Carnegie Mellon University
Software Engineering Institute

Guide to Implementing DevSecOps for a System of Systems in Highly Regulated Environments

Jose Morales
Richard Turner
Suzanne Miller
Peter Capell
Patrick Place
David James Shepard

April 2020

TECHNICAL REPORT
CMU/SEI-2020-TR-002

Software Solutions Division

http://www.sei.cmu.edu

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

This report was prepared for the SEI Administrative Agent AFLCMC/AZS 5 Eglin Street Hanscom AFB, MA 01731-2100

DM20-0258

Table of Contents

List of Figures

List of Tables

Executive Summary

This document provides guidance for implementing DevSecOps (DSO) in defense or other highly regulated environments, including systems of systems. It provides information about DSO, its principles, operations, and expected benefits. It describes objectives and activities that are needed to implement the DSO ecosystem, including adopting the culture, deploying technology, and adapting processes.

Section 1 introduces the rationale for adopting DevSecOps, the dimensions of change required for that adoption, and the scope of this document.

Section 2 defines the DSO concept, key principles, and its operation. Special DSO concerns raised in high-risk environments (HREs) and SoS environments are also addressed.

Section 3 is an overview of the adoption guidance.

Sections 4-6 describe the activities for adopting DSO. The core of the guide, they address three major efforts: Preparation, Establishment, and Management.

- Preparation is necessary to create achievable goals and expectations and to establish feasible increments for building the ecosystem. This includes considering the distance between where you are and where you want to be, and understanding the effort necessary to travel that path.
- Establishing the ecosystem includes evolving the culture, automation, processes, and system architecture from their initial state toward an initial capability.
- Managing the ecosystem includes measuring and monitoring both the health of the ecosystem and the performance of the organization. The intent is to evolve the ecosystem toward a desired state in a manner that balances speed with depth, minimizes unnecessary rework, and constantly revalidates and adapts the desired state.

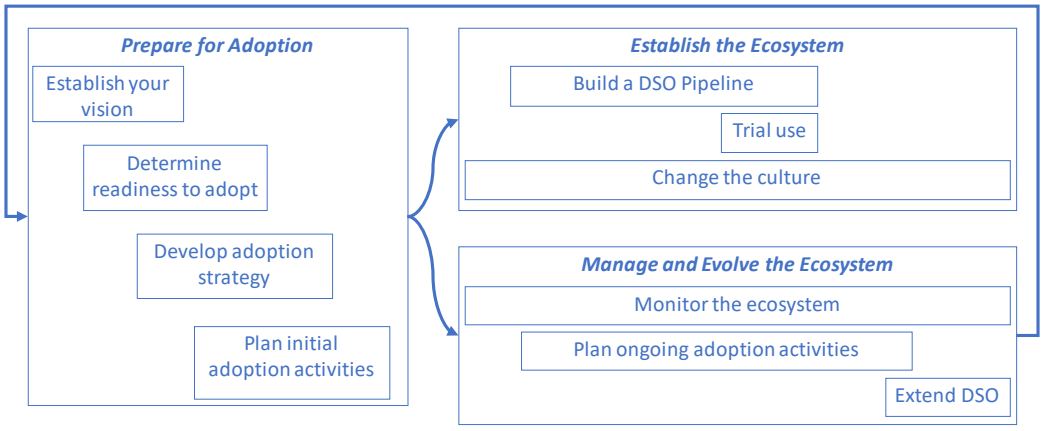


Figure 1: Structure of the Implementation Objectives and Activities

- Appendix A is a collection of the activity summaries.
- Appendix B contains references on technical subjects, change management approaches, and more advanced DSO information and guidance.

As illustrated, activities are not necessarily sequential—some are dependent on other activities, and some may be done simultaneously. The descriptions are general enough to support adaptation to varied environments.

Section 7 is a compendium of information on the concepts and principles that provide the foundation for DSO adoption—Technology Adoption, Culture Change, Lean and Agile, Systems Engineering, and others—to introduce the concepts and show how they relate to DSO success.

Abstract

DevSecOps (DSO) is an approach that integrates development (Dev), security (Sec), and delivery/operations (Ops) of software systems to reduce the time from need to capability and provide continuous integration and continuous delivery (CI/CD) with high software quality. The rapid acceptance and demonstrated effectiveness of DSO in software system development have led to proposals for its adoption in more complex projects. This document provides guidance to projects interested in implementing DSO in defense or other highly regulated environments, including those involving systems of systems.

The report provides rationale for adopting DSO and the dimensions of change required for that adoption. It introduces DSO, its principles, operations, and expected benefits. It describes objectives and activities needed to implement the DSO ecosystem, including preparation, establishment, and management. Preparation is necessary to create achievable goals and expectations and to establish feasible increments for building the ecosystem. Establishing the ecosystem includes evolving the culture, automation, processes, and system architecture from their initial state toward an initial capability. Managing the ecosystem includes measuring and monitoring both the health of the ecosystem and the performance of the organization. Additional information on the conceptual foundations of the DSO approach is also provided.

1 Introduction

Your team of developers will soon start a new software project. The project goal is the creation of a new capability that involves developing several systems dependent on each other to function appropriately, in other words, a system of systems (SoS). Some software or hardware used by the SoS is of a sensitive nature requiring development and testing in closed areas with high security—a high-risk environment (HRE). Your organization has decided to adopt DevSecOps (DSO) and requires your team of developers to use it for this project. In this document, we will help you implement DSO in an HRE for the development of an SoS.

DSO is a socio-technical system that integrates development, security, and operations in support of a continuous integration, continuous delivery (CI/CD) environment. DSO promises a high return on investment but requires a significant shift in existing culture, process, and technology. The DSO environment is specifically designed to increase system quality, reduce capability time-to-value, and minimize cognitive differences among the developers, securers, operators, and users of mission-critical defense and intelligence community systems.

Substantively changing even one of those things in an established organization is difficult. The impact of changing them all may seem impossible, but it has been done with significant success [CircleCI 2019]. Clearly, moving your organization down a path to DSO without compromising your existing mission goals and strategic trajectories is a daunting task—one that is different for every organization.

As shown in Figure 2, tools and practices are not the only consideration in DSO; successful adoption must navigate four interrelated dimensions of change.

Culture—DSO integrates activities of people with different mental models and responsibilities; this impacts the way the organization communicates and works together. DSO is a **no-blame** culture. "Blame-processing" wastes time and energy; the focus of DSO is on identifying, fixing, and preventing the problem from recurring. DSO culture is **transparent**; Lean and Agile practices require sharing information to make decisions at the lowest level and orchestrate the overall flow. DSO culture is **efficient**, constantly eliminating limited-value work. DSO is **integrated**, reducing silos and incorporating all of the disciplines (e.g., development, verification and validation/Quality Assurance, operations, users, managers, finance, procurement) in the team.

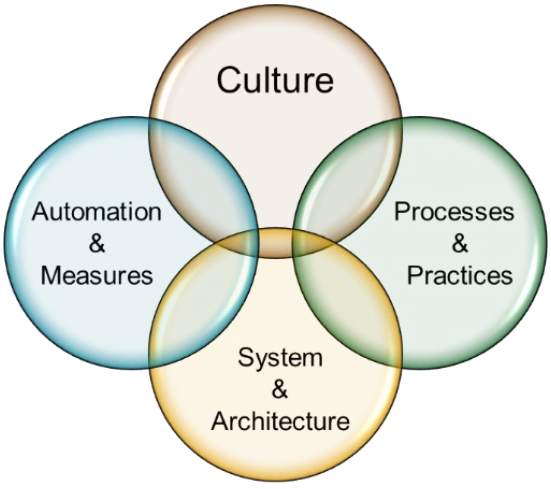


Figure 2: DSO Dimensions

Processes and Practices—DSO requires processes that support the culture and integrate the development and operations work. Organizational changes will likely be needed. Organizational

structures, work descriptions, responsibilities, reward systems/incentives, VV&A processes, procurement/licensing practices, decision making, and feedback mechanisms are examples of the DSO scope.

System and Architecture—DSO works most efficiently if the target system is architected to support the DSO practices. The architecture should support test automation and continuous integration goals; applications should support changes without release (e.g., late binding) and ensure the required -ilities (e.g., scalability, security, reliability).

Automation & Measures—While DSO is primarily about culture, people, and processes, automation is a **primary enabler** for achieving its benefits. DSO seeks to automate repetitive and error-prone tasks (e.g., build, testing, deployment, maintaining consistent environments), provide static analysis automation (for architecture health), and enhance communications and transparency through performance dashboards and other radiators.

1.1 Using the Guide

This guide was written to lead you through the activities to implement DSO practices and evolve your organization to provide the desired benefits.

Adopting a technology like DSO is like any other project—it needs a set of goals and measures, a management process, an adoption team to lead the transition, participation from the technical stakeholders that will use and benefit from the project, and the support and funding of the organization. The guide provides the information necessary to capture/create all of these and manage a successful adoption.

The guide is organized as follows:

- This Introduction describes the scope of the provided adoption guidance.
- Section 2 introduces the DevSecOps concept and describes how information flows though the DSO-enabled organization.
- Section 3 provides an overview of the adoption process.
- Sections 4-6 describe the activities for adopting DSO. This is the “meat” of the guide and comprises three major efforts: Preparation, Establishment, and Management.
- Section 7 is a compendium of information about the concepts and principles associated with approaches that provide the foundation for DSO adoption—change management, Lean and Agile, DevOps, systems engineering, architecture, and policy. This section is provided to introduce the concepts and show how they relate to DSO success.
- Appendix A is a collection of the activity summaries.
- Appendix B contains references on technical subjects, change management approaches, and more advanced DSO information and guidance.

1.2 Scope

This guide is intended specifically for adopting DSO infrastructure to support systems development in HREs and is based on *Implementing DevOps Practices in Highly Regulated Environments* [Morales 2018]. An HRE is enforced when some system software or hardware is of a sensitive nature, requiring development and testing in closed areas with high security. It addresses the creation and operation of a single, dedicated pipeline architecture that can be adopted and scaled to support multiple products.

The guide also suggests ways for the development team to address SoS issues. These issues arise in DSO operations when the software developed supports external functionality and needs to integrate, test, and receive feedback from systems outside software development boundaries.

Both of these topics are addressed more fully in Section 2.

2 The DSO Concept

DSO integrates software development and operational process security activities into the DevOps approach. It focuses on assuring security best practice is enforced at each step. For cyber-physical systems, DSO is a set of principles and practices emphasizing collaboration and communication among staff from engineering, hardware/software (HW/SW) design, development, integration and test, acquisition, security, services, end users, and any other stakeholders key to delivery of a secure software-intensive HW/SW system.

2.1 DSO Principles

DSO principles are based on the Lean and Agile principles, whose foundational concepts are addressed in Section 7, and DevOps principles [Kim 2016]. These principles are broadened to integrate development, security, and operations activities into a continuous integration/continuous deployment (CI/CD) pipeline. While several versions of these principles exist, the Software Engineering Institute (SEI) articulates them in Sections 2.1.1-2.1.8.

2.1.1 Collaboration

Full stakeholder engagement in every aspect of the software development lifecycle (SDLC), illustrated in Figure 3, facilitates full awareness and input on all decisions and outcomes. Developers, operators, engineers, end users, customers, and other relevant stakeholders are allowed to be part of decision making and work progress. This allows transition from development to operations to occur with fewer or no blockers.

2.1.2 Infrastructure as Code (IaC)

IaC is code written to build infrastructure. IaC is a core principle of DSO. The code specifies needed components and the details of how each should be installed. The medium where this infrastructure will be installed and executed can be specified in the code or at build time. The medium can be actual or virtualized hardware or a mix of both. Infrastructure is not limited—it can be the software of a single, end-user machine or an entire organization's enterprise. Software components such as operating systems, servers, and applications are typically specified in IaC. Hardware components such as storage, CPU, memory, and network topology are also specified. For more information, see *Infrastructure as Code: Final Report* [Klein and Reynolds 2019].

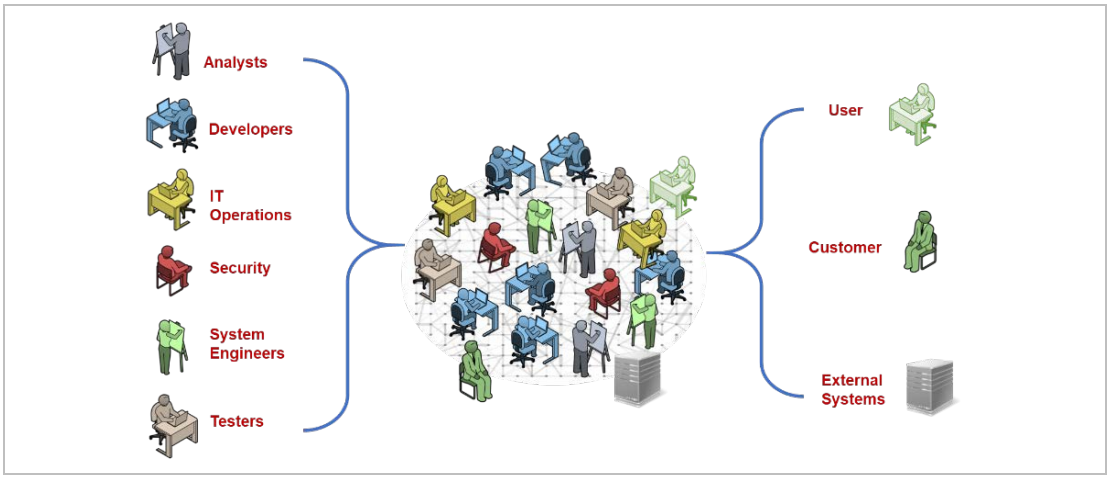


Figure 3: Full Stakeholder Engagement in SDLC

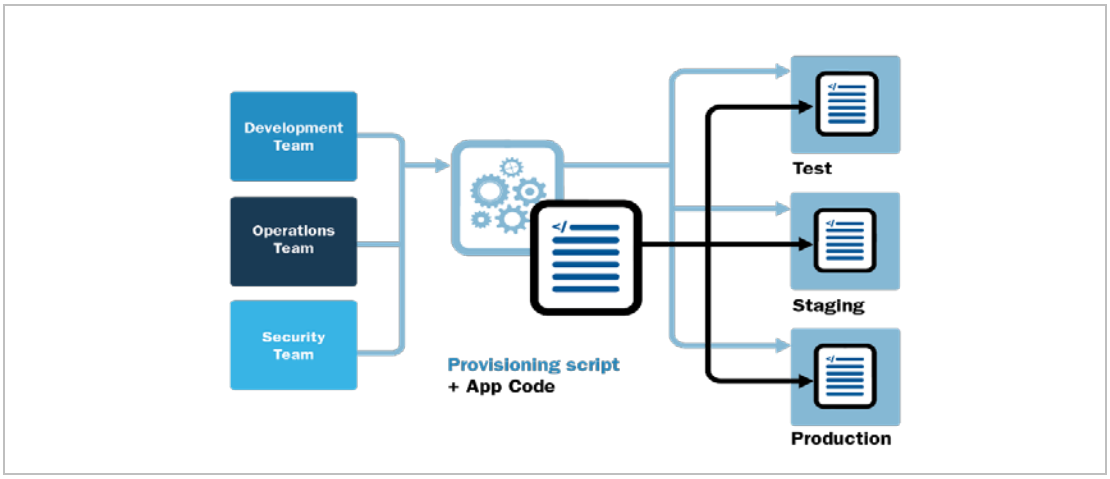


Figure 4: Scripting Environment + Application for IaC

From a DevOps perspective, the IaC code is usually a companion to software requiring this infrastructure to execute They are stored together in a version-controlled repository and used by some mechanisms, typically a build server, to produce the needed infrastructure with the software capable of running on it. Using version control guarantees standardization and reproducibility from source code; these further allow environment parity (identical or highly similar environments) of infrastructure and automated building. The ability to dynamically build a fully functioning infrastructure, as shown in Figure 4, sustains the velocity of integration while assuring environment parity in development, staging, and production. More information on IaC can be found in *Infrastructure as Code: Final Report* [Klein and Reynolds 2019].

2.1.3 Continuous Integration

CI is the unification of individual components into one entity. Unification occurs on a regular basis. The components, once unified, are meant to function together as a whole. The components may have dependencies on one another to function properly. CI is a core principle of DSO and is often referenced together with CI/CD, which we discuss in the next sections.

In DevOps, CI occurs at three specific phases in the pipeline.

1. The first phase is source code CI. This occurs in the version-controlled repository. Individual developers push their code to their personal branch in a repository. The repository pulls all the code from all personal branches into one unified master branch. The master branch is the culmination of the first phase. With this master branch, all forms of static analysis and testing should be performed. Tests should include a focus on security, needed dependencies, and program logic. Once all tests pass, the master branch can be cloned into a release branch. A new release branch should be created each time the master branch is updated and passes all tests.
2. The second phase is deployable component CI. This is typically carried out by the build server. The server pulls code from all master branches along with IaC and dependencies. These are all combined, creating a unified component placed on a deployable medium. The component represents the executable form of all the pulled source code installed within the infrastructure needed to successfully run. With this component, all forms of dynamic analysis and testing should be performed. The tests should include a focus on proper execution, security, expected input, output, and results.
3. The third phase is capability CI. This occurs in the staging environment. All deployable artifacts from multiple development teams are unified within one staging environment. The environment represents the current state of the capability being developed for a given project. The current state is measured by the portions of the capability, housed in deployable components, currently present in the staging environment. With this staging environment, dynamic analysis and testing should be performed. The tests should include a focus on integration and security. Integration testing should ensure that newly added components function as expected without disrupting the execution of other components.

At the beginning of the pipeline, a developer must manually push their completed source code to their personal branch in the version-controlled repository. This is usually referred to as a *commit operation*. The CI steps taken from personal branches to the staging environment can be fully automated. The automation includes all testing. The whole automation process is dependent on the initial, manual step of developers pushing their source code into respective branches. This leads to the often-stated phrase among developers “commit often.”

2.1.4 Continuous Delivery

CD is the automated transfer of software to an environment that has parity with the production environment, followed by a manual decision to transfer the software into production. The environment-sharing parity with production is often referred to as the *staging environment*. CD is the culmination of CI. CD encompasses the last steps of CI and represents the actual act of placing software into staging and production. This critical step is a needed manual decision to transfer code into production. This occurs due to additional tasks the software must complete along with manual validation of their outcomes before allowing transfer into production. A CI/CD pipeline can be fully automated up to staging but must pause for the manual decision, allowing a push into production. This is part of the DevOps pipeline, but it slows down at this point due to the need for manual decision making to proceed. This is the dominant form of CI/CD in HREs. In this document, we only refer to this form of CI/CD.

2.1.5 Continuous Deployment

CD is the automated transfer of software directly into a production environment. The production environment is typically live and in full operation. In contrast to continuous delivery, there is no manual decision needed. There is also no staging environment. This form of CD relies on rigorous static testing of source code and dynamic testing of deployable artifacts. Very small pieces of code are purposely pushed through this pipeline to facilitate rigorous testing. The lack of testing in a staging environment introduces a high risk of defects entering the production environment. This form of CI/CD is rarely used in HREs.

