&E) DTE&A
Review automated vs. manual testing content and placement.	OUSD(R&E) DTE&A
Planned future versions will include additional topics/updates including: (1) DoD enterprise DevSecOps common security activities; (2) security and cyber testing in DevSecOps.	Not captured
Describe DT events designed to evaluate performance, interoperability, reliability, and cybersecurity. The result of this scheduling activity is captured as high-level milestones on the Product Roadmap.	OUSD(R&E) DTE&A
4.1.1 Detailed Test Plan Development. The title of this section does not align with the content. Expected to see something addressing how each test plan is captured in the stage's JIRA (or equivalent) descriptions? This is an area DOT&E trips upon for OT as they want to review and approve test plans, but for pipelines, there is no formal test plan that gets approved for each of the micro tests that are continuously happening. Would there be a test plan required for manual tests?	OUSD(R&E) DTE&A
This guidebook does not fully vet non-acquisition projects like S&T. Late breaking addition to guidebook. Pre-program of record (S&T, Prototyping) included upfront (Section 1.2). To be reviewed and incorporated into document. Is it applicable from a DSO perspective?	Not captured
AI/ML w/in this guidebook, point to DTE&A and OT AI/ML guidance.	Not captured
Review regression testing context and messaging (see section 5.3.4, 5.4.3, 5.5.1)	OUSD(R&E) DTE&A Software Technical Director

Glossary

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, IaC, documentation, etc.

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. (DoDI 5000.87, p. 20)

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. (DoDI 5000.87, p. 20)

Continuous Delivery (CD). Continuous delivery is an extension of continuous integration to ensure that a team can release the software changes to production quickly and in a sustainable way. (DoD Enterprise DevSecOps Fundamentals, Version 2.5, p. 35)

Continuous Integration (CI). Continuous integration goes one step further than continuous build. It 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. (DoD Enterprise DevSecOps Fundamentals, Version 2.5, p. 35)

Developmental Test and Evaluation (DT&E). The disciplined process of generating substantiated knowledge on the capabilities and limitations of systems, subsystems, components, software, and materiel. This knowledge is used to inform decision makers on risks in acquisition, programmatic, technical, and operational decisions throughout the acquisition life cycle. DT&E assesses the maturity of technologies, system design, readiness for production, acceptance of government ownership of systems, and readiness to participate in OT&E, and sustainment. (T&E Enterprise Guidebook, pp. 3–4)

DevSecOps. DevSecOps is a combination of software engineering methodologies, practices, and tools that unifies software development (Dev), security (Sec), and operations (Ops). It emphasizes collaboration across these disciplines, along with automation and continuous monitoring to support the delivery of secure, high-quality software. DevSecOps integrates security tools and practices into the development pipeline, emphasizes the automation of processes, and fosters a culture of shared responsibility for performance, security, and operational integrity throughout the entire software lifecycle, from development to deployment and beyond. (DoD Enterprise DevSecOps Fundamentals, Version 2.5, p. 36)

DevSecOps Pipeline. A collection of DevSecOps tools, upon which the DevSecOps process workflows can be created and executed. (DoD Enterprise DevSecOps Fundamentals, Version 2.1, p. 39)

Integrated Decision Support Key (IDSK). The IDSK establishes the evaluation strategy. It articulates how an acquisition Program of Records's technical, programmatic, and operational decisions will be informed with an evaluation of the system's technical and operational capabilities using data generated from contractor, developmental, integrated, and operational test, and M&S events throughout the acquisition lifecycle. (forthcoming DODI 5000.DT)

Software Factory. A software factory is defined as a collection of people, tools, and processes that enables teams to continuously deliver value by deploying software to meet the needs of a specific community of end users. It leverages automation to replace manual processes. (DoD Enterprise DevSecOps Fundamentals, Version 2.5, p. 37)

Integration Definition for Function Modeling (IDEF0). IDEF0 is a method designed to model the decisions, actions, and activities of an organization or system. (https://www.idef.com/idefo-function_modeling_method)