2.1.6 Environment Parity

Environment parity occurs when two or more environments are as identical as possible. In DevOps, parity is pursued between staging and production and between development environments. The use of IaC and deployable artifacts is critical to achieving parity. With parity, developers work in identical environments. Parity between staging and production reduces or eliminates potential problems in the production environment, although integration testing should always be performed in production.

2.1.7 Automation

Key to a successful DSO process is the scripted configuration and automation of Build, Automated Testing, and Automated Delivery/Deployment in continuous iterative cycles. Automation is a core principle of DSO. An example of implementing automation with continuous deployment in DSO is shown in Figure 5.

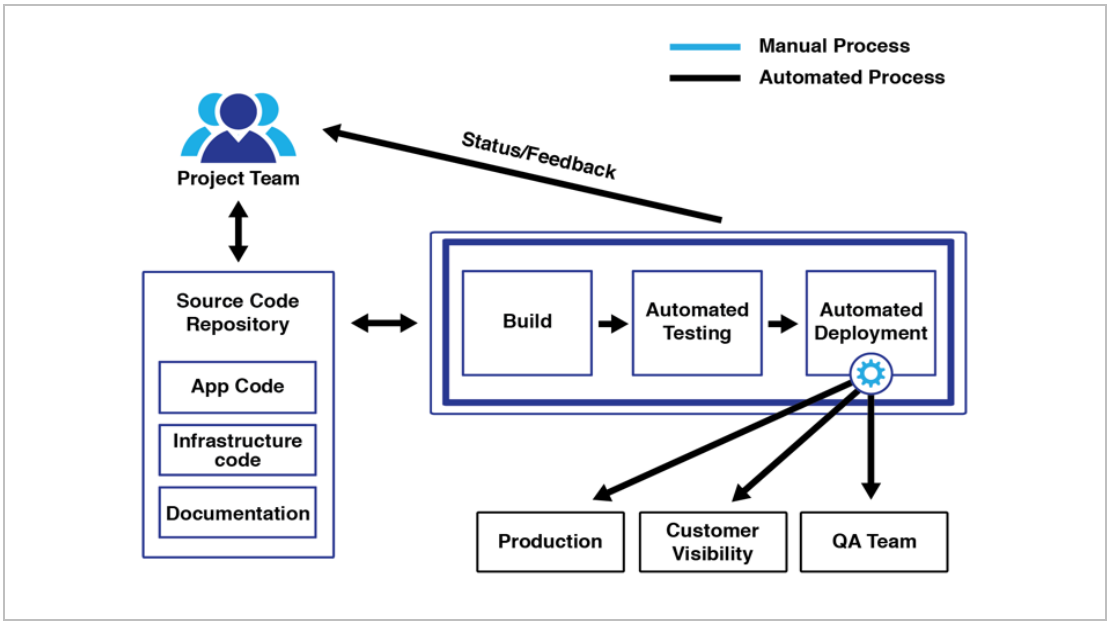


Figure 5: Automation in DevOps

Removing tasks from developers allows for focused code writing and testing. A single command issued by a developer pushes code to a version-controlled repository. Another command builds an entire project and the needed platform for execution onto a selected medium. A separate

command commences complete testing with integrated deliveries and deliveries into staging and production environments.

2.1.8 Monitoring

Continuous monitoring using a collection of performance metrics simultaneously improves the DSO pipeline and software under development. Monitoring is a core principle of DSO. Improvement occurs by alerting stakeholders when the pipeline falters or the software under development fails tests or execution.

Examples of where processes falter in the pipeline include incomplete code deliveries, failed test starts, unresponsive staging environments, unsuccessful builds, and denied pushes to the repository. Examples of failures in software under development are failed unit tests, failed functional tests, system crashes in staging or production, and unexpected functionality after integration with other software.

Automation and continuous cycles of development and test require non-stop monitoring to ensure an optimally functioning pipeline. At the same time, work progress, satisfaction of requirements, and, most importantly, quality are quantified with code execution and testing results. Automated resolution can occur for some sub-optimal metrics while the balance may require manual intervention.

2.2 DevOps Pipelines

A pipeline assists all stakeholders in every aspect of software development including building, testing, delivery, and monitoring. For engineers, the main use of a pipeline is to build, test, and deliver code through automation and continuous iterative processes. A pipeline is the technical implementation of DevOps principles. In the general sense, DevOps encapsulates culture, process, and technical components. The pipeline is the realization of the technical and, to some extent, process components of DevOps.

For the purposes of software development and following an SDLC, a pipeline has the following general uses:

1. **Code development**: This includes the writing, testing, and delivery of code. A pipeline facilitates the complete environment needed for multiple developers to write, integrate, and test multiple code segments in a continuous iterative process. The majority of this process is automated, especially in testing and delivery.
2. **Project management**: Resources such as ticketing systems, centralized document repositories, shared schedules, work progress monitors, and other performance metrics facilitate overall progress on any software project to all stakeholders.

Figure 6 illustrates the interaction between a development pipeline and the software development process, including the critical role played by iteration.

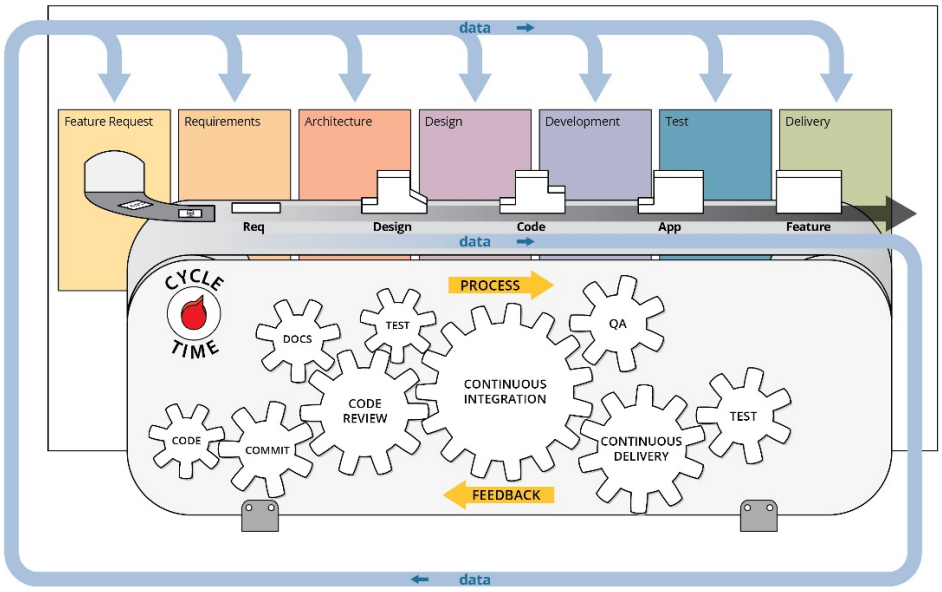


Figure 6: Communication Between the Development Pipeline and the SDLC

Here are the core activities that occur in a pipeline:

1. **Feature request:** This is the initial idea for the software project as discussed by the customer, developers, operators, and other stakeholders. This is a manual step that produces the set of requirements and schedule.
2. **Project configuration:** This is a combination of requirements, architecture, and design. Project configuration is a pre-step to development. This step provides input to the pipeline in the form of project requirements and schedule. A new instance of the pipeline is created for use with this project. The ticketing system is populated with project tasks that encompass code writing, testing, and delivery. The document repository stores the schedule and requirements for all stakeholders to view and edit (if needed).
3. **Code/test:** This activity captures code and commit. The interactive development environment (IDE) is used by development engineers to write source code as defined in a code-writing task on the ticketing system. As work advances, the engineer logs hours in the ticketing system spent on development. If issues arise, a ticket is created and assigned to personnel to resolve. A commit of source code occurs by pushing code to the version-controlled system repository in what is called a *branch*. This can be viewed as a folder with all the pushed code from a specific developer. A push of existing source code occurs each time a developer recommences work. Each developer has an individual branch in the repository, and only their code is pushed there. Testing at this phase is done by the engineer, normally on their IDE, to assure the code functions properly. Since the source code is present, white-box testing (testing with knowledge of the source code) can be implemented. Source code reviews with peers and other stakeholders also occur at this stage.
4. **Commit/code review:** This occurs in the version-controlled system repository after code reviews have validated an engineer's source code. Validation results from shoulder-to-shoulder source code reviews with peers and the analysis of testing results from code/test described above. The act of committing is pushing code from an engineer's individual branch into the

project's master branch. This is typically a one-button automated action. The master branch stores all written code that has been validated and approved for committal. A master branch itself is versioned and scripts should be auto-generated to recreate any version.

5. **Continuous integration/testing:** This is the first half of CI/CD; it involves the version-controlled system repository, build, provision, and dependencies servers, and the staging environment. For this step to occur, a user must first provide the build server with instructions on how to assemble the code and environment into a working system. With these instructions, the build server will pull code from the master branch of the repository and needed artifacts from the dependencies server to build a project image. The project image is pushed to the user-defined environment, which is pulled from the provisioning server by the build server. As building occurs, all needed configurations for applications and the system are done according to the user's instructions. The result is a fully functioning system that can be delivered into a staging environment for testing. Automated software-driven system building is a form of implementing infrastructure as code (IaC). The user needs to create test procedures with data sets for each requirement of the system. The test results validate that the requirement is either met or not met. Some test procedures can be auto-generated. Test results are sent to the developers for review and potential modifications. In the latter case, the modified code repeats activities 3 and 4.
6. **QA/integration testing:** This is an extension to CI/CD with further tests being executed to cover areas such as security and usability. This can be implemented as additional tests that are added for execution in staging and delivery.
7. **Continuous delivery:** This is the culmination half of CI/CD—transferring software to the staging environment followed by a manual decision to push to the production environment. The main purpose of continuous delivery is to allow a pause once staging tests complete to manually decide if the software is ready for the push into production. Recall that parity between staging and production is as similar as possible and thus the same tests should be run in both environments.
8. **Feature delivery:** This is the handover of the completed project to the customer—the final step of the development pipeline. The whole system should be delivered into production and all tests rerun to assure successful completion. At this point, the customer takes control of the system and receives documentation.
9. **At every step:** Any and all documents produced are stored in the document repository and linked via a user interface for stakeholders to access.

2.3 HREs and DSO Security

For a large portion of industry, academia, and government, applying DSO to the SDLC is straightforward. While some regulatory issues are part of any development, particularly in the government, HREs provide high levels of security regarding classified, proprietary, or otherwise controlled information. This environment is typically characterized by the following: air-gapped[1] physical spaces and computer systems with heightened security and access controls, segregation

[1] In this guide, *air-gapped* denotes a physical space, personnel, computer system, or other technology physically and digitally isolated from the rest of the HRE and all HRE external entities.

of duties, inability of personnel to discuss certain topics outside of closed areas, and the inability to take certain artifacts off premises. An HRE can be referred to as a closed area, classified space, controlled access area, or a Sensitive Compartmented Information Facility (SCIF).

2.3.1 HRE Challenges

As mentioned above, HREs are typically encumbered with regulatory constraints on everything from sharing process-related information to technical details only being available onsite to persons holding a particular clearance level. The closed nature of HREs leads to the following (at minimum) impediments to the implementation of Agile, Lean, and DSO practices:

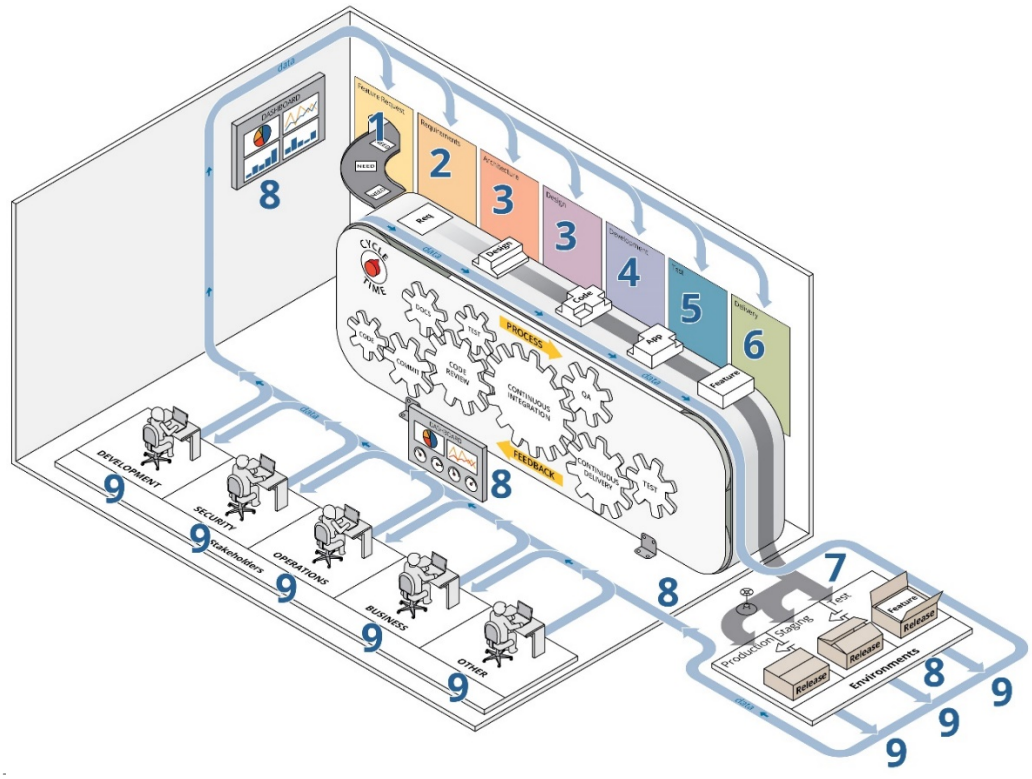
- Collaboration is impaired because individuals have different security clearances; therefore, they can only "see" certain things.
- Some tools may be authorized for one part of the setting but are not authorized in other parts of the setting due to certification or authorization issues.
- IaC may be impaired because assets that would normally be included have restricted access.

2.3.2 HRE Considerations

Considerations for HREs include any or all of the following:

- How will collaboration be achieved if the individuals involved have different access levels to the data being produced?
- How will authority to operate be accomplished in all the different enclaves that the tools and data will reside in as development progresses?
- Which assets can be incorporated into the IAC baseline? Which cannot and how will they be made available to all relevant parties?
- How will tool authorizations across different enclaves be kept up to date as the development progresses?

Figure 7 shows a typical DSO ecosystem specifically calling out the security-related additions. In this figure, we suggest critical security topics that can be addressed in multiple ways. Implementation details for these topics are left to the user.



1 Feature Request
- Strategy & Metrics
- Policy & Governance
- Education & Security Guidance
- Organizational Risk Factors
- Threat Assessment

2 Requirements
- Security Requirements (SFR/SAR)
- Risk Assessment
- Abuse Case Development
- Threat Modelling
- Security Stories
- Screen Development Tools
- Secure/Hardened Environments

3 Architecture & Design
- Security Architecture
- Architectural Risk Analysis
- Security Design Requirements
- Attack Surface Analysis
- Threat Modelling
- Vulnerability Analysis and Flow Hypothesis
- Security Design Review
- Dependencies List, Open-source libraries

4 Development
- Secure Coding Practices
- Security Focused Code Review
- Deprecate Unsafe Functions
- Perform Security Unit Testing
- Static Code Analysis
- Checking of process and procedures for secure coding & traceability

5 Testing
- Security Test Planning
- Security Testing
- Fuzz Testing
- Risk Based Security Testing
- Perform Dynamic Analysis
- Penetration Testing
- Verification of Security Implementation
- Verification of Process and Procedures
- Dependency Monitoring

6 Delivery
- Container Security
- Final Security Review
- Certify, Release and Archive
- Security Acceptance Testing
- Transition Incident Response Plan

7 Deploy
- Application Security Monitoring
- Secure Deployment Process
- Secure Environment
- Secure Operational Enablement

8 Data
- Mean Time to Recovery (MTTR)
- Mean Time to Detection (MTTD)
- Issue Volume and Resolution Time
- Deployment Frequency
- Change Lead Time and Volume
- Change Failure Rate
- Time to Patch Vulnerabilities
- Development and Application Logging Availability
- Retention Control Compliance
- SAR Findings
- Attack Vector Details (IP, Stack Trace, Time, Rate of Attack, etc)
- Server Disk Space, Load and Process Monitoring
- Application Performance
- Maximize Monitoring
- Change in Size to Code Base
- Most Active Code Contributors
- Most Changed Code Areas

9 All stakeholder security issues are satisfied

Figure 7: DevSecOps Overview (The Software Factory)

In general, pipeline security should assure that all activities on the pipeline occur in the expected manner. This includes the overall operation through the technical configurations. Critical to achieving this security is full awareness by multiple persons of the modifications made. Tight access control is also required to limit the personnel interacting with the pipeline's configuration and operation. Extensive security testing of the pipeline should occur to ensure a user cannot

invoke unintended outcomes, facilitating malicious goals. The second security focus is on the code under development. Security for code under development should achieve the following:

1. disallow insecure memory usage, such as pointers
2. is free of vulnerabilities
3. has exceptional input validation
4. interacts with only authorized persons and systems

The best practices of securing code under development in DSO are the same as those in the community. The difference is that, in DSO, the practices are automated via testing and validation. The extent to which the code is tested can determine its security. The diversity of testing also plays a role in determining how secure code is. Creating security requirements and misuse cases will facilitate capturing concerns in this area, especially for the customer. For additional information, view Hasan Yasar's webinar *Security Practitioner Perspective on DevOps for Building Secure Solutions* (https://youtu.be/U8972_RR9p0) [Yasar 2018].

2.4 SoS and DSO

Defense systems are increasingly developed as or evolve into systems of systems (SoSs) comprising a number of constituent systems that operate together to provide syncretic, synergistic capabilities. The various governance methods involved with SoS may raise significant issues when deploying a DSO environment.

The U.S. DoD *Systems Engineering Guide for Systems of Systems* [DoD 2008] offers the following definitions:[2]

System: *A functionally, physically, and/or behaviorally related group of regularly interacting or interdependent elements; that group of elements forming a unified whole [DoD 2001, DoD 2006]*

Capability: *The ability to achieve a desired effect under specified standards and conditions through combinations of ways and means to perform a set of tasks*

2.4.1 Types of SoS

There are four types of SoS found in the DoD today [Bass 2015, Dahmann and Baldwin 2008, Maier 1998]:

- **Virtual.** Virtual SoS lack a central management authority and a centrally agreed-upon purpose for the SoS. Large-scale behavior emerges—and may be desirable—but this type of SoS must rely upon relatively invisible mechanisms to maintain it.
- **Collaborative.** In collaborative SoS, the component systems interact more or less voluntarily to fulfill agreed-upon central purposes. The Internet is a collaborative system. The Internet Engineering Task Force works out standards but has no power to enforce them. The central players collectively decide how to provide or deny service, thereby providing some means of enforcing and maintaining standards.

[2] Source references have been changed to conform with this document.

- **Acknowledged.** Acknowledged SoS have recognized objectives, a designated manager, and resources for the SoS; however, the constituent systems retain their independent ownership, objectives, funding, and development and sustainment approaches. Changes in the systems are based on collaboration between the SoS and the system.
- **Directed.** Directed SoS are those in which the integrated SoS is built and managed to fulfill specific purposes. It is centrally managed during long-term operation to continue to fulfill those purposes as well as any new ones the system owners might wish to address. The component systems maintain an ability to operate independently, but their normal operational mode is subordinated to the centrally managed purpose.

The Future Combat Systems program was an example of a rare DoD-directed system, with its vehicles, weapons systems, and command and control systems coordinated and designed with a common objective and central authority. Acknowledged systems, however, are becoming more common in the defense sector; unfortunately, they lack central control of the constituent systems; each maintains its own objectives, management, funding, and development process.

2.4.2 SoS Considerations

The networked-based, multi-supplier, rapidly evolving software development environment, along with the complexity of inter-constituent operation and communication, can lead to poorly defined and/or highly variable information and control requirements throughout the lifecycle. The layers of governance in such situations are often an amalgam of all four types. This complicates and elevates risks in CI/CD if testing and integration are not continuous across all areas where the development product interacts with other components of the system or the SoS.

If the constituent systems are independently evolving, the concepts of continuous testing and continuous integration become much more difficult to manage and synchronize. This gets particularly challenging if there are intellectual property conflicts or multi-supplier competitions. Issues can also arise with multiple independent DSO pipelines. Continuous integration and testing must be managed so that developers can be sure their code is compatible with, and sufficient for, the overall system behavior. While a similar situation can arise where multiple independent development pipelines support a single system, there is usually an overall system authority who can mitigate most issues.

Several industries beyond defense have identified DSO in SoS issues, including telecommunications network providers, financial service providers, and large commercial research organizations [Martinez 2018, Fazal-Baqaie 2017, McCarthy 2015]. More information on managing SoS is available in *System-of-Systems Navigator: An Approach for Managing System-of-Systems Interoperability* [Brownsword 2006].

2.4.3 SoS Integrated Testing

In every SoS there is an interdependence among systems forming the foundation of successful operations. The existent interdependence among systems in an SoS requires extensive testing of data flows across various systems and data processing, input, and output within those systems. These data flows are the essential components assuring successful operation.

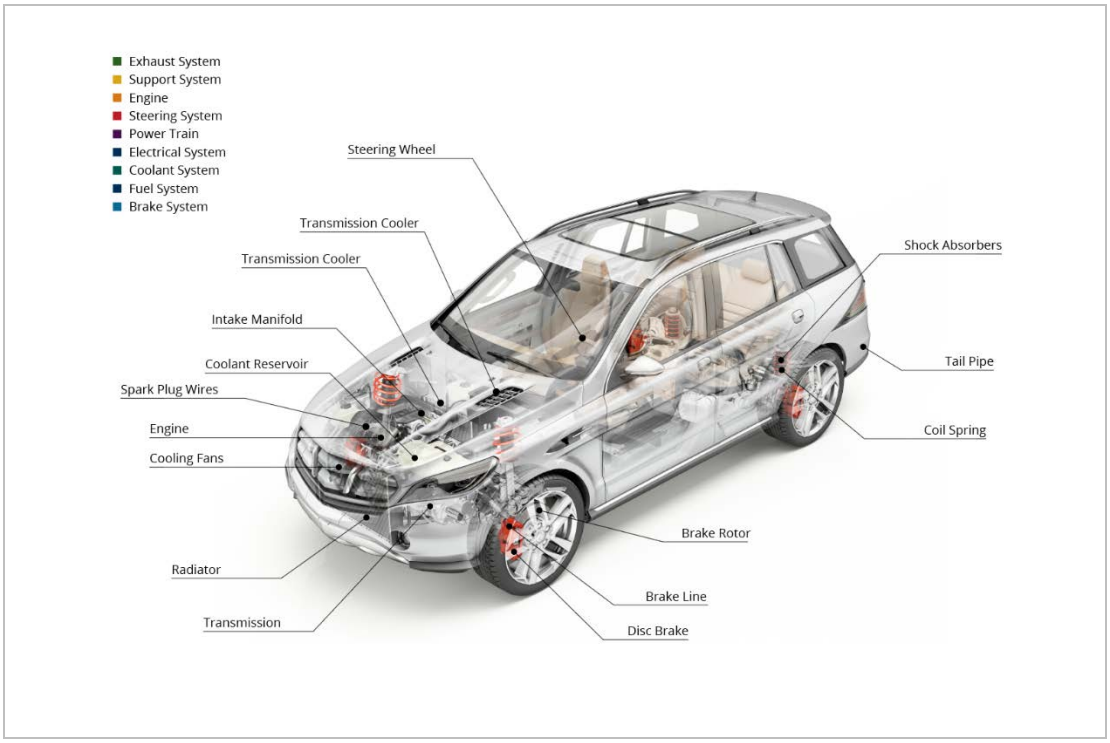


Figure 8: A Vehicle as an Embedded SoS (https://schoolworkhelper.net/vehicle-systems-overview/)

In Figure 8, we present a simplified general diagram of the various sub-systems and corresponding components found in a modern-day automobile. This is an example of an embedded system, which is the first form of SoS as previously discussed. The corporate IT network infrastructure in Figure 9 is an example of another form of SoS, as previously discussed. This figure illustrates the complex data flows that occur between sensors and various systems for the purposes of security, driver awareness, and third-party assistance. One can appreciate how data flows from original sources (such as a disc brake sensor) through the brake system and onward to other areas of the vehicle. As the sensor data flows, it can be algorithmically or logically transformed into new data, such as a dashboard alert. The sensor data can simultaneously be read and stored in various other subsystems. Testing for correct usage of data in a complex flow like this is a critical challenge of SoS integration.

In performing integration tests for these data flows, we define two test types: *source value* and *derived value*. A source value is created within a single constituent system and does not involve values passed in from other systems. A derived value is created within a single system, using both source and other derived values in a logical expression or algorithmic output. Both values are created within one system called the *originating system*.

A source value integration test achieves the following goals:

1. It validates that any source value exiting a system in which it was created and routed to another system is both sent and received correctly.
2. It validates that any source value received from another system is preserved and read correctly by all requesting functionalities of the receiver system.

3. The two goals of source value integration testing rest on the assumption that the value being tested was created in one system and then passed to and manipulated by several other systems. In cases where a source value gets passed through multiple systems, the value should be tracked, and the test goals should be applied to each system.

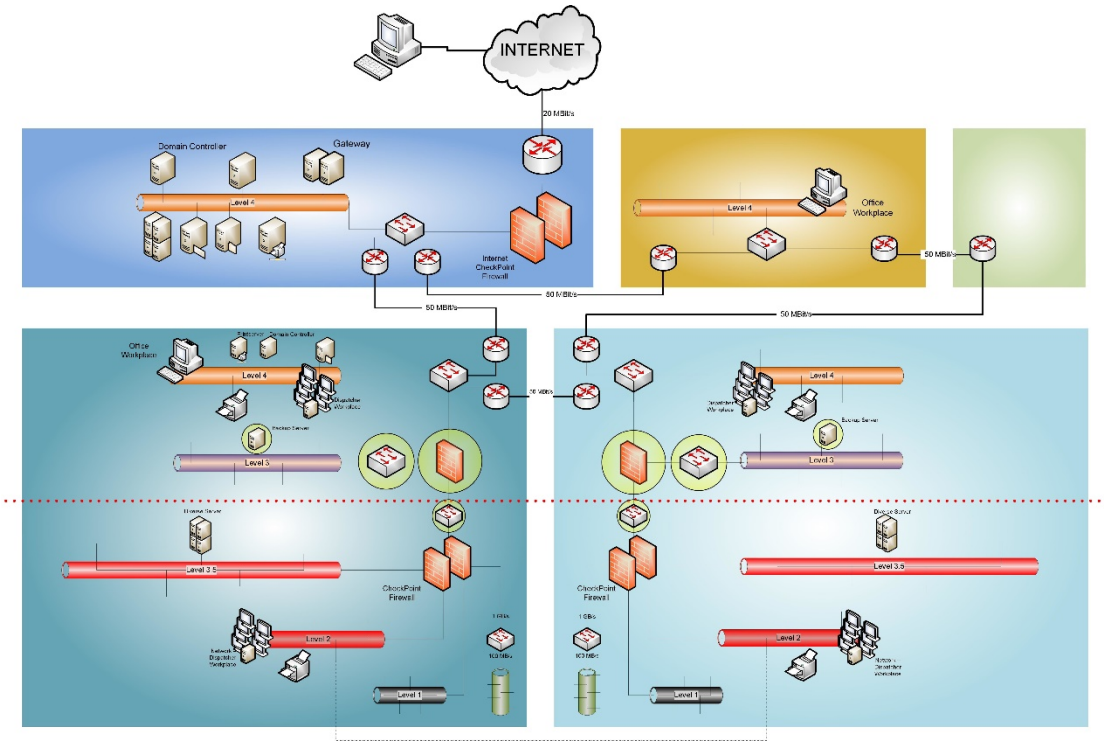


Figure 9: A Corporate IT Infrastructure as a System of Systems

A derived value integration test achieves the following goals:

1. It validates that input values are correctly used to create the current outputted derived value.
2. It validates that any derived value exiting a system in which it was created and routed to another system is both sent and received correctly.
3. It validates that any derived value received from another system is preserved and read correctly by all requesting functionalities of the receiver system.

Derived value integration testing is similar to source value integration testing. They both track the movement of a value across several systems and test for its correct usage by any functionality requesting it. Recall that both values are created within the originating system. The critical difference is that a source value is created exclusively from inputs of the operating system, runtime environment, or hardware; a derived value is a mix of system, environment, hardware, source, and other derived values. The actual mechanics of a source and derived value integration test is as follows:

1. From the moment a source value passes from its originating system through multiple other systems via parallel and sequential data flows, a test procedure should be written for each code segment in those systems using the source value as input. This can result in a set of test

procedures for each code segment use of the source value. There can be long sequential and parallel data flows through various systems by a source value. At some point, the source value is no longer needed. When this occurs, the source value is either stored, its variable overwritten, or is placed in a restful state and no longer accessed by a system. These and other similar actions mark the end of source value integration testing.

2. A derived value may flow across multiple systems beyond its originating system. A test procedure should be written for each code segment in those non-originating systems that inputs the derived value. Once the derived value is no longer needed, termination of testing is appropriate.

The mechanics of source and derived value integration testing are essentially the same. The main difference is the creation of the value. When you test source values, your tests will cover that value's lifespan from creation in the originating system through all its traversals across multiple systems, up to and including its use in creating derived values—as well as the point when it is no longer used. Note that the use of a source value in creating a derived value is a termination point in testing that path of the source value. Derived value integration testing is performed the same as source value testing.

SoS integration testing starts with either a source or derived value. The testing focuses on the value's traversal and usage across multiple non-originating systems. Along this path, in the case of a source value, it may be used to create a derived value. When this occurs, source value testing ends. Testing derived values can end when used to create other derived values. Each source and derived value are tested separately with their own set of test procedures and data. Note that source values can be part of creating derived values but the opposite should not occur.

3 Adoption Overview

As illustrated in Figure 10, there are three focus areas of DSO adoption, each with associated objectives: preparation, establishing the ecosystem, and managing and evolving the ecosystem. As discussed earlier, the adoption process should be treated like any other software development or technology adoption project. To that end, there is intentional overlap in both areas and their objectives to support an incremental, iterative, and concurrent approach to the adoption process.

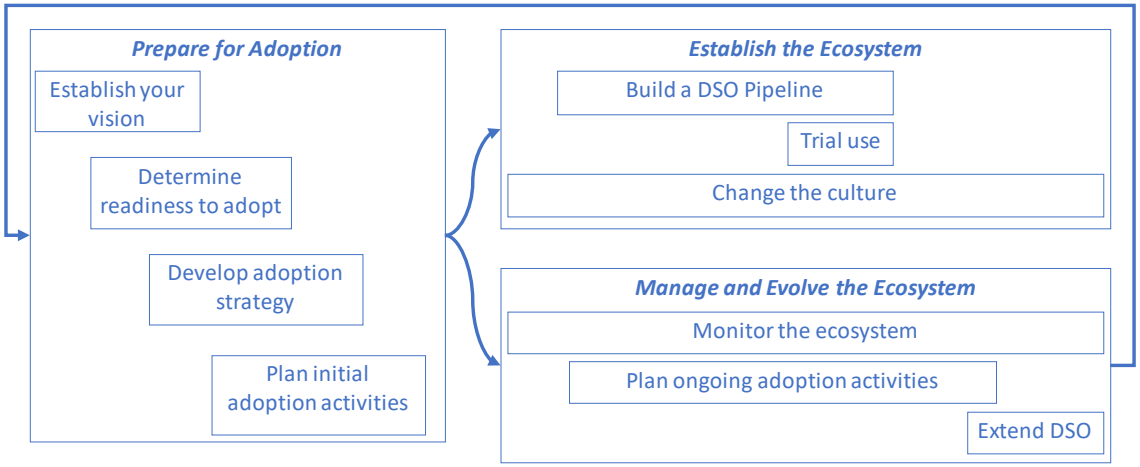


Figure 10: DSO Adoption Overview

Preparation is necessary to create achievable goals and expectations and to establish feasible increments for building the ecosystem. This includes considering the distance between where you are and where you want to be, and understanding the effort necessary to travel that path.

Establishing the ecosystem includes evolving the technology, processes, and culture from their initial state toward an initial capability.

Managing the ecosystem includes measuring and monitoring both the health of the ecosystem and the performance of the organization, evolving the system toward a desired state in a manner that

- balances speed with depth
- minimizes unnecessary rework
- adapts to changes in needs, priorities, technology, and innovation
- constantly revalidates and adapts the desired state

The next three sections describe activities for each of these stages. The activities are not necessarily sequential—some are dependent on other activities, and some may be done simultaneously. The descriptions are general enough to support adaptation to your environment.

To support quick reference to key points, each activity is first summarized in a table with content as shown in Table 1. Detailed process descriptions and additional information about the activity follow the summary. The summaries appear in Appendix A.

Table 1: Key for the Activity Summaries

Context	*How this activity relates to the overall adoption*
Purpose	*What the activity accomplishes*
Overview	*Overview of the activity*
Primary Actors	*The roles that are the most likely to be involved with this activity*
Inputs	*Necessary information to successfully accomplish the activity*
Outputs	*Information or actions that the activity produces*
Resources	*Work products, guides, and other information that could support the activity*
Tips, Tricks, and Wisdom	*Heuristics, lessons learned, and commentary from those who came before*

4 Prepare for Adoption

DSO's breadth of scope makes preparation for adoption critical, and there are three objectives that must be met. The first objective is to establish your organization's understanding of the desired outcomes (vision). Then evaluate readiness for the adoption project by understanding how your current work compares to DSO and identifying barriers and risks that will need to be addressed. Goal and strategy development can then more easily and realistically be undertaken. With goals and strategies agreed to, adoption project planning can begin in earnest, with the development of backlogs, iterations/increments, and the underlying management practices established. Figure 11 provides an overview of preparation activities.

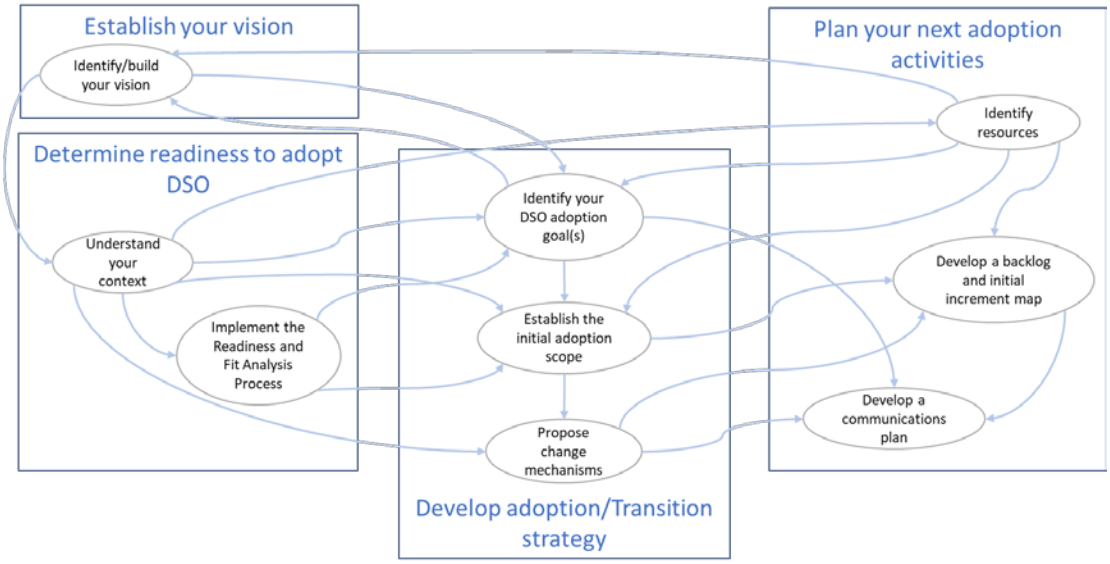


Figure 11: Overview of Preparing for Adoption

4.1 Objective: Establish Your Vision

The very first activity is to establish a common understanding of what the organization expects from the adoption of a fully implemented DSO environment. This vision is the compass that keeps the adoption on track when planning meets reality. The most effective vision is one that is shared by the enterprise. While that is rarely accomplished before adoption, the more stakeholders who feel ownership early, the more likely the vision will be shared later.

4.1.1 Activity: Identify/Build Your Vision

Context	*This activity is the first step in adoption. Without a shared vision the odds of successful adoption are significantly reduced.*
Purpose	*The purpose of this activity is to achieve a vision that supports readiness and fit analysis and adoption planning activities, maintains the long view, and is validated and agreed to by stakeholders and practitioners.*
Overview	*Develop a common understanding of a desirable outcome of the adoption activities. What will success look like?*
Primary Actors	*This activity involves leadership, management, and representative practitioners (organic or contracted).*
Inputs	*Inputs to this activity include decision information, survey questions, access to personnel, and means of capturing the information gathered.*
Outputs	*Outputs from this activity include documentation of the information gathered and the vision identified.*
Resources	
Tips, Tricks, and Wisdom	*Use visualizations (e.g., value networks, organizational structures, process diagrams) to promote participation and reduce unnecessary "wordsmithing."*

Visions are usually derived from needs, direction from higher authority, concerns of practitioners, unexpected or unplanned changes in the environment. There are many tools available for establishing and capturing a vision, but each situation will determine where the vision work is initiated. It could be a meeting to specifically address the driving issue, a consensus from the practitioners that significant change must occur, or an evolution from strategic or other broader goals.