Infrastructure as Code (IaC). Infrastructure as Code (IaC) is infrastructure definition and configuration that is defined with text files that are checked-in to a source code repository and kept under configuration management. It includes the management of networks, storage, virtual machines, load balancers, and even connection topologies. IaC evolved to solve a real-world problem referred to as environment drift in the release pipeline. (DevSecOps Playbook, Version 2.1, p. 5)

Safety-Critical Functions. Those software functions that, if they fail, will likely cause a failure of the mission:

- Functions associated with catastrophic and critical severities for friendly assets (infrastructure, personnel, systems, etc.) that directly affect mission effectiveness and capability and are therefore equally important.
- Functions currently utilized within mission impact analysis for severity reasons and, in addition, are criteria to identify critical infrastructure.
- Functions that can be vulnerable to exploitation, like mission-critical functions.
- Functions that are formulated and accepted by "competent persons" to identify and understand the ramifications of "function" failure so the task cannot adequately be

performed by anyone. (For example, arson (fire) investigators are used to investigate a house fire because they can tell you the cause of the fire, ignition point, etc., based on education and experience versus residents who went through a house fire and can only potentially tell you what happened to them—if they were cognizant of what occurred and why.)

Security as Code (SaC). The integration of automated security measures directly into the software development process. SaC introduces proactive rather than reactive security measures, which is essential given the increasing sophistication of modern cyber threats (https://www.crowdstrike.com/cybersecurity-101/cloud-security/security-as-code/). It is a methodology of codifying security tests, scans, and policies, and implementing them directly into the CI/CD pipeline to detect security vulnerabilities automatically and continuously.

Sprint. A short, time-boxed period when a scrum team works to complete a set amount of work. (https://www.atlassian.com/agile/scrum/sprints)

Structured Query Language (SQL). A specialized programming language for managing relational database data. It allows users to store, manipulate, and retrieve data efficiently in databases. (https://www.geeksforgeeks.org/what-is-sql/)

Technical Debt. Consists of design or implementation constructs 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 quality. (DoDI 5000.87, pp. 22–23)

Test Coverage, 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. (DoD Enterprise DevSecOps Fundamentals, Version 2.1, p. 41)

Unified Modeling Language (UML). A general-purpose visual modeling language that is intended to provide a standard way to visualize the design of a system. (*The Unified Modeling Language User Guide* (2nd ed.). Addison-Wesley, 2005, p. 496)

Unit Test. A short program fragment which exercises some narrow part of the product's source code and checks the results. (https://www.agilealliance.org/agile101/agile-glossary/)

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 comprised 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 an end user or user of a system. (DoDI 5000.87, p. 23)

Vulnerability. A verified weakness in a system, embedded subsystem, platform, system security procedure, internal control, or implementation by which an actor or event may intentionally exploit or accidentally trigger the weakness to access, modify, or disrupt normal operations of a system—resulting in a security incident or a violation of the system's security policy (e.g., a known software vulnerability that is actively being exploited). (Forthcoming DoD Cyber DT&E Guidebook, Version 3.0)

Weakness. A defect or characteristic that may lead to undesirable behavior. (Forthcoming DoD Cyber DT&E Guidebook, Version 3.0).

Acronyms

AAF	Adaptive Acquisition Framework
AI	Artificial Intelligence
API	Application Programming Interface
ATO	Authorization to Operate
CAE	Component Acquisition Executive
cATO	Continuous Authorization to Operate
CD	Continuous Delivery
CDD	Capability Development Document
CDT	Chief Developmental Tester
CI	Continuous Integration
CIO	Chief Information Officer
CNS	Capability Needs Statement
COE	Center of Excellence
CONOPS	Concept of Operations
CR	Capability Release
CSS	Cybersecurity Strategy
CWE	Common Weakness Enumeration
CyWG	Cyber Working Group
DA	Decision Authority
DAS	Defense Acquisition System
DAST	Dynamic Application Security Testing
DAU	Defense Acquisition University
DE	Digital Engineering
DevSecOps	Development, Security, and Operations
DoD	Department of Defense
DoDD	DoD Directive
DoDI	DoD Instruction
DOE	Design of Experiments