In the introduction, we assumed a decision has been made in your organization to implement DSO. If a vision is provided with the direction, be sure that it is clear; challenge any ambiguous or unrelated elements. If there is no vision provided, then creating one is critical. Your vision should address the rationale for that decision (if available) and must create a concrete description of how that implementation is expected to be superior to the current environment.

One way to develop the vision is through a survey of key stakeholders and representative organizational personnel about their thoughts on the following generally desirable properties of their current software development environment:

1. **Length of project life.** This is the amount of time from start to finish of a project. Is it within the budgeted time or is additional time regularly required?
2. **Frequency of source code commits.** This reflects code development tasks. If a task is too large, it may not be completed and committed in one day. Multiple smaller tasks may be easier to complete and commit in a single day. Is the group satisfied with the commits per day? Does it support scheduled project completion?
3. **Quality and degree of stakeholder communication.** How well and often does this occur? Is the group able to speak with a stakeholder as often as needed or desired? What are typical response times, and are they sufficient to reduce rework?
4. **New employee onboarding time.** How much time is needed for a new employee to be productive in the group's development process?

5. **Stability of the development environment.** This relates to environment parity. Are the various environments used by developers and other HRE personnel kept consistent and up to date with tools, updates, features, and the like?
6. **Tool availability and usage.** This refers to development team tools such as chat services, versioning systems, testing and development infrastructures, automated IT request fulfillments, and other similar tools. Are these tools available to developers, and do they function effectively? Are they're multiple tool sets for the same purpose? Can the current tool set be reduced to exclude those not favored by developers?
7. **Consistent appropriate staff**. Does the HRE provide long-term, well-qualified personnel that are assigned to specific job functions such as IT infrastructure, software delivery, programmers for specific languages, unit testers, and system integration?
8. **Delivery cycles.** How often is code pushed to production for end-user feedback? Is it hourly, daily, weekly, or longer? Is delivery performed by one person or a group? Is delivery a consistent, repeatable process, or is it a unique effort each time? Is approval required? How long is the wait for approval? Does it negatively affect the delivery process?

This information can be obtained in interviews, group discussions, or any other appropriate method. Once collected, a small group should evaluate the information and build a vision that includes the emerging ideas and desires, and is structured in a useful way. The vision should then be circulated to the information providers for feedback and any significant concerns addressed.

4.2 Objective: Determine Readiness to Adopt DSO

An organization (or enterprise) is a complex, interacting socio-technical system. It has a unique set of context, culture, operating norms, communications channels, roles, and personnel. Understanding the as-is environment, how it needs to change, and the barriers and risks associated with the changes is critical to success. We recommend a readiness and fit analysis as a structured way to accomplish this objective. There is a significant amount of information to be collected to support this process, and it is called out in the first activity.

As you can see in Figure 12, all change is not equal. So, in preparing for DSO adoption, you need to have an understanding of the difficulty of the areas you need to change. More information on managing change can be found in Section 7.1.

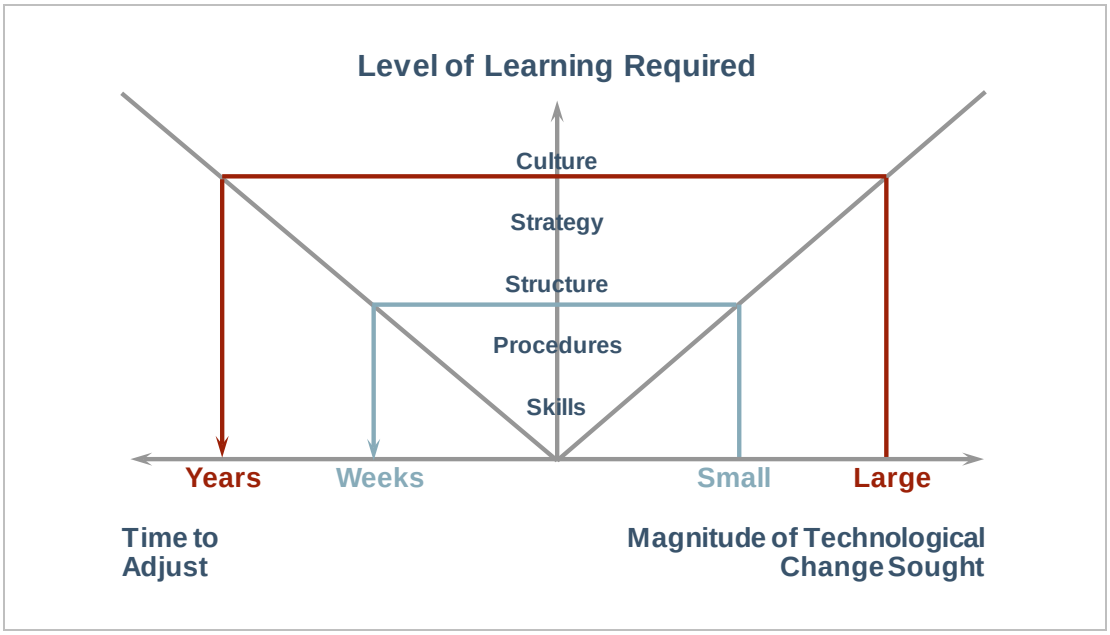


Figure 12: Difficulty of Change (Adapted from Adler and Shenhar [Adler 1990])

4.2.1 Activity: Understand Your Context

Context	*This is the first step in adoption. Understanding the current environment is critical to how this activity relates to overall adoption.*
Purpose	*This activity supports readiness, fit analysis, and adoption planning activities.*
Overview	*Collect and validate information about context.*
Primary Actors	*This activity involves everyone.*
Inputs	*Inputs to this activity include access to personnel and means of capturing information gathered.*
Outputs	*Outputs from this activity include documentation of the information gathered.*
Resources	*Section 7.4*
Tips, Tricks, and Wisdom	*Using visualizations (e.g., value networks, organizational structures, process diagrams), promote participation and reduce unnecessary "wordsmithing."*

Knowing where your team or organization fits within the enterprise often varies by individual; it is colored by the practitioner's role, background, and often their longevity in the organization.

4.2.1.1 Organizational Context

Organizational context determines the various stakeholders, communications networks, programs, and chains of command that influence the work your organization does. There are several ways to create a common understanding, but one of the most effective is creating a value stream or value network. This is a visual representation of how value flows through the development, delivery, and operational groups of your organization and the first- and second-order stakeholders for the

products it produces. More details about value stream and value network development can be found in Section 7.

4.2.1.2 Cultural Context

Culture is another aspect of context. It a significant factor in gaining DSO benefits. Understanding the current culture allows the organization to map out the changes that are needed and have a baseline against which to gauge progress.

Culture change is hard, and success depends on aligning all DSO-related activities with understood cultural norms. Tools and practices are necessary but insufficient if the cultural norms are not in place and not constantly reinforced. There are many ways to measure culture, but every organization is unique in some way that may make any given approach difficult. In general, the context drives the measurement approach. Do you have time for a canvas or survey of all the people involved? If not, are you comfortable that you can cover the differences with a sampling of input? Do you need to have the input non-attributable, or can you use staff meetings or other gatherings to collect data?

Culture change requires champions; so identify people who will create a sense of urgency, form a powerful coalition, and create a vision.

The following sections present statements about the aspects of your culture that are directly related to the DSO ecosystem. As you read through these statements, consider each for either inapplicability or a sense that they may be of special concern.

4.2.1.3 Communications Maturity

- Mechanisms are in place in the contract and acquisition strategy to allow close collaboration between developers and end users.
- Sponsor support for Lean and Agile/DevOps methods use is explicit and cascading. (It isn't just the program manager; sponsorship cascades throughout the acquisition chain.)
- The operational need for close collaboration among developers and users is explicit and clear.
- Senior stakeholders openly and explicitly support the use of Lean/Agile/DevOps methods in the program.
- The environment supports increasing automation.

4.2.1.3.1 Personal Relationship Maturity

1. A "fail early, fail fast, and learn" philosophy is supported by the organization in which development occurs.
2. The organization supports a climate of trust.
3. Management is a coaching function (as opposed to traditional command-and-control) that helps to eliminate barriers to progress.
4. The team is made up of task-mature individuals operating in high-trust groups.

4.2.1.3.2 Integration and Collaboration Levels

1. A “many-hats” culture exists among development and operations roles.
2. The organization supports direct collaboration among development, test, and operations teams.
3. The organization provides the physical and social environments needed for team success.
4. The organization supports the early and frequent delivery of potentially shippable software to customers and end users.
5. Sponsors understand and support the difference in roles and business rhythm when using DSO.
6. The organization's change history for introducing new engineering and management approaches is recently positive.
7. The testing and evaluation of the product occurs frequently (every code commit) and regularly, and is expected.
8. Management understands and advocates for the extra time/effort to ramp up DSO techniques within the organization.

4.2.1.4 Understand System and Architecture Characteristics

Understanding the current state of your technical environment and the technical challenges the future might hold informs how you define and implement your DSO adoption approach. DSO is significantly influenced by the infrastructure of the development system as well as the architecture of the product (applications, systems, SoSs) under development and in operation. More information on the effects of architecture on DSO (including Lean and Agile approaches) is available in Section 7.3.1.

4.2.1.5 Identify Security Considerations

This security perspective is primarily derived from the data and code being accessed in the pipeline’s various components. Consideration must be given to projects with artifacts associated with an elevated impact level or a security classification. Available secure networking and communications associated with the development should also be described, including how they are currently accessed and gaps between the current and the required infrastructure.

Considering the applicability of the statements below helps you determine your security needs:

1. Some components of the project are classified.
2. The project will be developed or delivered in a classified space.
3. Some parts of the project have specific access restrictions.
4. Some parts of the project are considered controlled information other than being classified (such as ITAR).
5. There are specific access requirements (such as citizenship) for project developers.
6. Part of the pipeline will exist in open space while the rest is in a controlled or classified environment.
7. Mechanisms for connectivity between open and controlled portions of the pipeline or project must be established.

8. Mechanisms to handle code contributed by "less trusted" partners must be established.

4.2.1.6 Inventory Technical Platforms and Software Assets

Performing an inventory of your platforms and environments helps to identify and leverage existing assets for DevOps adoption and provides a basis for planning budget and schedule requirements for DevOps adoption.

Consider the applicability of these statements in creating your inventory:

1. A development platform and/or environment exists and has been used on multiple projects already.
2. Mechanisms are in place for using multiple different operating systems in development and/or testing.
3. Workflow management, document repositories, messaging services, or other DevOps project-management related tools are in place.
4. The project is developing platforms and environments or using unmodified COTS.
5. Some software development assets are hosted in the cloud or some other location not in-house or under our administration.
6. Testing or enforcing environment parity is performed in current development environments.
7. Open source and/or licensed tools are effectively managed.
8. There is a need for managing assets unique to us or used by others in our organization.
9. There are unique assets used across the entire organization.

4.2.1.7 Identify Success-Critical Stakeholders and Adopters

The stakeholders in a DSO environment stretch across both technical and organizational roles, and each environment has its own set. In DSO adoption, these groups might include security; IT; systems engineering; verification, validation, and transition; acquisition; legal; contracting; policy; external certifications; and, of course, users.

4.2.1.8 Identify Software Development Roles

An important part of adoption preparation is identifying the roles of the people who will work on the DSO-adoption project and the roles of the performers of the DSO processes and pipeline. The people with one or more roles need to be trained on how to use the pipeline to fulfill their responsibilities. Table 2 provides a sample of common roles, typical responsibilities, and the pipeline components they will likely interact with.

Table 2 includes a mix of technical and non-technical roles. The listed pipeline components are described in Table 6. Table 2 illustrates that, in DevSecOps, all stakeholders may access the pipeline, and access is not limited to only technical roles. For each of the roles defined, instructions must be provided to the user. These instructions are part of the ‘Process and Practices’ dimension of DevSecOps. In Section 5.2.3, we describe the mechanics of documenting the process for various role types.

Table 2: Roles, Responsibilities, and Pipeline Interactions

Role	Responsibilities	Pipeline Components
Program Manager	Continuously manages throughout the program lifecycle Plans the overall program and monitors progress Manages budget, risks, and issues, and takes corrective actions	Workflow management system Document repository Monitoring service Performance metrics
Pipeline Architect	Leads the technical design, development, and evolution of the pipeline	All
Culture Change Coach	Plans, organizes, coordinates, facilitates, and reports on culture change activities and progress	Access to measures, processes, and retrospectives Workflow management system
Customer	Interacts with the software in the operational environment	Workflow management system Document repository production environment
End User	Benefits from interacting with the delivered system in production	Production Document repository Workflow management system
Software Engineer	Writes code based on requirements Tests and delivers programs and systems Fixes and improves existing software	Integrated development environment (IDE) Version control repository Workflow management system
Requirements Engineer	Works with stakeholders to elicit, understand, analyze, and document requirements for a project	Document repository Workflow management system Staging and production
Test Engineer	Creates and documents test cases Performs and documents risk analysis Codes and runs automated tests Determines product quality and release readiness	Build server Staging and production
Operations Engineer	Operates by accessing software on computers Monitors and manipulates daily system jobs Starts operations by entering commands Performs defined tasks per documented instructions/processes	Build server Staging and production Dependencies server Provisioning server
Security Engineer	Performs security testing and code review to improve software security	Build server Staging and production

4.2.1.9 Understand Current Process and Practice Characteristics

Most organizations operate within a reasonably consistent environment. Consider which of the following descriptions most closely match your existing environment. The lower the number, the more change will be required for full adoption; you will need to manage multiple change initiatives during adoption, and make sure that the enabling managers and executives fully understand the scope of the change required.

1. **Traditional Waterfall Environment:** One-time through the process; late integration and testing; Systems Engineering V-model with traditional milestones
2. **Early Lean-Agile Environment:** Mostly waterfall with Lean-Agile pilots in place; some Agile champions/sponsors
3. **Mature Lean-Agile Environment:** Mostly Lean-Agile with a few remnants of waterfall; leadership and management fully supportive of Lean-Agile
4. **Existing DevOps Environment:** One or more projects using collaborative development and operations decision making and CI/CD pipelines

4.2.1.10 Assess Initial DSO Posture

This is an initial assessment that provides a baseline for measuring technical progress throughout the DSO-adoption process. Information is gathered through discussions with a broader set of team members and captures a technical picture of the organization at the beginning of DSO adoption.

A critical component of the adoption process is the ability to measure, or at a minimum demonstrate, progress of SDLC DevSecOps implementation of a group in an HRE via comparison with a fully implemented ideal DevOps SDLC process before and after an assessment. The HRE personnel should describe their ideal DevSecOps SDLC. The associated measurement will likely be subjective; each group may have their own view on the ideal DevOps SDLC process. Topics considered in a DSO posture assessment include

1. The means and frequency of stakeholder communication
2. How requirements for software are received and when
3. How artifacts are delivered
4. How access to artifacts, documents, and project status is provided to HRE external stakeholders
5. How feedback moves from stakeholders and end users in the production environment to developers
6. Developers' access to staging environments
7. Authority required to make artifact modification during and after initial delivery
8. How projects receive authorization to commence work
9. Bottlenecks causing delays to project commencement and completion
10. The hardware acquisition process
11. Management of non-HRE development activities contributing to the project

4.2.2 Activity: Implement the Readiness and Fit Analysis Process

Context	*This activity establishes the enablers and barriers in your organization associated with adopting DSO. Having the value map and profiles make this an easier task. While this can produce significant concern, particularly if the barriers outweigh the enablers, it is critical to manage expectations and conduct rational planning.*
Purpose	*This activity captures the current organization's readiness to adopt DSO in terms of risks, opportunities, barriers, and enablers. It is a significant planning asset.*
Primary Actors	*This activity involves the manager, teams, S-CSs, and the culture change coach.*
Relevant/Key Events	*Events include the decision to adopt DSO and a DSO Posture Assessment.*
Activity Input(s)	*Inputs to this activity include the results of 4.1.1 and 4.2.1.*
Activity Outputs(s)	*Outputs from this activity include an adoption risk assessment with identified mitigation approaches and proposed adoption progress measures.*
Other Resources	*Other resources include the RFA White Paper [Miller 2014], RFA Presentation slides, and RFA Forms [Miller 2014].*
Tips, Tricks, and Wisdom	*A workshop approach to this analysis is faster but requires more coordination.* *It is just as important to identify enablers as risks.*

Gather key participants in the current process plus Agile/DevSecOps champions. The leader takes the group through each of the dimensions, asking the provided questions and discussing them.

During the group discussions, the following activities take place:

- Each participant records their perception of the DevSecOps "fit" for their environment to support a culture profile.
- Each participant listens and considers if there are particular risks or opportunities (things that might happen), or issues (things that are happening now) in this topic area.
- If present, the participant records the risks, opportunities, or issues, one per sticky note, in the form of "Given that (*condition*), there is a possibility (*risk* or *opportunity*) that (*consequences* or *benefits*)" or describes the issue(s) in a brief statement.
- All of the sticky notes are posted, arranged in affinity groups, and then summarized to develop a set of risk/opportunity/issue statements.

More information is available in Suzanne Miller's 2014 white paper, *The Readiness & Fit Analysis: Is Your Organization Ready for Agile?* [Miller 2014].

Table 3: Readiness and Fit Analysis Assumptions for DevSecOps

Fit Dimension	Agile and DevOps Assumptions
Business and Acquisition	The program acquisition strategy and practices enable, or at least don't disable, differences in developing and deploying Agile and DevOps approaches.
Organizational Climate	Reward systems, values, skills, and sponsorship explicitly support Agile and DevOps values and principles.
Project, Team, and Customer Environment	Frequent collaboration between the development team and test, operations, customers, and end users is actively supported. Program management practices don't force teams to work across different projects.
System Attributes	System architecture is loosely coupled. (Interfaces are external vs. internal among system components.) System solutions benefit from fast user/operational feedback.
Technology Environment	Technology support for virtualization, automated testing, and continuous integration are in place. An integrated collaboration platform is in place, including monitoring and feedback.
Team Technical Practices (subset of Practices)	Technical practices that support high-quality code production in small batches from a prioritized product backlog are in place. Technical practices integrate automated testing and integration.
Team Management/ Coordination Practices (subset of Practices)	Decentralized decision making that allows team members to self-organize their work are in place and supported. Team management practices that support short (2-4 week or less) time boxes are in place. Coordination practices among development, test, and operations stakeholders are routinely used.
Program Practices (subset of Practices)	Synchronization of multiple teams is occurring. Practices that reinforce respecting team management and measurement boundaries are in place. Automated governance mechanisms are used where appropriate.

4.3 Objective: Develop an Adoption/Transition Strategy

An overall strategy is needed to be able to begin more specific planning. Having captured the existing context and a steering vision, you can begin to identify specific goals and build a multi-increment strategy for achieving them. This strategy is an opportunity to engage with the success-critical stakeholders to

- show how their needs are addressed in DSO adoption
- clarify expectations
- create a management infrastructure
- address required resources in terms of funding, personnel, and process adaptation

4.3.1 Activity: Identify Your DSO Adoption Goal(s)

The goals and measures you create in this activity will drive the initial stages of adoption. Parsimony is important; don't create so many goals that the impact of the most valuable goals is diluted. Understanding the needs and concerns of the S-CSs can help couch the goals in language that can be easily understood and agreed to. Goal creation is often an iterative process, with the need for rapid feedback.

4.3.1.1 Defining Goals

Context	***The DSO adoption goals are the core guidance for all of your strategic and tactical planning. They continue to be evaluated and evolved throughout the adoption and management process.***
Purpose	***This activity produces a set of goals aligned with the DSO principles that identify the specific outcomes desired from the adoption of DSO along with broad indicators of accomplishment.***
Primary Actors	***This activity involves team lead(s), customers/users, developers, Operations-Deployment, IT, Security, and the pipeline (PL) architect.***
Activity Inputs	***Inputs to this activity include a list of success-critical stakeholders (S-CSs).***
Activity Outputs	***Outputs from this activity include the initial goals statement.***
Other Resources	***Other resources include the blog post DevOps and Your Organization: Where to Begin (*** https://insights.sei.cmu.edu/devops/2014/12/devops-and-your-organization-where-to-begin.html***) and the webinar Three Secrets to Successful Agile Metrics (https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=507850).***
Tips, Tricks, and Wisdom	***The goals can (and most likely will) evolve as the adoption progresses. Give priority to cultural outcomes and stakeholder pain points. Review the goals and their measures regularly.***

The goals that you set have a significant effect on what activities you undertake, how you decide to measure them, and what kinds of attitudes and behaviors you incentivize and discourage. So, setting and evolving *useful* goals is a priority in implementing DSO.

The SMART goals model is a well-defined approach for developing *useful* goals:

1. *Specific.* It's something where successful completion can clearly be determined.
2. *Measurable.* The measure could be a specific value (average 500 widgets/day over three months), binary (yes/no), or scaled (10 percent versus 50 percent), but the measure must be appropriate for the goal.
3. *Achievable/Attainable.* It's something you can actually do something about.
4. *Realistic/Relevant.* Even though it may be a stretch, it's something you truly believe is within the capabilities of your staff and the constraints of your environment, and is something whose achievement will be beneficial to you.
5. *Time-Based/Tangible.* For some goals, a time factor is necessary; otherwise, the goal is overcome by events. Goals that are not time based should be tangible and observable, so that an objective evaluation of its satisfaction is feasible.

The Balanced Scorecard concept is used in many industries to provide a way of segmenting your goals so that no single aspect of your adoption gets exclusive focus. The scorecard is usually presented as a four-quadrant matrix. DSO headings could include

1. *People* goals relate to establishing a DSO culture.
2. *Process* goals relate to establishing governance and technical processes that support DSO.
3. *Technology* goals relate to automation to support development, operations, deployment, and security.
4. *Learning and Innovation* goals relate to improving the distinctive competencies of the organization and making it more responsive to the relevant changes in its technical and mission environment.

4.3.1.2 Defining Measures

There are many resources on measuring technical- and project-related progress. The following discuss those measures that may be useful in your measurement strategy.

Cultural Goals/Progress Measures. For culture change, the measures are usually specific to removing identified risks or are measures of the degree of adoption. The measures to understand progress in this important factor are usually measures for diffusion (i.e., how broadly a new technology has reached within the organization) and infusion (i.e., how deeply a new technology has penetrated into the intended organizational culture) [Adler 1990, Leonard-Barton 1988, Miller 2014, Miller 2006]. Additional measures can be derived from surveys of primary users and other stakeholders about their experiences with DSO implementation and outcomes.

Operational Goals/Progress Measures. These measures are well represented in the Lean and Agile literature. They tend to be created out of the automation in many DSO implementations and cover flow, quality, defect escape, and other specific technical aspects.

Organizational Goals/Progress Measures. Organizational change is similar to cultural change; it addresses larger impacts. Examples are measures of worker and customer satisfaction; overall throughput and cycle time; and efficiency, effectiveness, and resilience.

4.3.1.3 Example DSO Goals

The following are adapted from the Joint Improvised-Threat Defeat Organization (JIDO) *SecDevOps Concept of Operations* [DTRA 2007]:

Collaboration: People + Process

- Every process participant understands the entire process and their contribution to it.

Automation: Process + Technology

- Technology supports the process.
- Technology eliminates repetitive or tedious tasks
- Quality assurance (QA) is automatically tool enforced at various steps to provide an enterprise Secure Development Operations (SecDevOps) approach.

Analysis: Technology + People

- Technology improves workflow and the analysis of bottlenecks to improve results with cross-functional skills.

4.3.2 Activity: Establish the Initial Adoption Scope

Context	*The DSO adoption goals are most likely visions for the future. There needs to be an identified scope for initial adoption. Is it one team, one organization, or an enterprise? The answer to this question will determine how you will size the increments and will be highly dependent on the resources available over time.*
Purpose	*This activity identifies the specific goals to be addressed in the current adoption effort.*
Primary Actors	*This activity involves team lead(s), customers/users, developers, Operations-Deployment, IT, Security, and the pipeline (PL) architect.*
Activity Inputs	*Inputs to this activity include the overall adoption goals.*
Activity Outputs	*Outputs from this activity include specific goals to be addressed in the initial effort.*
Other Resources	*Other resources include Section 7.1.4 and the CMMI Survival Guide [Miller 2006].*
Tips, Tricks, and Wisdom	*This is where understanding technical feasibility pilots and adoption feasibility pilots can be useful.*

Once you have a sense of context and you have characterized your goals and measures, you can begin to consider the scope of your adoption. As stated in this report's introduction, an adoption effort is very similar to a development effort, and using some form of iterative or incremental approach is recommended. That said, one way of scoping your effort is to consider the amount of change you believe your organizational culture can survive and the size of the effort in terms of the organization. These may not be under your control, but they will definitely need to be understood before you can begin the adoption effort.

4.3.2.1 Identify Culture Changes Required

The distance between your current organizational culture and a DSO culture can seem immense—particularly if you are still following traditional waterfall practices. Luckily, the establishment of a pipeline reinforces the changes in culture that are most important. Compare your current culture with a DSO culture. Identify the critical things that need to change. Determine if there are already activities in any area and, if so, leverage them to the fullest. Are there simple things that can be started to support more complex things? Techniques like maintaining backlogs, limiting work in progress, and holding daily standup meetings are reasonably easy to implement, and if implemented correctly, can act as a springboard for other changes.

Another option is using the pipeline to approach culture change in an iterative manner. You may decide on all the components for your pipeline at the beginning but choose to implement them in an order that makes sense to your processes and culture. Start with activities that provide the most interaction among the developers, operators, and security to establish the collaboration patterns that are central to DSO.

4.3.2.2 Identify Organizational Scope

In most cases, adoption happens on an initial project or within a particular product team. This is the path we suggest, but there still needs to be a decision up front about the organizational scope for the current adoption effort. As indicated, the initial scope is nearly always one team/product.

Once that has been accomplished and lessons learned and reflections have been addressed in the processes, it is time to extend the scope. It may be that the next step is to consider projects related to the initial project as the next target. On the other hand, it may be that there is another type of project, significantly different from the first, that makes more sense as you begin to tune DSO to your enterprise. In your planning, you can walk through these options early on, make some preliminary decisions, and then revisit the issue over time based on the initial experience.

4.3.3 Activity: Propose Change (Transition) Mechanisms

Context	***Change is not a passive activity. There must be specific actions taken to reach out to stakeholders and practitioners to enable and reinforce change.***
Purpose	***This activity produces a set of transition mechanisms that are tailored to the scope and target of the adoption effort.***
Primary Actors	***This activity involves team lead(s), customers/users, developers, Operations-Deployment, IT, Security, and the pipeline (PL) architect.***
Activity Inputs	***Inputs to this activity include specific (scoped) goals.***
Activity Outputs	***Outputs from this activity include a set of communication and implementation mechanisms and actions that propel changes in technology and culture.***
Other Resources	***Other resources include the CMMI Survival Guide [Miller 2006] and Section 7.1.3.***
Tips, Tricks, and Wisdom	***Two types of failure modes are frequently seen: (1) focusing only on communication mechanisms (the "train people to death" failure mode) and (2) the opposite—providing new procedures, measures, and other implementation mechanisms before enough communication has occurred for staff to understand what the goal of the adoption is.***

Transition mechanisms are events or job aids that support communication and implementation activities. Table 4 illustrates a small subset of mechanisms available, and can help you with timing your development of guidance and other work products that need to be developed or acquired.

Table 4: Typical Transmission Mechanisms by Adoption Commitment Curve Stages [Adler 1990]

Commitment Stage	Typical Mechanisms
Contact and awareness	"Elevator speech" Standard 45-minute pitch; road show FAQ Blog posts Short "testimonial" briefings or whiteboard talks Conference briefings Online training assets
Understanding	One-day seminars, symposia for various vendors Identified stakeholder roles, responsibilities, and interrelationships
Trial Use	Pilot programs Carefully identified focused pilots (or experiments) Defined incentives for pilot participation Small working group to support pilots

Commitment Stage	Typical Mechanisms
	Defined measures of success Two- to three-day course for pilots and interested others
Adoption	Strong set of incentives; rewards, and consequences Education: mature courses, modularized for just-in-time (JIT) delivery Policies or standards
Institutionalization	Fully realized curriculum of training for different types of users New-employee training/orientation

4.3.4 Example: JIDO DSO Strategy Summary

The following is taken directly from the JIDO *SecDevOps Concept of Operations:*

> *This section, framed with People, Process, and Technology as its core components, describes in strategic terms what a transition to SecDevOps entails.*
>
> *Engaged stakeholders, daily engagements, and constant communications facilitate rapid application development that incorporates compliance with DoD STIGs and security policy requirements through the development process. Teams and team members must work collaboratively instead of procedurally to remove organizational siloes. To achieve this shift in team interaction, SecDevOps proposes a new centralized workflow management platform to organize team roles and responsibilities. An accelerated development and release cycle ensures that stakeholders are continually engaged and do not lose track of ongoing efforts. Teams that adopt SecDevOps will experience a major shift in culture that mimics the rapid tempo of ever-shifting customer needs.*
>
> *Successful implementation of SecDevOps enables true ongoing risk management through CI, CD, continuous deployment, and continuous monitoring. A process built around quick and incremental releases ensures that projects remain manageable and closely tied to schedule. Furthermore, as customer needs shift in real time, incremental development pivots to meet new demands with low latency. The SecDevOps process is well documented, fast, and repeatable, meaning that it scales to the enterprise level with less of the traditional growing pains of new methodology adoption. The adaptive process mirrors an adaptive workforce and team structure; the two complement and enhance one another.*
>
> *A detailed and technologically sophisticated CONOPS empowers the workforce by minimizing the manual labor required to shepherd a change to the production environment. SecDevOps begins and is constantly driven by a focus on automation. From the deployment of critical development infrastructure to code quality checks and continuous monitoring, the need for human interaction is refocused to core development practices. This not only increases efficiency in expensive labor hour allocation, but ensures that large parts of the process are consistent across teams, locations, and projects. Advanced open source tools prevent vendor lock, intrinsically promote collaboration, and allow for faster and cheaper technological adaptation to changing demand signals.*
>
> *DevOps places speed and flexibility at its core and is continuously mindful of customer demands and project resources. The addition of cybersecurity as a new but equally important*

and constant element demonstrates considerable value and applicability to all DoD organizations regardless of size or mission. Thus, the DevSecOps methodology allows IT organizations to build and shift on the fly, without the need to decide on a cumbersome grand strategy that cannot rapidly meet changing demands and environments. The end result is an evolution of core mission and Mission IT practices that more rapidly and more precisely meet the unprecedented needs of today's warfighter [DTRA 2007].

4.4 Objective: Plan Your Next Adoption Activities

DSO adoption is not a sequential process. Like software development, it is iterative and evolutionary. Don't attempt a "one-step to glory" waterfall of typical milestones and baselines. Rather, tailor the strategy and plan artifacts for your specific situation. The overall complexity of your environment, the flexibility and commitment of the success-critical stakeholders, and the scope of your strategy all contribute to the detail required in this planning effort and how it will be captured and evolved. It may consist of a short backlog to accomplish or several epic-level activities. Following Lean and Agile planning methods is generally effective, even if it is a new experience for your organization. Conversion to Lean and Agile approaches is a critical success factor to implementing and maintaining DSO benefits.

4.4.1 Activity: Identify Resources

Context	***Plans without appropriate resources are worthless. Iterative planning based on realistic availabilities is necessary for success.***
Purpose	***This activity identifies the resources (e.g., skilled staff, facilities, materials) that are needed to accomplish the agreed-upon goals.***
Primary Actors	***This activity involves team lead(s), customers/users, developers, Operations-Deployment, IT, Security, the pipeline (PL) architect, and the financier.***
Activity Inputs	***Inputs to this activity include the identified scope and DSO adoption goals.***
Activity Outputs	***Outputs of this activity include a list of resources and when and for how long they are needed.***
Other Resources	
Tips, Tricks, and Wisdom	***The amount of "free stuff" on the Internet in topic areas related to DSO is staggering. Don't ignore free training and communication resources that are relevant to your setting.***

The resources needed go beyond technical work by people, activities, and time. You need to factor in additional interactions with the other organizations and influencers that were represented in the value network. It is likely that software, hardware, and services will need to be acquired, and critical stakeholders need to be associated with the infrastructure and overall lifecycle planning. Are there appropriate communications media and infrastructure to handle the needs of the DSO environment's extensive and continuous collaboration between the development, security, and operational personnel and their tools? Do current team members have the skills needed? Is intensive training or personnel action required? While most of the work can be done by team members, you may need a coach or specialist for some special analyses. Use the strategy document and other artifacts that have been created to provide ranges for the various costs envisioned, and keep in mind the time required for staffing actions or training.

4.4.2 Activity: Develop a Backlog and Initial Increment Map

Context	*There is nothing like using the techniques you are espousing to help your team and organization understand that this effort is serious.*
Purpose	*This activity produces an initial backlog of items that need to be accomplished within (1) the next increment and (2) a breakdown of those items into those that can be accomplished in the next iteration.*
Primary Actors	*This activity involves team lead(s), customers/users, developers, Operations-Deployment, IT, Security, and the pipeline (PL) architect.*
Activity Inputs	*Inputs to this activity include the adoption strategy, adoption goals, and adoption resources.*
Activity Outputs	*Outputs from this activity include a roadmap for the next increment and the backlog of both high-level and more granular product backlog items.*
Other Resources	*Other resources include Section 7.1, Appendix B, and any number of books or websites describing fundamental Lean and Agile software development practices.*
Tips, Tricks, and Wisdom	*User stories are used in some settings; they may work if the environment is already accustomed to them, but may be awkward for some service-oriented tasks.*

Using Lean and Agile techniques support an ongoing process that can more easily adapt to changes in the environment. The initial plan, however, should provide a sense of the time required for the scope desired. Estimates of completion by the month or quarter could be sufficient for this initial phase. Planning at the team level should be done incrementally and at a short cadence or continuously at the practitioner level. Remember to include time for the creation and use of the culture-change mechanisms that were identified in the strategy. Assume there will be an initial drop in productivity until the changes can be assimilated by the staff and the stakeholders. Practice expectations management in your higher level planning, so there are no significant disconnects.

4.4.3 Activity: Develop a Communications Plan

Context	*Communication is the lifeblood of change management. Without a well-thought-out plan for how information is distributed, how questions are answered, and how progress and stories are captured, inertia will take over and the adoption will fail.*
Purpose	*This activity produces a communication plan tailored to the adoption activities and the environment and culture of the change target.*
Primary Actors	*This activity involves team lead(s), customers/users, developers, Operations-Deployment, IT, Security, and the pipeline (PL) architect.*
Activity Inputs	*Inputs to this activity include vision goals, scoped goals, and RFA results.*
Activity Outputs	*Outputs from this activity include the communications plan.*
Other Resources	*Other resources include the blog post DevOps and Your Organization: Where to Begin (*https://insights.sei.cmu.edu/devops/2014/12/devops-and-your-organization-where-to-begin.html*)*
Tips, Tricks, and Wisdom	

As with any human endeavor, good communication is required for anything to be achieved. Communication among humans is tricky. Thinking about and capturing what you need to

communicate and how to get it across to various audiences is important; communication supports consistency and helps maintain a sense of common vision as you discuss changes and their value across the organization.

The first chapters of Alistair Cockburn's *Agile Software Development* focus entirely on how humans communicate [Cockburn 2002]. His insights are based on keen observation of technical activities in several countries. These are some high points:

- *People parse complex experiences in very different ways*. In general, we all perceive information in somewhat different orders. We "parse" it (break it into little pieces) and then reconstruct it according to the patterns we recognize.
- *Understanding includes internal information restructuring and shared experience*. We understand by developing models of what we're hearing and constantly updating them as more facts or descriptions are gathered. A corollary to this is that group communication is based heavily on shared experiences and or terms (think acronym hell).
- *The three stages of learning behavior—following, detaching, and fluent—are critical in communicating.* People in the first stage, following, are ready to hear about one thing that works. In the second phase, detaching, people parse an idea and look for places where it doesn't hold true or can't work. When fluent, people generally are unaware they are following any pattern; they understand the desired end effect and move toward it based on their integration of experience.

Developing a simple but effective communication plan is not hard. It is significantly more difficult to manage and execute. One widely used and generally successful way to define a communication strategy is to fill in a table that maps objectives to activities. Table 5 provides the headings and a description of the information needed.

Table 5: Communication Plan in Tabular Form [Miller 2006]

Objective	Responsibility to Report Information	Member(s) Receiving Information	Receiving Information Mechanism	Medium Used	Frequency
Purpose the communication is meant to achieve	*Who needs to make sure it gets communicated*	*Who needs to get it*	*What the communication event/artifact is called*	*How the material should be transmitted*	*How often this kind of communication should occur*
Example: Ensure staff can interpret pipeline performance information	Adoption team	Developers	Initial training class (Performance monitoring and analysis)	Face-to-face class with an on-demand video refresher	Quarterly Class offering with monthly reminders

5 Establishing the DSO Ecosystem

This section describes creating the environment, tools, and activities of DSO as an ecosystem. The four dimensions introduced earlier must be addressed in the adoption activities. Culture, automation and measures, processes and practices, and system and architecture are all addressed in this section. Figure 13 provides an overview of ecosystem establishment.