DOT&E	Director, Operational Test and Evaluation
DT	Developmental Test
DT&E	Developmental Test and Evaluation
DTE&A	Developmental Test, Evaluation, and Assessments
dTEaaC	developmental Test and Evaluation as a Continuum
FedRAMP	Federal Risk and Authorization Management Program
GSA	General Services Administration
HFE	Human Factors Engineering
HSI	Human Systems Integration
IaaS	Infrastructure as a Service
IaC	Infrastructure as Code
IAST	Interactive Applications Security Testing
ICAM	Integrated Computer-Aided Manufacturing
ICD	Initial Capabilities Document
IDEF0	Integrated Definition for Function Modeling
IDSK	Integrated Decision Support Key
IEC	International Electrotechnical Commission
IMS	Integrated Master Schedule
IPS	Integrated Product Support
ISO	International Organization for Standardization
ISP	Information Support Plan
IT	Information Technology
ITT	Integrated Test Team
JCIDS	Joint Capabilities Integration and Development System
JITC	Joint Interoperability Test Command
LDTO	Lead Developmental Test Organization
LFT&E	Live-Fire Test and Evaluation
M&S	Modeling and Simulation
MBCRA	Mission-Based Cyber Risk Assessment

MBSE	Model-Based Systems Engineering
MDA	Milestone Decision Authority
ME	Mission Engineering
ML	Machine Learning
MTTD	Mean Time to Detect
MVCR	Minimum Viable Capability Release
MVP	Minimum Viable Product
NIST	National Institute of Standards and Technology
NR KPP	Net Ready Key Performance Parameter
OSD	Office of the Secretary of Defense
OT	Operational Test
OT&E	Operational Test and Evaluation
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
OWASP	Open Worldwide Application Security Project
P&E	Prototyping and Experimentation
PaaS	Platform as a Service
PEO	Program Executive Officer
PM	Program Manager
PMO	Program Management Office
PPP	Program Protection Plan
PrjMgr	Project Manager
PSM	Product Support Manager
PTaaS	Penetration Testing as a Service
QA	Quality Assurance
RAM	Reliability, Availability, and Maintainability
RMF	Risk Management Framework
ROI	Return on Investment
RTM	Requirements Traceability Matrix

S&T	Science and Technology
SaaS	Software as a Service
SaC	Security as Code
SAST	Static Application Security Testing
SBOM	Software Bill of Materials
SCA	Software Composition Analysis
SCRM	Supply Chain Risk Management
SDLC	Software Development Life Cycle
SE	Systems Engineering
SOO	Statement of Objectives
SoS	System of Systems
SOW	Statement of Work
SQL	Structured Query Language
STAT	Scientific Test and Analysis Techniques
STRIDE	Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, and Elevation of Privilege
SUT	System Under Test
T&E	Test and Evaluation
TEMP	Test and Evaluation Master Plan
TES	Test and Evaluation Strategy
TTP	tactics, techniques, and procedures
UA	User Agreement
UAT	User Acceptance Testing
UI	User Interface
UML	Unified Modeling Language
USD(A&S)	Under Secretary of Defense for Acquisition and Sustainment
USD(R&E)	Under Secretary of Defense for Research and Engineering
UX	User Experience
WIPT	Working-level Integrated Product Team

References

Agile and Earned Value Management: A Program Manager's Desk Guide. Office of the Under Secretary of Defense for Acquisition and Sustainment, November 17, 2020. https://www.acq.osd.mil/asda/ae/ada/ipm/docs/AAP%20Agile%20and%20EVM%20PM%20Desk%20Guide%20Update%20Approved%20for%20Nov%202020_FINAL.pdf

Agile Assessment Guide: Best Practices for Adoption and Implementation. GAO-24-105506. U.S. Government Accountability Office, November 2023. https://www.gao.gov/assets/d24105506.pdf