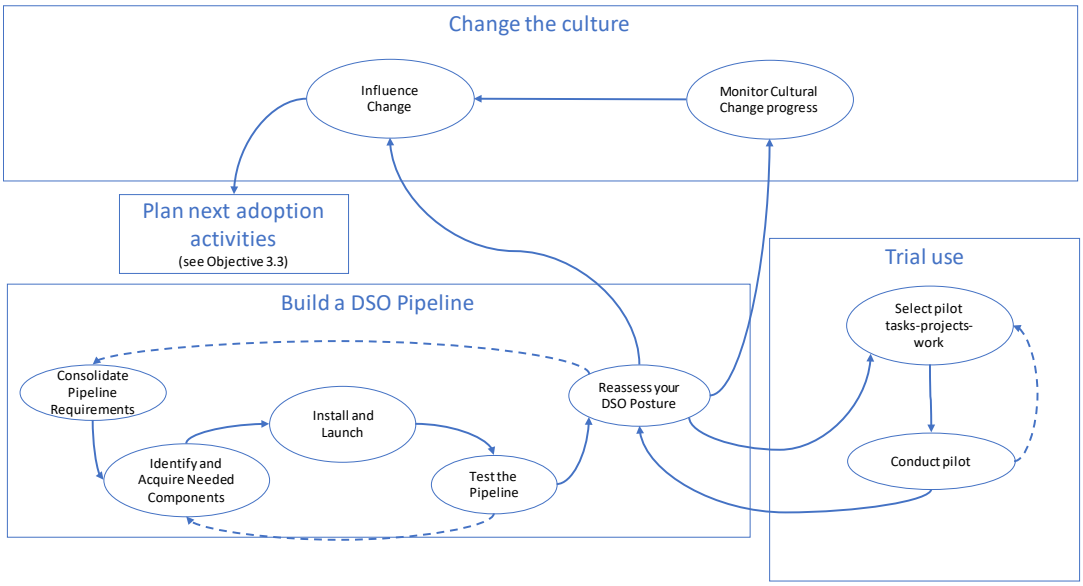


Figure 13: Overview of Establishing the Ecosystem

5.1 Objective: Change the Culture

For many DSO adopters, their first instinct (and probably highest interest) is establishing the tools, servers, and policies for the pipeline. While this is certainly an essential part of the effort, if the pipeline was ready on day 1, you probably couldn't effectively use it. A primary reason for identifying context and goals in the preparation phase is to understand the amount of cultural and process change that is required, and identify resources and a strategy that includes adapting the current culture into one that supports DSO principles and behaviors. That evolution must begin at the initiation of the adoption activities and continue as the pipeline comes online. Practitioners must use it properly, and customers' and other stakeholders' interactions with the new concepts must be routine.

Key cultural aspects of DSO adoption include, but are not limited to

- ensuring that adopters have the skills needed to succeed
- preparing upstream and downstream stakeholders for the DSO workflow (and the associated changes in process), and setting expectations on performing their responsibilities
- ensuring the developers, testers, security, and operations members of the pipeline team understand their roles and expectations

- ensuring the no-blame culture, fail early, honesty and transparency, and integration norms are understood and followed
- ensuring that Lean and Agile principles are exhibited by all

5.1.1 Activity: Monitor Cultural Change Progress

Context	*This activity is a continuous monitoring of the organizational culture to understand progress with respect to the measures adopted in response to the Adoption Culture Risk Assessment.*
Purpose	*This activity captures the current state of the supporting DSO culture in terms of risks, opportunities, barriers, and enablers.*
Overview	*The cultural profile is established and monitored at an appropriate cadence. Information comes from the culture- and risk-related measures collected periodically. This activity is often associated with technical milestones.*
Primary Actors	*This activity involves the manager, teams, S-CSs, and the culture change coach.*
Activity Input(s)	*Inputs to this activity include the current Adoption Culture Risk Assessment, adoption measures, and the current Culture Action Plan.*
Activity Outputs(s)	*Outputs of this activity include culture change information, the revised Adoption Culture Risk Assessment, the revised Culture Action Plan, and new culture awards.*
Other Resources	*Other resources include the RFA White Paper [Miller 2014], change management literature, and case studies of similar organizations.*
Tips, Tricks, and Wisdom	*Having leadership exhibit the appropriate behaviors (or not) is a significant measure of progress.*

Monitoring cultural change is particularly important during the initial establishment of the ecosystem. You have a good idea of the cultural issues from the readiness and fit analysis. The measures you identify can provide an initial measurement approach, but you will need to focus on more informal feedback early on. Retrospectives, interviews, monitoring status reports, and looking for friction points in meetings or informal gatherings can provide valuable information.

Maintaining a log of risks or opportunities identified as well as specific barriers and enablers discovered, perhaps as a blog or newsletter, helps the team understand the different ways their culture is changing and where change is still needed.

5.1.2 Activity: Influence Change

Context	*As cultural goals and risks are addressed and evaluated, actions are continuously taken to improve and maintain a DSO-supportive culture.*
Purpose	*This activity supports DSO culture acceptance and maintenance.*
Overview	*Change mechanisms are used to improve specific concerns or problems identified in monitoring. The measurement strategy and metrics may be adjusted to more accurately capture progress around the specific issues.*
Primary Actors	*This activity involves the manager, teams, and S-CSs.*
Activity Input(s)	*Inputs to this activity include current cultural needs, barriers, or enablers.*
Activity Outputs(s)	*Outputs from this activity include specific culture-related actions added to the increment plans and feedback from actions.*
Other Resources	*Other resources include the mechanisms identified in the adoption strategy and Table 4. Typical transmission mechanisms by Adoption Commitment Curve Stages.*
Tips, Tricks, and Wisdom	*Be innovative in responding to issues; don't overuse one or two mechanisms.* *Enlist leadership to exhibit and reinforce needed behaviors.*

Supporting the change must be an intentional, planned, and resourced activity. It is led by the coach or architect, but it must be supported by leadership, management, and representatives of the practitioners. Mechanisms exist to support change and maintain the change over time, but they are not sufficient in themselves. The mechanisms have to be used consistently, and this has to happen in a way that makes sense to the stakeholders. The old joke of "the floggings will continue until morale improves" is particularly applicable. The change needs to be seen as worth the effort expended. Using success stories, rewarding members of the team who "get it," and having them support change in other parts of the team are examples of maintaining excitement—or at least tolerance.

This activity will continue until the cultural norms of the team have become ingrained in the team behaviors to the extent that they are no longer noticed as different or new.

5.2 Objective: Build a DSO Pipeline

A pipeline is the most important technical implementation of DSO principles. Through automation and iterative processes, it enables the engineers to rapidly build, test, and deliver code. It requires careful design, attention to the requirements as captured in Epic 1, and a clear understanding of the development and delivery environment. A pipeline is composed of a collection of software applications connected via various communication scripts. Each software element serves to realize one component of the pipeline. Each component implements one or more requirements of the pipeline. Each requirement fulfills one or more DSO principle.

Table 6 describes common elements of a DSO pipeline. There are many commercial and open source software package options for each pipeline component. Implementation requires understanding the purpose and critical functions of each component regardless of the software used to realize it.

Table 6: Common Components of a DSO Pipeline

Interactive development environment (IDE)	The IDE is the environment that developers use to write source code. The code is also checked for syntax errors and built into an interpretable or executable file format. This is usually the first stop on a pipeline. Developers are tasked to write code and create a new code project on their IDE of choice to commence source code creation. In some cases, building the code may require including dependencies, such as libraries. This would be provided by the dependencies server which is discussed below.
Version control system repository	This repository stores versioned code for each developer. This component is essential to supporting continuous CI/CD. It facilitates developers as they create code in small segments that are easy to test, integrate, and troubleshoot. This component also provides a historical record of developed code and makes it possible to refer to older versions if the need arises.
Build server	The build server pulls together source code, dependencies, and environments to create a fully functioning environment. It plays a critical role in the automation of CI/CD and testing. With instructions from the user, the server pulls code repositories from the version-controlled server and builds them. The dependencies and provisioning servers are leveraged in the build process. The final output of this server is an image of an environment with executable code in the form of a deployable artifact. The artifact can be used for testing and delivery.
Provisioning server	This server implements the IaC and is used to create software-based platforms for various environments. A request is made for a particular infrastructure (e.g., network layout, desktops, servers), operating systems, and software. The server creates all this in one image. Typically, this server receives requests from the build server.
Dependencies server	Often when building an image, artifacts such as libraries, services, runtime environments, and other functionality are required for proper building and functionality. The dependencies server provides these artifacts—typically in response to a request from the build server. If the needed artifacts are external to the pipeline, this server retrieves them from external hosts.
Staging environment	This environment is internally hosted and is identical (or as similar as possible) to the intended production environment. The staging environment is part of testing and is meant to (1) emulate the intended environment where the code will be transferred to or (2) to discover and fix errors before transfer to production occurs. Often, this environment is a deployable artifact.
Production environment	The production environment is the intended environment for a software project. It's essential to include this environment as part of testing to ensure that code will run on the intended environment as expected. Performing testing in a CI/CD manner in production incrementally builds a successfully running code base.
Document repository	A document repository is a central place to store all project-related documents and updates. This repository usually includes a GUI providing last-minute updates and news for a given software project. Searchable folders for document artifacts are also provided. All stakeholders should be given access to this repository. When controlled information is considered, access controls must be in place to assure security protocols.
Workflow management system	This system is used to forecast and track the progress of each project-related coding task. A ticket is created for each task in code writing, testing, approval, faults, fixes, delivery preparation, and final delivery. This serves a source of metrics for project progress. As an individual task is advanced, the associated ticket should be updated. Each task is initially given a set of estimated hours and completion date. As work progresses, the actual hours spent are logged and compared to the estimate. Probability of completing by the estimated calendar date is partially determined with the comparison of estimated vs. actuals.
Monitoring service	The monitoring service collects data points from several parts of the pipeline, such as ticket progress, scheduled tests and deliveries, failures in testing and delivery, errors in the pipeline itself (e.g., component not functioning properly or at all), and scripts that are not responding or reporting errors. This service is a health monitor of

	the pipeline and the software project. It can also be applied to the performance of post-delivered systems.
Performance metrics	Performance metrics work with the monitoring service and provides the user with various analytics on collected data. These metrics help identify needed fixes, optimizations, and enhancements to the pipeline or the active software project.

As described in Table 7, a well-designed and well-implemented pipeline has several desirable characteristics. All of the characteristics are not always applicable; your particular environment determines those that are and their priority.

Table 7: Characteristics of a Well-Designed Pipeline

Loose coupling	Each pipeline component should have little to no dependency on other components to properly function. Using scripts for communication between components will achieve this.
High cohesion	Every component should serve only one purpose; ideally, each implements one required DSO principle.
Portability	Ability to run on diverse platforms using either virtualization or containerization.
Load scalability	The pipeline should seamlessly handle multiple developers working on a single or multiple project simultaneously.
Functional scalability	The pipeline should function while being enhanced in some way.
Heterogeneous scalability	Execution should occur regardless of which software is used for any component of the pipeline.
Recoverability	A “save state” should occur iteratively and restore if a pipeline stops functioning.
Usability	A clearly defined process should ease usage of a pipeline and its various components.
Maintainability	Functionality should be exposed to facilitate the updates and patching of various components.0
System security	Assure the pipeline’s infrastructure is controlled and monitor to avoid unauthorized data exposure or outside access.
Software security	Code should be developed with secure coding techniques and checked for vulnerabilities and unintended misuse.
Environment parity	Development, staging, and production environments should be the same. Infrastructure as Code (IaC), virtualization, and containerization assist in achieving this.

5.2.1 Activity: Consolidate Pipeline Requirements

Context	*This activity uses the information gathered in adoption preparation to capture the software and hardware requirements for the pipeline. Pipeline construction can be iterative or incremental. The requirements may evolve, but there are specific questions that need to be answered before construction begins.*
Purpose	*This activity captures the initial requirements for the pipeline based on information gathered in Epic 1 and a set of questions provided.*
Overview	*This activity draws on the information developed in preparation activities and establishes the requirements for a DSO pipeline that meets the context, readiness profile, and strategy of the organization.*
Primary Actors	*This activity involves the manager, team leads, and the pipeline (PL) architect.*
Activity Input(s)	*Inputs to this activity include the goals statement, DSO Adoption Strategy and Plan, DSO Posture Assessment Report, Technical Inventory, and Security Profile.*
Activity Outputs(s)	*Outputs from this activity include the pipeline requirements.*
Other Resources	*Other resources include the Pipeline Requirements Questionnaire.*
Tips, Tricks, and Wisdom	*If selecting open source components, verify they provided in-help guides connectivity scripts to other components. Many components are commonly used together in pipelines; leverage this to reduce scripting and configuration needs.*

Establishing requirements for the pipeline is essential to building a pipeline that is suited to your environment. The questions provided in the next section help you complete this task. Much of the information used to answer the questions will have been captured in the preparation activities. Some questions may require additional consideration or collaboration with stakeholders.

5.2.1.1 Pipeline Requirements Questionnaire

The answers to these questions help you create the requirements that the pipeline must satisfy.

1. What is the pipeline's purpose? A pipeline can be used for development of software, machine learning algorithms, and AI models along with data collection and curation, and program and project management. A pipeline can also be used to produce other pipelines. These are called *provisioning pipelines*. The answer to this question will help determine if the inclusion of specialized components is required. The Goals Statement should drive this information.
2. Are there security requirements for the pipeline? This question focuses on the security of the actual pipeline. This security perspective is primarily derived from the data and code being accessed in the pipeline's various components. Consideration must be given to projects with artifacts associated with an impact level or a security classification. The Security Profile should drive this information.
3. Where will the pipeline live? A pipeline needs to run from some place. Determining that location may not be trivial—and once decided will likely add needed compliance requirements. Primarily, a pipeline lives on actual hardware or in a virtual environment. The latter provides more accessibility, portability, and environment parity; the former can facilitate customized physical access controls. Security concerns will also define the pipeline's location. In answering this question, consider the security requirements (mentioned in the

previous paragraph) and as access controls. The required access can lead to the pipeline living in a publicly accessible system; in the other extreme, it can live in a highly classified system or physical location, or both. The Technical and Security Profiles support this information.

4. Will the pipeline carry out automated SoS integration tests? Recall that SoS relies on data flows between systems; testing for correct sending and receiving is critical. To ensure broad coverage in testing, you should understand all ingress and egress data flows for a given system being built by a pipeline.
5. What is the expected input to the pipeline? Since a pipeline can serve many purposes, its input can vary. In cases dealing with software development, the focus of this guide, the input is a newly defined project. The requirements and schedule are taken into the pipeline and converted, mostly manually, into development tasks, each of which is assigned one or more engineers to complete in an IDE.
6. Is there required access to specific services or resources? This requirement could impact the pipeline's location and, more than likely, its communication requirements. If a resource or service is internal, communicating with it may be relatively simple. If it is external, protocol requirements and authentication details will likely be needed. Secure connection requirements affect location.
7. What is the minimal viable product (MVP) that the pipeline must produce? The pipeline's minimum capability must be considered; it must have the capacity to produce at least one well-defined artifact. This artifact serves as a baseline, or starting point, from which the pipeline builds on in an iterative manner, primarily using CI/CD.
8. Are there budget considerations? Licensing and usage costs can partially determine the location and components of a pipeline. There are many open source (free) solutions often favored in pipeline construction. Open source software often includes various defined security measures, often referred to as being "hardened," that are usable in highly regulated environments. A bigger financial concern is paying for the use of resources, such as platforms and bandwidth. When first building a pipeline, the owners must do a long-term cost analysis to compare internal hosting or hosting using a third party. Usage must be determined to forecast associated costs. Third-party hosting, such as cloud environments, is often chosen for many reasons, including perceived cost savings. Carefully forecasting costs for expected usage may prove to be much higher than perceived savings.

5.2.1.2 Documenting the Requirements

The answers to the above questions provide a loosely coupled set of concerns, needs, and constraints. Capturing them in a more integrated form anchors the design and acts as the reference for validation and verification of the pipeline. The form the requirements take is less important; what is critical is ensuring that stakeholders and the pipeline architect understand the requirements.

5.2.2 Activity: Identify and Acquire Needed Components

Context	*Identifying and acquiring the infrastructure and components are a critical part of achieving DSO benefits. It should be approached collaboratively and consider the stakeholders' expectations and the available technical and security environment. The components may be selected and acquired in a single activity, or they can be incrementally acquired as resources become available. The DSO Adoption Strategy and Plan should reflect the approach.*
Purpose	*This activity provides the building blocks for the DSO pipeline.*
Overview	*This task translates the requirements into a set of ordered software and hardware components that will make up the pipeline infrastructure.*
Primary Actors	*This activity involves the manager, team leads, the pipeline architect, Procurement, and IT.*
Activity Input(s)	*Inputs to this activity include the Technical Profile, Goals Statement, and Pipeline Requirements.*
Activity Outputs(s)	*Outputs from this activity include pipeline components.*
Provided Work Aids	*Work aids provided include Pipeline Component Considerations and the Pipeline Design Template.*
Other Resources	
Tips, Tricks, and Wisdom	*Making pipeline decisions should be aligned with the culture assessment and strategy and never be the first activity undertaken. Without culture change, the pipeline will have only a minimal effect on overall SDLC performance.*

Figure 14 illustrates how the pipeline components listed in Table 6 interact. The two critical elements that must be decided are (1) the type of software components, and (2) the hosting of those software components.