Agile Metrics Guide: Strategy Considerations and Sample Metrics for Agile Development Solutions, Version 1.2. Office of the Under Secretary of Defense for Acquisition and Sustainment, November 11, 2020. https://www.dau.edu/sites/default/files/Migrated/CopDocuments/Agile%20Metrics%20v1.1%2020191122.pdf

Application Programming Interface (API) Technical Guidance. Washington, DC: Office of the Under Secretary of Defense for Research and Engineering, October 2023. https://www.cto.mil/wp-content/uploads/2023/11/API-Guide-2023.pdf

Automated Software Testing Implementation Guide. Scientific Test and Analysis Techniques Center of Excellence, April 2017. https://www.afit.edu/stat/statcoe_files/Automated_Software_Testing_Implementation_Guide.pdf

Beck, Kent, et al. "Manifesto for Agile Software Development," 2001. https://agilemanifesto.org/

Chairman of the Joint Chiefs of Staff Instruction 5123.01, "Charter of the Joint Requirements Oversight Council and Implementation of the Joint Capabilities Integration and Development System," October 30, 2021. https://www.jcs.mil/Portals/36/Documents/Library/Instructions/CJCSI%205123.01I.pdf

Colombi, John, Brannen C. Cohee, and Chuck W. Turner. "Interoperability Test and Evaluation: A System of Systems Field Study." CrossTalk: The Journal of Defense Software Engineering, Vol. 21, No. 11, pp. 10–14, November 2008. https://apps.dtic.mil/sti/tr/pdf/ADA499158.pdf

Department of Defense Cloud Computing Security Requirements Guide, Version 1, Release 4. Defense Information Systems Agency, January 14, 2022. https://public.cyber.mil/dccs/dccs-documents/

Department of Defense Cyber Developmental Test and Evaluation Guidebook, Version 3.0, currently under development to be published by OUSD(R&E)/DTE&A.

Department of Defense Cyber Table Top Guide, Version 2.0. Offices of Director of Defense Research and Engineering (Advanced Capabilities), Deputy Director for Engineering, and Deputy Assistant Secretary of Defense Developmental, Test, Evaluation, and Assessments, September 16, 2021. https://ac.cto.mil/wp-content/uploads/2021/09/DoD-Cyber-Table-Top-Guide-v2.pdf

Department of Defense Interoperability Process Guide, Version 3.0. Joint Interoperability Test

Command, July 2023. https://jitc.fhu.disa.mil/isg/downloads/IPG_Version_3_Final_signed_20230703.pdf (CAC required)

Department of Defense Technology and Program Protection Guidebook. Office of the Under Secretary of Defense for Research and Engineering, July 2022. https://aaf.dau.edu/storage/2022/09/TPP-Guidebook_Jul2022_DOPSR-approved_Released.docx.pdf

DevSecOps Continuous Authorization Implementation Guide, Version 1.0. Department of Defense, Office of the Chief Information Officer, March 2024.

DevSecOps Fundamentals Guidebook: DevSecOps Activities & Tools, Version 2.2. Department of Defense Chief Information Officer, May 25, 2023. https://dodcio.defense.gov/Portals/0/Documents/Library/DevSecOpsActivitesToolsGuidebookTables.pdf?ver=_Sylg1WJB9K0Jxb2XTvzDQ%3d%3d

DevSecOps Playbook, Version 2.1. Department of Defense Chief Information Officer, September 2021. https://dodcio.defense.gov/Portals/0/Documents/Library/DevSecOps%20Playbook_DoD-CIO_20211019.pdf

Director, Operational Test & Evaluation FY 2021 Annual Report. Office of the Director, Operational Test and Evaluation. https://www.dote.osd.mil/Portals/97/pub/reports/FY2021/other/2021_DOTEAnnualReport.pdf?ver=YVOVPcF7Z5drzl8IGPSqJw%3d%3d

DoD Directive 5000.01, "The Defense Acquisition System," September 9, 2020, as amended.