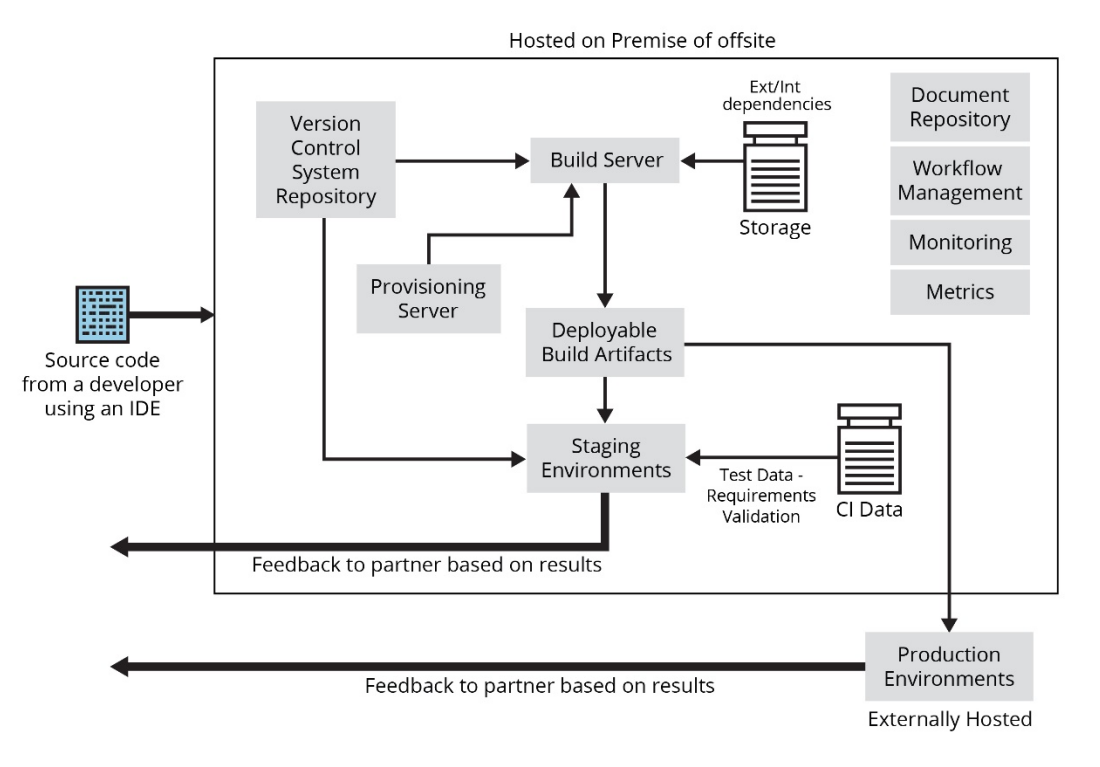


Figure 14: Connectivity Layout of DSO Pipeline Components

Each of the components can be implemented in one or more ways: purchased software, open source software, and software as a service (SaaS). Consider the following factors in determining your implementation:

1. Security. In a highly regulated environment, security is often top priority. An often-discovered source of security violations is software containing exploitable vulnerabilities. These vulnerabilities are often kept private and can be exploited for malicious purposes. Both purchased and open source software can contain vulnerabilities; if you are compromised via one of them, there could potentially be a "no fault" agreement on the part of the software provider based on the terms of usage. Ideally, vendors should promptly release a security patch once a vulnerability is disclosed. It is here where your decision matters.
   - Purchased software may be constantly tested for vulnerabilities, and the vendor should alert you of a patch that must be applied manually. In most cases, you can configure the software to have patches applied automatically.
   - Open source software may have varying degrees of ongoing vulnerability discovery with patch deployment. Some entities provide third-party support for open source software, but this increases cost.
   - SaaS for either open source or purchased software does not eliminate the vulnerability discovery and patching issues, but it may automate patch application due its own internal exposure to compromise. Furthermore, some SaaS providers are accountable when a client is compromised since they provided the vulnerable software.

Software security requirements are plentiful in an HRE. Leveraging SaaS implies that the service provider has already covered all security requirements for the offered software. This will save you time and money, and it shifts accountability to the provider for any security violation (assuming you validated their compliance before using their services).

1. **Cost**. Most open source software is free. Some have restrictions, so reading the license details is important. The license will state what a consumer can and cannot do with the software. A potential downside to open source is a lack of support from the software provider. Some community-driven support exists for popular open source software but the community often lacks sufficient understanding of the software's inner workings. This knowledge gap can result in sub-optimal response time and quality. The option to contract for third-party support for open source tools is usually available, but may be expensive. Buying commercial software or SaaS is often the best choice from an overall cost perspective. However, this approach should include long term support from the vendors gratis or at a reasonable cost.
2. **Support**. For many reasons, some discussed here, having support for your software is highly desirable. The support should include updates; security patches; online, phone, and around-the-clock (or a schedule suitable to your needs), in-person response personnel and support services. This is the basic support; there may be more options (such as training) that may incur additional expense. Closely study the support of ongoing products predict how many years of service you can expect. In some cases, your use of the software may outlive the provider's support. In these cases, upgrading to newer versions should be in your support agreement.

Hosting can be accomplished in-house or remotely (e.g., in a cloud environment). The following points overlap the software considerations, but they represent some of the issues to consider when deciding where to host a pipeline's component software:

1. Security. As previously stated, in a highly regulated environment, security—specifically, the proper handling and storage of controlled information—is often a top priority. To host this type of environment in-house requires meeting all the needed certifications required by law to be granted permission to access controlled information. If the in-house location only requires adding hardware and software to an existing infrastructure with the needed permissions, the time and cost to add a new host or system is greatly reduced. On the other hand, if no infrastructure with permissions currently exists, hosting remotely is a better option. There are several well-established remote hosting services that offer all the needed security certifications.
2. Cost. The main hosting-related investments fall into short- and long-term costs. If you are hosting on an existing in-house environment, the initial costs of setup are minimal. The sustainment costs may rise depending on usage and the need to hire new maintenance personnel. If the in-house environment is sub-optimal (i.e., it does not exist or it lacks certain requirements), the initial costs may be higher, but the sustainment could cost the same as an existing in-house environment. When considering remote hosting, some of the biggest costs can be related to bandwidth, storage, and usage (BSU). Forecasting different scenarios that combine these three factors is a good approach to assessing potential long-term costs. Of course, with remote hosting, there should be little or no up-front expense; the monthly costs are almost always based on BSU.
3. Access. A cornerstone of DevOps is full stakeholder access. When choosing a hosting solution, be sure to identify the stakeholders and determine if they require access to the data in a particular component. Ensure that a remote solution can provide for their data-access needs. If not, in-house may be the right solution.
4. Support. When hosting in-house, you may need staff to provide support that sustains the system in working order. An option is to use third-party contractors. On the other hand, remote hosting enterprises oversee and sustain all the infrastructure in use by their client portfolio. This lessens your burden and may result in needing only minimal staff to work with the remote hosting service to collectively resolve some of the issues.

The question of having all components of a pipeline live in the same system or not can be indirectly answered with the above considerations. In general, all pipeline components should live together in the same system. If a remote connection is needed, there should be strong justification for it. Furthermore, the security needs of one component should be extended to all components. This could help avoid running some components in less-secure systems and inadvertently passing controlled information to those areas.

To further aid in the component-selection process, consider the following:

1. Prior team experience. When choosing specific software for a pipeline component, it is important to determine if your personnel has experience using it. If not, then training may be required (and counted as an associated cost). There will be a learning curve for personnel who have no experience with the software. The effect of this curve on project scheduling

must be considered. As experience grows with usage, learning curves will diminish along with training needs. The critical risk is the added cost and potential delays upfront.

2. Component interoperability. When considering your component software options, realize the need to assure communication requirements with other pipeline components as seen, for example, in Figure 14. Remember that a pipeline is a sequence of components that interact in some way, and this interaction may be standardized or custom. The ability for a software component to communicate with other components may be by default or require third-party software to facilitate. The latter will introduce other technologies and require time to develop and test. This, of course, consumes time and funds. When possible, choose software components that include scripts or other methods to communicate with the other components.

5.2.3 Activity: Install and Launch the Pipeline

Context	*Technical implementation of the pipeline involves integrating the pipeline software and hardware, understanding the connectivity and creating links, and capturing the process as defined by the selected components.*
Purpose	*This activity creates the pipeline infrastructure and process.*
Overview	*This activity integrates the pipeline component in an iterative manner, identifying and documenting the roles and responsibilities of the pipeline components.*
Primary Actors	*This activity involves the pipeline (PL) architect.*
Activity Input(s)	*Inputs to this activity include pipeline requirements and pipeline components.*
Activity Outputs(s)	*Outputs from this activity include an operational pipeline with a validated use process.*
Other Resources	*Other resources include online resources associated with the components to support development of component communication scripts as needed.*
Tips, Tricks, and Wisdom	*In most installations, the default setup for any component will suffice minus scripts to connect with other components.*

The various components of a pipeline can be installed using simple scripts that pull software from a repository and launch its installation onto an existing environment. This is a straightforward process; installation scripts exist for several purchased and open source solutions. In the case of SaaS for multiple components, the vendor should provide the scripts and may only require configuration. Once the individual components are installed, communication scripts need to be written that allow multiple components to interact as needed.

Earlier in this report, we noted the need to identify the various roles required for a project. Each role has to be given instructions about how to use the pipeline to fulfill the role's responsibilities. Documenting the instructions is based on role-based pipeline interactions. Given a pipeline and set of roles, instructions can be developed about how each role will interact with the pipeline. A well-constructed pipeline will minimize manual tasks for the user.

5.2.3.1 Installation Process

1. Install a selected component in the pipeline. Remember that the technical implementation of a pipeline is a connected sequence of components that interact with each other. In most cases, the IDE or version-control repository are logical entry points for developers. Either of these components can be installed first.

2. Review the roles; identify the roles that interact with the installed component and their responsibilities.
3. Capture the necessary activities for accomplishing each responsibility for each role.

   For each role:

   - Do the following for each responsibility of that role:
     - Identify and carry out the activities required using the pipeline component.
     - Assure the desired results are achieved.
     - Document the steps taken, creating the instructions for executing the responsibility of this role using the current pipeline.

   This will be a real-time discovery of the steps needed to complete the actions for each responsibility for each role. Documenting the product could be captured on paper, maintained in a wiki, or implemented in a software-guided script.

4. Repeat role activity discovery and documentation (3) for each component.

The final product is a complete, step-by-step description of the activities required to satisfy the responsibilities of each role for each component.

The outcome of this activity is an operating pipeline and a complete process for using the pipeline by all identified roles, described as individual instruction sets. These instructions are the basis for testing the whole pipeline (covered in Section 5.2.4)

5.2.4 Activity: Test the Pipeline

Context	*Before trial use, the pipeline should be tested end to end.*
Purpose	*This activity validates the overall functionality of the installed pipeline.*
Overview	*This activity tests the overall functions of the pipeline as an integrated tool. Concurrently, it captures the user role responsibilities to create a documented process.*
Primary Actors	*This activity involves software engineers, operations engineers, pipeline (PL) architect*
Activity Input(s)	*Inputs to this activity include the Installed pipeline and the test project.*
Activity Outputs(s)	*Outputs from this activity include test results.*
Other Resources	*Other resources include information you can obtain by following each component's official online documentation for recommended testing.*
Tips, Tricks, and Wisdom	*Use a simple project that includes aspects that use every component of the pipeline, including testing (manual and automatic).* *Your very first run will likely have connectivity problems. This is normal. Resolve and continue testing. Repeat testing until you complete one successful run of the entire pipeline.*

Once a pipeline has been constructed, testing is the next critical step. One new project should be chosen to test the pipeline. The processes for using the pipeline defined in Section 5.2.3 should be distributed to all the test project stakeholders. As the project starts, execute the component and associated processes individually. After completion of each, determine if the component responded as expected; if it didn't respond as expected, determine why, and fix the issue before continuing

pipeline usage. Once a few cycles of the pipeline have occurred, reflect and ask all the stakeholders the following basic questions:

1. Did anything unexpected happen?
2. Was the pipeline able to handle unexpected occurrences?
3. Was a modification to a process needed to pass through a component?
4. Did the pipeline function expeditiously or slowly?
5. Were additional optional software packages needed for a component to perform a seemingly routine task?
6. Did the entire pipeline or any component stop working or become unresponsive during use?

Answer and address these questions on the first project used with the pipeline. Do not let sub-optimal answers go unresolved as other projects commence with the pipeline.

The pipeline should be validated to ensure that it responds and performs in normal and unusual conditions. Craft and execute tests for the following conditions:

1. higher than expected loads across the whole pipeline
2. higher than expected loads on an individual component
3. ability for a component to exit gracefully under unexpected conditions
4. effective handling of user saturation
5. functional operation under high-stress by combining points 1-4

Once testing results are satisfactory, the pipeline can be scaled for use by other new projects. A good rule is to not require ongoing projects to switch to the pipeline. The time and resources needed to accomplish this action are likely overwhelming and unacceptable under that project's forecasted schedule completion.

5.2.5 Activity: Reassess Your DSO Posture

Context	*This activity happens at each step of the adoption process.*
Purpose	*This activity quantifies the impact of the pipeline on providing your desired SDLC.*
Overview	*This activity determines if the pipeline addresses its associated DSO technical goals and provides insight into progress.*
Primary Actors	*This activity involves software engineers, QA, requirements engineers, and the program manager.*
Activity Input(s)	*Inputs to this activity include a detailed pipeline architecture with performance metrics and testing results, the questionnaire below, and results from the initial DSO Posture Assessment.*
Activity Outputs(s)	*Outputs from this activity include answers to the questionnaire, a determination of progress, and potential remediation action tasks.*
Other Resources	
Tips, Tricks, and Wisdom	*Repeat pipeline testing if remediation action items are put forth. Continue with this testing until no action items are given. Be aware that the pipeline may introduce new issues not previously present; remediate and repeat test in this case.*

An initial DSO posture assessment should be carried out before actually considering implementing DSO to your environment. At this point, you have (1) installed and tested the DSO pipeline and (2) collected data via logs and reporting; you can now measure progress and impact by revisiting the following posture assessment survey questions relevant to the pipeline:

1. *Source code commits per day*. Is the group satisfied with the commits the pipeline can perform per day?
2. *Consistent development environment.* Are the various environments being used by developers and other HRE personnel able to be kept consistent and up to date with methods such as tools, updates, and features with this pipeline?
3. *Tooling usage.* Does the pipeline make a diverse set of tools available to developers? Are there multiple tool sets for the same purpose? Can the current tool set be reduced to exclude those not favored by developers?
4. *Production delivery*. How often can code be pushed to production for end-user feedback? Can delivery be performed by one person and a group? Is delivery a consistent, repeatable process, or is it a unique effort each time?

These questions and the DSO reassessment quantify the impact of the pipeline on providing your desired SDLC. A full project has not been carried out at this point, so a full DSO reassessment is premature. The pipeline will provide technical advances and some technical process changes; it is not sufficient to establish the culture and organizational process changes critical to DSO. Compare your current answers to the original answers, and assess if improvements have occurred. If the pipeline has not advanced your SDLC in a technically specific way, you can modify the pipeline components and configurations and then retest and reassess.

5.3 Objective: Conduct Trial Use

The trial use is the turning point in determining the true impact that DevSecOps will have on your SDLC. The trial use will provide a first-hand look at DSO in action and its effects on your culture, personnel, project performance, and finances. You should not expect a smooth experience since this is a first-time use. Rather, this trial use is a good opportunity to discover "rough spots" in the DevOps practice that need to be resolved. A rough spot is anything that causes a pause or regression in advancing to your desired SDLC. Be aware of new negative issues that may arise as a result of implementing DevOps; capture and resolve these. At the end of this trial, you should have a clear understanding of how DevOps will impact your technical, business, and financial strategies. You should also be satisfied with DSO to the point where you are ready to implement, or recommend its implementation, across your organization.

5.3.1 Activity: Select Pilot Tasks/Projects/Work

Context	*Selecting pilots is the first step in trial use. It attempts to maximize the information gained from Trial Use.*
Purpose	*This activity creates a list of tasks or projects to act as pilots.*
Overview	*Trial use pilot(s) selections are made collaboratively, using established criteria for considering the projects available.*
Primary Actors	*This activity involves the program manager and the pipeline (PL) architect.*
Activity Input(s)	*Inputs to this activity include DSO Goals, the latest version of the DSO Posture Assessment, the installed and tested pipeline, and validated process guidance.*
Activity Outputs(s)	*Outputs from this activity include a selected pilot project.*
Other Resources	*Other resources include Sections 7.1.4 and 7.1.5.*
Tips, Tricks, and Wisdom	*Senior leadership should buy in on selected project.* *Don't choose a multi-year project for the trial run. It may be too long before remediation can occur.*

The first step in carrying out the trial use is identifying a project to execute using DSO. The project should be a new one that is set to launch. Requirements should have been gathered, and a statement of work (SOW) should be in place. The SOW should clearly state the project, its goals, and deliverables. The project should not require the use of technology that is new to your organization.

The following should help guide you in selecting the ideal project for trial use:

1. Choose a completely new effort that has not yet commenced.
2. Check that requirements have been obtained from the client.
3. Be sure your organization has a clear understanding of the project and its deliverables.
4. Select a project that uses technology with which your organization has experience.
5. Choose a project that requires personnel covering a diverse set of roles.
6. Find a project that will need a baseline of sufficient personnel to produce a minimally viable product.
7. Choose an effort that will have people touch every component of the DSO pipeline.
8. Engage a project that has the flexibility to schedule extra time to resolve DSO-related issues.

9. Pick a project where the customer is in favor of using DSO.

The goal of the above guidance is to select a project that the team feels comfortable with, where the customer is agreeable to DSO, and which has a delivery date that allows time to fix potential DSO issues. Having diverse role types allows testing the various sets of created instructions that belong to the DSO process. It's critical that the organization is able to visualize the project, how it can be executed, the phase of work and their outcomes, and the final delivery. This visualization is important because it should facilitate the use of a new SDLC approach. If the team found a project difficult or challenging, using a new approach would be a sub-optimal choice.

The trial project chosen should be for an SoS. Recall the two types of SoS defined in this document. The additional tasks in testing SoS are based on the data flows of source and derived values between systems. These tests will be automated just like all the other integration tests in the pipeline. In designing the required tests consider the following:

1. Do you understand all the ingress and egress data flows? It is best to have an architectural diagram of the various systems with their relevant usage and data flows for source and derived values.
2. Is the value created in the current system? It is important to confirm if the value, either source or derived, is truly created in the current system and not passed in from another system. If the value is passed from another system, then the system that created this value is responsible for its testing.
3. Do the components of other systems exist for testing? Inherent to source and derived value integration testing is the existence of other systems providing a pathway for value traversal. Following the DSO iterative approach of building, testing, and delivering small code segments, it is inevitable that SoS integration testing will start early in the SDLC. Systems will need to provide function and procedure calls (possibly empty), to allow a value to traverse onto the required functionality. The functionality that uses a value as input must exist to test—even if the rest of the system is nothing more than function calls creating the value's traversal pathway.

5.3.2 Activity: Conduct Pilot Tasks/Projects/Work

Context	*The pilots will identify technical, process and culture conflicts, mismatches, errors, and improvements.*
Purpose	*This activity supports and monitors the pilots.*
Overview	*The pilots are conducted on the operational pipeline using the validated use process.*
Primary Actors	*This activity involves the program manager, the pipeline (PL) architect, customers, end users, software engineers, requirements engineers, the quality assurance engineer, the operations engineer, and the security engineer.*
Relevant/Key Events	*Events include the commencement of project, schedules milestones, final delivery into production, and handover to the customer.*
Activity Input(s)	*Inputs of this activity include selected pilots, requirements, personnel, the process of using the pipeline, the schedule, milestones.*
Activity Outputs(s)	*Outputs of this activity include the delivered product to the customer.*
Other Resources	
Tips, Tricks, and Wisdom	*Before the project starts, hold meetings and talks to get personnel thinking in the DSO style, so they mentally plan and make decisions conducive to DSO.*

Once a project has been chosen as a pilot, it should commence using the DSO approach combined with the organization's traditional approaches. Traditional approaches include creating an overall schedule, finalizing personnel selections for the project, and allocating time and budgets. Once these traditional approaches are completed, some of them should be transitioned and maintained on various components of the DSO pipeline. Schedules are broken down into milestones, then work tasks, then work subtasks. Each subtask can be a ticket pending assignment to personnel and estimate of hours to complete. Using the ticketing system to show progress that builds toward major milestone deadlines is a key part of DSO since it allows full stakeholder viewing, transparent tracking, and dependencies of other tickets needed for completing some bigger task or milestone.