DoD Directive 5205.07, "Special Access Program Policy," September 12, 2024.

DoD Enterprise DevSecOps Fundamentals, Version 2.1. Offices of the Chief Information Officer of the Department of Defense and Under Secretary of Defense for Acquisition and Sustainment (unsigned), September 2021. https://dodcio.defense.gov/Portals/0/Documents/Library/DoD%20Enterprise%20DevSecOps%20Fundamentals_DoD-CIO_20211019.pdf

DoD Enterprise DevSecOps Fundamentals, Version 2.5. Chief Information Officer of the Department of Defense, October 2024. https://dodcio.defense.gov/Portals/0/Documents/Library/DoD%20Enterprise%20DevSecOps%20Fundamentals%20v2.5.pdf.

DoD Enterprise DevSecOps Reference Design, Version 1.0. Department of Defense Chief Information Officer, August 12, 2019. https://dodcio.defense.gov/Portals/0/Documents/DoD%20Enterprise%20DevSecOps%20Reference%20Design%20v1.0_Public%20Release.pdf?ver=2019-09-26-115824-583

DoD Instruction 5000.02, "Operation of the Adaptative Acquisition Framework," January 23, 2020, as amended.

DoD Instruction 5000.75, "Business Systems Requirements and Acquisition," February 2, 2017, as amended.

DoD Instruction 5000.83, "Technology and Program Protection to Maintain Technological

Advantage," July 20, 2020, as amended.

DoD Instruction 5000.85, "Major Capability Acquisition," August 6, 2020, as amended.

DoD Instruction 5000.87, "Operation of the Software Acquisition Pathway," October 2, 2020.

DoD Instruction 5000.89, "Test and Evaluation," November 19, 2020.

DoD Instruction 5000.90, "Cybersecurity for Acquisition Decision Authorities and Program Managers," December 31, 2020.

DoD Instruction 5000.91, "Product Support Management for the Adaptive Acquisition Framework," November 4, 2021.

DoD Instruction 5000.95, "Human Systems Integration in Defense Acquisition," April 1, 2022.

DoD Instruction 5000.97, "Digital Engineering," December 21, 2023.

DoD Instruction 5000.DT, "Developmental Test and Evaluation," currently under development by OUSD(R&E)/DTE&A to be approved by the USD(R&E) and published on the DoD Issuances Website.

DoD Instruction 5000.98, "Operational Test and Evaluation and Live Fire Test and Evaluation," December 9, 2024.

DoD Instruction 8330.01, "Interoperability of Information Technology, Including National Security Systems," September 27, 2022.

DoD Instruction 8510.01, "Risk Management Framework for DoD Systems," July 19, 2022.

DoD Instruction 8531.01, "DoD Vulnerability Management," September 15, 2020.

DoD Manual 5000.96, "Operational and Live Fire Test and Evaluation of Software," December 9, 2024.

DoD Manual 5000.UY, "Cyber Developmental Test and Evaluation," currently in review workflow to be approved by the USD(R&E) and published on the DoD Issuances Website.

Executive Order 14028, "Executive Order on Improving the Nation's Cybersecurity," May 12, 2021. https://www.whitehouse.gov/briefing-room/presidential-actions/2021/05/12/executive-order-on-improving-the-nations-cybersecurity/

Federal Acquisition Regulation: Use of Products and Services of Kaspersky Lab, September 10, 2019. https://www.federalregister.gov/documents/2019/09/10/2019-19360/federal-acquisition-regulation-use-of-products-and-services-of-kaspersky-lab

Human Systems Integration Guidebook. Washington, D.C.: Offices of the Deputy Director for Engineering and Under Secretary of Defense for Research and Engineering, May 2022. https://www.cto.mil/wp-content/uploads/2023/06/HSI-Guidebook-2022.pdf

ISO/IEC 7498-1:1994, "Information technology – Open Systems Interconnection – Basic Reference Model: The Basic Model." International Organization for Standardization and International Electrotechnical Commission, 1994. https://www.iso.org/standard/20269.html

ISO/IEC 27001:2022, “Information security, cybersecurity and privacy protection – Information security management systems – Requirements.” International Organization for Standardization and International Electrotechnical Commission, 2022. https://www.iso.org/standard/27001

JITC Guidebook for Development, Security, and Operations Test and Evaluation, Version 1.0. Defense Information Systems Agency, Joint Interoperability Test Command, July 2021. https://jitc.fhu.disa.mil/organization/references/publications/downloads/JITC_Guidebook_for_DevSecOps_v1.0_August_2021.pdf (CAC required)

Linh, Nguyen Duc, Phan Duy Hung, Vu Thu Diep, and Ta Duc Tung. “Risk Management in Projects Based on Open-Source Software.” ICSA ’19: Proceedings of the 2019 8th International Conference on Software and Computer Applications, pp. 178–183, February 19, 2019. https://dl.acm.org/doi/pdf/10.1145/3316615.3316648

Manual for the Operation of the Joint Capabilities Integration and Development System. Joint Staff J-8 Force Structure, Resources, and Assessment Directorate, October 30, 2021.

McDermott, John and Chris Fox. “Using Abuse Case Models for Security Requirements Analysis.” Proceedings of the 15th Annual Computer Security Applications Conference (ACSAC’99), pp. 55–64. Phoenix, Arizona, December 6–10, 1999. http://www.andrew.cmu.edu/course/95-750/docs/CaseModels.pdf

Minimum Elements for a Software Bill of Materials (SBOM). National Telecommunications and Information Administration, United States Department of Commerce, July 12, 2021. https://www.ntia.doc.gov/files/ntia/publications/sbom_minimum_elements_report.pdf

Mission Focused Evaluation – Guidance, Version 3.0. Office of the Director, Operational Test and Evaluation. https://www.dote.osd.mil/Portals/97/docs/TEMPGuide/Mission_Focused_Evaluation_Guidance_3.0.pdf

Nichols, Bill. “The Current State of DevSecOps Metrics.” *Carnegie Mellon University, Software Engineering Institute’s Insights (blog)*. Carnegie Mellon’s Software Engineering Institute, March 29, 2021. https://insights.sei.cmu.edu/blog/the-current-state-of-devsecops-metrics/

NIST Guidance, “Software Security in Supply Chains: Software Bill of Materials (SBOM),” May 5, 2022. https://www.nist.gov/itl/executive-order-14028-improving-nations-cybersecurity/software-security-supply-chains-software-1

NIST Special Publication 800-53, Revision 5, “Security and Privacy Controls for Information Systems and Organizations,” September 2020. National Institute of Standards and Technology, September 2020. https://doi.org/10.6028/NIST.SP.800-53r5

NIST Special Publication 800-218, “Secure Software Development Framework (SSDF) Version 1.1: *Recommendations for Mitigating the Risk of Software Vulnerabilities*, February 2022. https://doi.org/10.6028/NIST.SP.800-218

Paliwal, Ashwani. “SBOM vs SCA.” SecOps Solution, Inc., September 7, 2023. https://www.secopsolution.com/blog/sbom-vs-sca

Public Law 116-92, Section 800(a)(2)(B), “National Defense Authorization Act for Fiscal Year 2020, December 20, 2019. https://congress.gov/116/plaws/publ92/PLAW-116publ92.pdf

Quantitative Mission-focused Measures – Guidance, Version 3.0. Office of the Director, Operational Test and Evaluation. https://www.dote.osd.mil/Portals/97/docs/TEMPGuide/Mission_Focused_Metrics_Guidance_3.0.pdf

Sanger, David E., Nicole Perlroth, and Eric Schmitt. “Scope of Russian Hacking Becomes Clear: Multiple U.S. Agencies Were Hit.” The New York Times, December 14, 2020. https://www.nytimes.com/2020/12/14/us/politics/russia-hack-nsa-homeland-security-pentagon.html

Software Engineering for Continuous Delivery of Warfighting Capability. Washington, D.C.: Office of the Under Secretary of Defense for Research and Engineering, Office of the Executive Director for Systems Engineering and Architecture, April 2023. https://www.cto.mil/wp-content/uploads/2023/07/SWE-Guide-April2023.pdf

Test and Evaluation Enterprise Guidebook. Offices of the Under Secretary of Defense for Research and Engineering and Director, Operational Test and Evaluation, August 2022. https://www.test-evaluation.osd.mil/T-E-Enterprise-Guidebook/

Test Planning Guide. Scientific Test and Analysis Techniques Center of Excellence, October 5, 2022. https://www.afit.edu/images/pics/file/Final%200930_Test%20Planning%20Guide_2_2%20(1).pdf.

Vailshery, Lionel Sujay. “Python frameworks used in data science 2021.” Statista, March 28, 2024. https://www.statista.com/statistics/1338424/python-use-frameworks-data-science/

Watson, Chris. “Joint Interoperability Certification: What the Program Manager Should Know.” Defense AT&L, January–February 2010. https://apps.dtic.mil/sti/tr/pdf/AD1016371.pdf.

Websites

Air Force Institute of Technology, STAT Center of Excellence.
https://www.afit.edu/STAT/page.cfm?page=1093

CycloneDX: The International Standard for Bill of Materials (ECMA-424).
https://cyclonedx.org/

DAU Adaptive Acquisition Framework.
https://aaf.dau.edu/

DAU Glossary of Defense Acquisition Acronyms and Terms.
https://www.dau.edu/glossary/

DAU Product Roadmap.
https://aaf.dau.edu/aaf/software/product-roadmap/

DAU Program Management Metrics and Reporting.
https://aaf.dau.edu/aaf/software/metrics-and-reporting/

DoD CIO Library.
https://dodcio.defense.gov/library/

DoD Issuances.
https://www.esd.whs.mil/DD/DoD-Issuances/

GSA DevSecOps Guide.
https://tech.gsa.gov/guides/dev_sec_ops_guide/

GSA Section 508: 2024 Interagency Accessibility Forum.
https://www.section508.gov/

Joint Chiefs of Staff Universal Joint Task List.
https://www.jcs.mil/Doctrine/Joint-Training/UJTL/

Meyer, Bertrand, Object-Oriented Software Construction (2nd ed.). Prentice Hall, 1997, p. 342.
https://bertrandmeyer.com/OOSC2/

MITRE ATT&CK.
https://attack.mitre.org/

MITRE Common Weakness Enumeration.
https://cwe.mitre.org/

OpenAPI Initiative.
https://www.openapis.org/

OWASP Application Security Verification Standard (ASVS).
https://owasp.org/www-project-application-security-verification-standard/

OWASP CI/CD Security Cheat Sheet.
https://cheatsheetseries.owasp.org/cheatsheets/CI_CD_Security_Cheat_Sheet.html

OWASP Developer Guide, Security Champions.
https://owasp.org/www-project-developer-guide/release/culture_building_and_process_maturing/security_champions/

OWASP Threat Modeling.
https://owasp.org/www-community/Threat_Modeling

OWASP Top 10 API Security Risks – 2023.
https://www.owasp.org/API-Security/editions/2023/en/0x11-t10/

Practical Software and Systems Measurement (PSM) Continuous Iterative Development (Agile) Measurement Framework.
https://www.psmsc.com/CIDMeasurement.asp

Software Testing Stuff. "What is Test Effectiveness and Efficiency?"
https://www.softwaretestingstuff.com/2007/10/test-efficiency-vs-test-effectiveness.html

SPDX: System Package Data Exchange (SPDX®).
https://spdx.dev/

Software Developmental Test and Evaluation (DT&E) in DevSecOps

Office of the Director, Developmental Test Evaluation and Assessments (DTE&A)
Office of the Under Secretary of Defense for Research and Engineering
3030 Defense Pentagon
Washington, DC 20301
https://www.cto.mil/dtea
osd.r-e.comm@mail.mil | Attention: Software Engineering Technical Director