Subtasks can be broken down into smaller tasks. In DSO, any individual task should be small enough that it can be done in a short period of time—maybe no more than two weeks—by one group involving no more than two or three people for the core work. Of course, people in other support groups (such as environments or configuration) may need to assist. This is true for many tasks.

The project should progress to completion if you follow the process and use tracking tools (e.g., the ticketing system) effectively. As work is completed, notes should be kept describing any negative issues or confusion that arose, and how previous bottlenecks and inefficiencies were eliminated. Performance metrics gathered and analyzed in the monitoring server will also quantify relevant metrics, including time to complete a task, number of deliveries per day/week, attempted/successful/failed builds, and delays in task completion. Publish (at least internally) a trial-use report summarizing the pros, cons, and impact of DSO on the pilot project.

As part of the process, you may want to use previously employed update events, such as daily and weekly meetings. This is perfectly fine and is a key component of DSO. During these meetings, use performance metrics and tickets to track and inquire about various work tasks.

You may not be willing to perform a trial use with a customer-funded project. If so, use an internally funded project; that way, any delays or negative results will not impact your customers. If an internally funded project is not an option, you may have to fabricate a test project.

Regardless of the project type, once chosen, follow the guidance in this section for the trial use. When running the trial use, be patient, expect problems, and be prepared to resolve them. When the first issue arises, do not abandon the adoption. Once the project completes and is delivered, conduct a full reassessment of your DevOps posture.

5.3.3 Activity: Reassess Your DevOps Posture

Context	*This happens at every step of the adoption process.*
Purpose	*This activity is a full assessment of post-trial use.*
Overview	*This is a final check on the initial pipeline deployment.*
Primary Actors	*This activity involves all stakeholders.*
Activity Input(s)	*Inputs to this activity include the DevOps posture questionnaire, trial-use issues, met and failed schedule dates, and milestones.*
Activity Outputs(s)	*Outputs from this activity include questionnaire answers and remediation action items.*
Other Resources	*Other resources include questionnaire answers for all previous posture assessments related to this project.*
Tips, Tricks, and Wisdom	*This was the first run; it won't be perfect. Learn from it, make changes to improve posture, and move on to the next project.*

Completion of the trial use implies having applied DSO to your SDLC's culture, process, and pipeline. Project experiences and results are documented and quantified. With this analysis, a full post-DSO implementation Posture Assessment is appropriate. Repeat the full survey, and compare your current answers to the previous answers from the pre-DSO implementation assessment. The current answers should be at (or advanced toward) your ideal SDLC. Noticeable improvements should be evident in every step of the SDLC. You should also list new issues that arose as a result of implementing DSO. These issues should be analyzed in detail. Hopefully, they were resolved during the trial use; if not, a proposal should be made on how to resolve them in future projects. The potential solution could be a change in process with the current pipeline, a modification to the pipeline, or both.

A good exercise is to conduct several meetings and surveys with all personnel involved in the trial use. Meetings should be conducted in large and small groups, and with each individuals (one-on-one). In each meeting, ask participants how they felt about using DSO; record all input. Repeat the meetings as needed, and ask as many questions as desired to conduct a project-wide census on DSO. This is an important step since you are moving toward a new organization-wide approach that will affect all future projects. Therefore, you must feel confident that the feedback reflects acceptance toward positive change. Once you're satisfied with the current DSO posture and the overall trial use, you should formalize DSO into your organization. In preparation for this step, ensure that your organization's senior leadership fully supports adding DSO as a standard practice to be adopted by all sectors and personnel.

6 Manage and Evolve the Ecosystem

Just as DSO is a continuous process, the refinement and improvement of the ecosystem is a continuous process. Lean and Agile practices include retrospective as an integrated means for feedback and improvement opportunities. The pipeline provides data that can be used to identify problems and illustrate the impact of process or tool changes. The DSO posture evaluations provide another means of tracking, maintaining, and improving the pipeline and cultural aspects.

Extending the DSO ecosystem can be done within a single enterprise or can extend to other developers in the system of systems who either see the results of the initial adoption or are directed by their own leadership. The data that is gathered and analyzed can be a powerful tool to aid DSO implementation and adoption.

It is important to understand that feedback must be honest, the measurements must be accurate, and the information must be provided in a manner that is transparent across the team. Maintaining those cultural norms are key to extending the DSO ecosystem and improving the pipeline and cultural aspects. Figure 13 illustrates the associated activities.

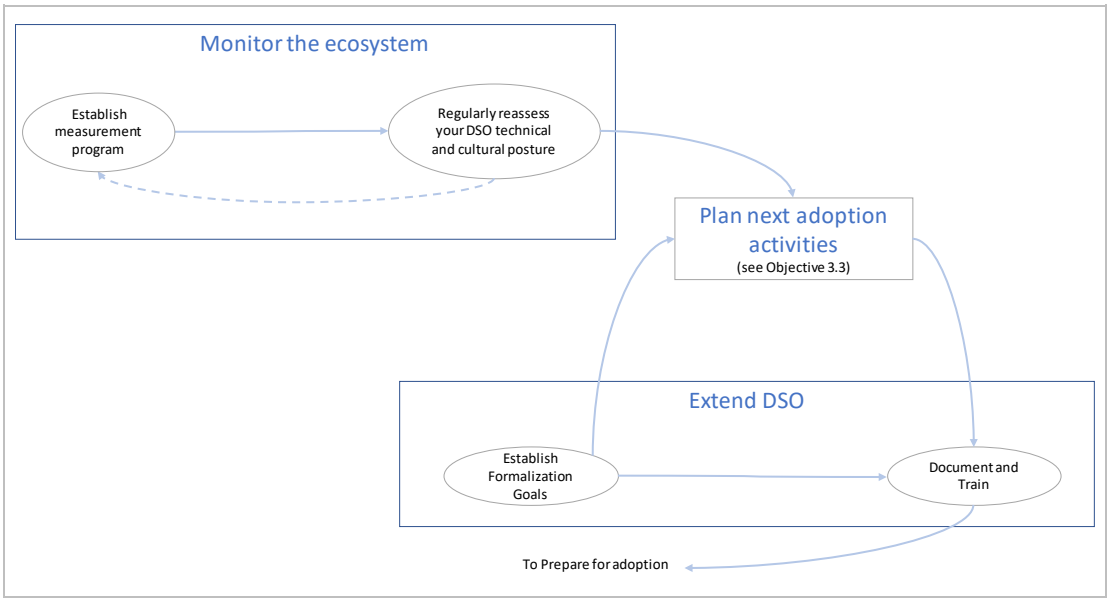


Figure 15: Overview of Manage and Evolve the Ecosystem

6.1 Objective: Monitor the Ecosystem

Once DSO has been institutionalized across the organization and is standard practice for all new projects, you must continue to monitor its use. It is important to realize that DSO-related issues will still arise and will need resolution.

Figure 16 illustrates the scope of this monitoring. Note that trial use and institutionalization will not capture all potential issues.

6.1.1 Activity: Establish a Measurement Program

Context	*This activity is a long-term performance metric tracking pipeline, process, and culture.*
Purpose	*This activity determines and creates long-term metrics for DSO effectiveness.*
Overview	*Senior leadership and program managers determine the best metrics for long-term monitoring of DSO effectiveness across the organization.*
Primary Actors	*This activity involves senior leadership, program management, and software engineers.*
Activity Input(s)	*Inputs to this process include formalization goals.*
Activity Outputs(s)	*Outputs of this activity include a list of desired data for long-term analysis of DSO.*
Other Resources	*Other resources include the webinar Three Secrets to Successful Agile Metrics.*
Tips, Tricks, and Wisdom	*Remember your analysis should focus on DSO culture and process. Don't get stuck only on the pipeline performance.*

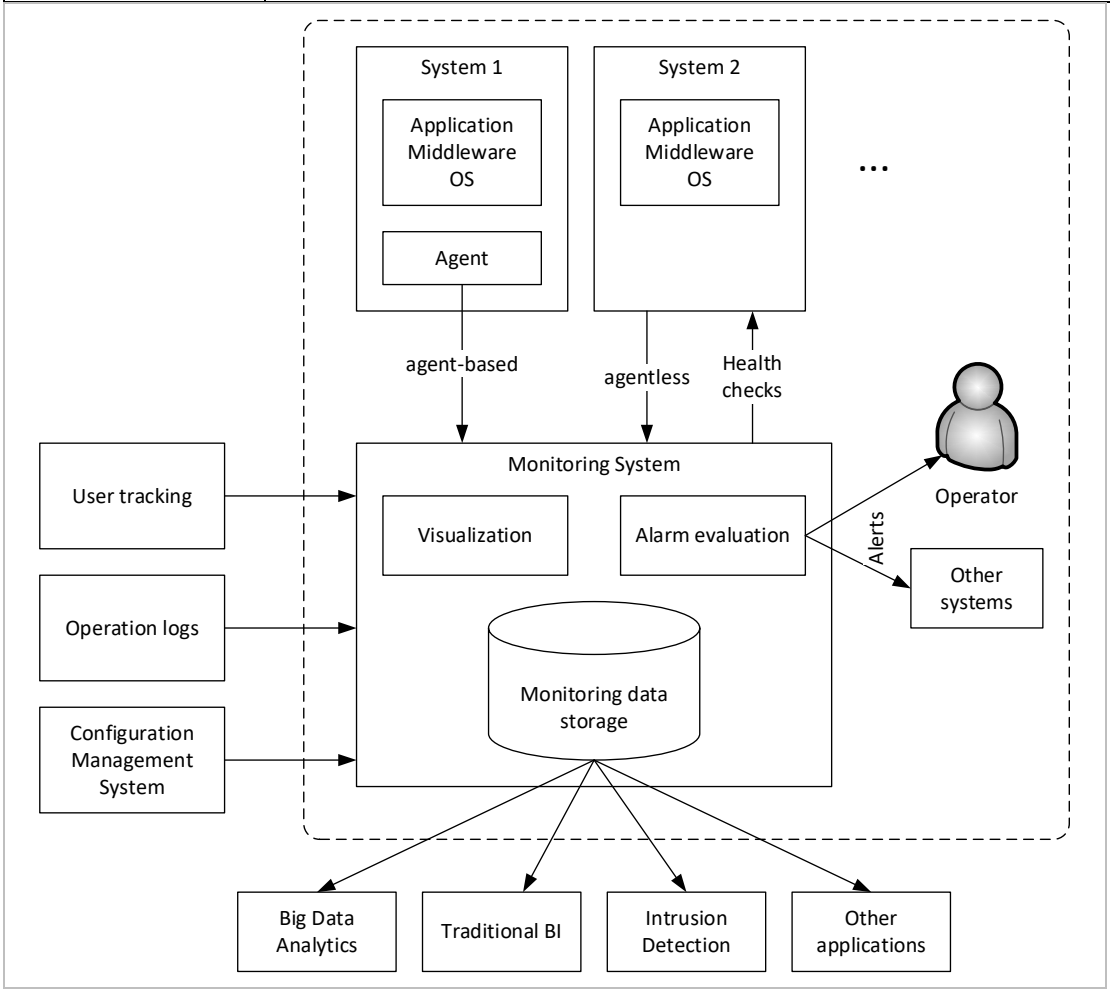


Figure 16: Monitoring System Architecture

An important aspect of confirming DSO institutionalization progress is tracking various metrics deemed representative of the status of effort status and its impact. Some of these metrics can be

derived from the performance metrics of the DSO pipeline. Other metrics can be sourced from (1) personnel using surveys and interviews and (2) the organization's personnel department and finance reports for a given project.

DSO pipeline performance metrics. As previously discussed, these metrics provide technical analysis on attempts, successes, failures, time to complete on various tasks (e.g., commits, unit tests, functional tests, builds, delivery into staging and production). As personnel become more comfortable and accustomed to DSO, these metrics should steadily improve. It is acceptable if some metrics plunge and spike because incidents will happen. If, over time, a metric stays the same or underperforms, it is likely a sign of a fundamental problem that should be investigated. It is up to you to set the threshold values that trigger an inquiry into a perceived problem.

Another metric to track is the number of tickets opened to report a problem. Tracking such numbers and the time to resolve is important since the data may indicate a deficiency in the pipeline or process. Alternatively, it may indicate a skill set is lacking; such an issue should be addressed but falls outside of DSO.

Interacting with personnel. Technical metrics may not capture the full status of DSO institutionalization. Those metrics lack in directly capturing the state of the culture shift. This aspect, on the other hand, can be implied with technical metrics that display slowdowns, delays, and failures. These metrics can signify frustration, lack of understanding, and—worst of all—lack of motivation. Regular discussions with small groups to discuss their experiences, highlights, and issues with DSO are critical to ensuring positive cultural shifts. These discussions can be prearranged or ad hoc; however, ad-hoc meetings are better since they can reduce potential influence of peers and management on answering questions. In the meeting, if you sense negativity, start interacting. In general, let everyone in the organization know that they can speak frankly without fear of repercussions. When needed, take all necessary steps to provide privacy and anonymity.

More information about change measurement is available in publications by Dorothy Leonard-Barton [Leonard-Barton 1988], Gerald Weinberg [Weinberg 1997], and R. Zmud [Zmud 1992].

Personnel and financial reports. When an organization's culture is happy and positive, employees stay with the organization. When it is not, tensions rise along with complaints to the human resources department. Another potential indicator of a sub-optimal culture is an increase in vacation and sick days. If any of these are occurring beyond normal thresholds, you should meet with the teams to which the unhappy individuals belong. Avoid meeting directly with the individuals at first, as this may cause them to think they are being singled out, which will not help the situation.

Overall, when addressing concerns, always meet with the group first. This facilitates addressing the issue without singling out an individual. Remember, the goal is to ensure that the cultural shift caused by implementing DSO is trending in a positive direction. The goal is not to single out struggling individuals or those who are not in concert with the shift. They can be helped through group training, presentations, practicums, and meetings.

6.1.2 Activity: Regularly Reassess Your DSO Technical and Cultural Posture

Context	*This activity is a routine assessment to gauge long-term impact and effectiveness of DSO.*
Purpose	*This activity assures DSO impact and effectiveness trends have not fallen to near unacceptable levels and fixes those that may be declining.*
Overview	*This activity is a progress assessment that looks at all aspects of the DSO adoption effort.*
Primary Actors	*This activity involves all stakeholders.*
Activity Input(s)	*Inputs to this activity include results of all previous DSO assessments and the assessment questionnaire in 2.1.5.*
Activity Outputs(s)	*Outputs to this activity include answers to the questionnaire, the current DSO posture, and remediation action items provided to incremental planning activities.*
Other Resources	
Tips, Tricks, and Wisdom	*Watching assessment trends over time will clearly indicate cultural acceptance and/or defiance.*

The key to continuous monitoring is threefold: reassessing your DSO posture, monitoring pipeline performance metrics, and regularly meeting with personnel. This approach provides a view of the DSO's culture, process, and pipeline. If you follow the potential causes of sub-optimal results and trends previously discussed, you will be able to continuously determine if problems are arising—or, in some cases, persisting—as a result of DSO.

The current DSO implementation is not meant to be static forever. Inevitably, the DSO will need modification in some fashion, probably to the pipeline or the process. Pipeline changes arise from technical shifts in client needs, standards, and community practice. Each of these may include the adoption of new technical approaches and retirements of older ones. Options for the various pipeline components will evolve over time, and better options will arise and be adopted.

A pipeline designed with heterogeneous scalability facilitates the swapping of individual components with minimal disruption. Changes in process can occur resulting from pipeline changes or new requirements from the organization, an oversight committee, or a regulatory policy. In these cases, processes must adhere to the prescribed requirements. Be aware that this may produce a sub-optimal DSO implementation. Consideration can be given to changes in other DSO aspects to balance out the sub-optimization but may require a new assessment and modification.

As discussed throughout this report, the hardest changes to assimilate are cultural. This type of change can come about from a number of events:

- sweeping and highly impactful shifts from senior leadership or policy
- attrition of trained, expert personnel
- significant influx of new personnel
- contract or contractor changes

These events must be analyzed closely as they may require changes in all aspects of your current DSO implementation. Determining and implementing those changes may require a new assessment and modifications according to the results. Remember that performance metrics are not

meant to always trend positive or in favor of your point of view. Metrics will rise, fall, and remain steady in predictable ways (or for no reason at all). Inquiries should be made when a change persists or impacts the organization beyond your comfort zone. As time passes, continuous monitoring will facilitate your organization's adherence to the "new normal" as reported by metrics, feedback, and assessment. With this exposure, you will be able to detect the truly unexpected and suboptimal, and quickly respond with inquiries and resolutions.

6.2 Objective: Extend DSO (Institutionalize)

Once your initial DSO implementation is operational, you may decide to embed DSO across your entire organization and make it part of core operating practices for software development. This is a serious but probably rewarding step to take; it will affect several sectors in your organization. Ideally, you observed these organization-wide impacts in trial use and have a good idea of the issues that may have arisen and some resolution strategies.

6.2.1 Activity: Establish Formalization Goals

Context	***These goals will institutionalize DSO as regular practice within your organization.***
Purpose	***This activity establishes the use of DSO on all projects, organization-wide.***
Primary Actors	***This activity involves senior leadership and program managers.***
Overview	***Senior leaders and program managers collaboratively identify goals that set the standard of practice in using DSO for all future projects.***
Activity Input(s)	***Inputs to this activity include trial usage final reports, DSO post-implementation assessment findings, and the DSO process and pipeline.***
Activity Outputs(s)	***Outputs from this activity include an organization-wide mandate detailing the mechanics of DSO usage on all projects.***
Other Resources	
Tips, Tricks, and Wisdom	***Several pilots will be needed for the whole organization to accept DSO as "the way we do things."***

Meet with every department in your organization to explain impacts to and expectations from them. Present DSO and its benefits to their group and the organization, providing details about the pros and cons specific to them. In the software development group, meet with individual teams to introduce and discuss the technical changes and how they are supported through the pipeline and other tools. At each meeting, detail the procedural changes that are being modified or replaced and the procedures that they must follow. Be sure to provide points of contact for questions and assistance.

Most importantly, develop and provide training about the organization's DSO ecosystem, and tailored it to the various sectors.

Update training as technology and processes change (such as adding a new pipeline component or a security requirement that affects the DSO process).

The main goal is for all sectors of the organization to be aware of how DSO affects their way of doing business. In some cases, there may be no or minimal impact; in other cases, a major shift

may occur. The trial use gave insight into this, so it is critical to meet with groups and explain what permanent changes they should expect.

The following guidance helps institutionalize DSO:

1. Meet multiple times with the groups to answer questions and receive feedback.
2. Allow people to ask questions and make comments at any time and any day, likely via the corporate website and by telephone.
3. With the help of senior leadership, ensure that DSO is included in all relevant organization-wide documents, procedures, yearly reports, marketing materials, and similar items.

It's critical to remember that institutionalization will not occur overnight. A cultural shift needs to occur within the organization. Only by repeatedly reminding, retraining, inquiring, and following up will a cultural shift have a strong chance of succeeding.

6.2.2 Activity: Document and Train Personnel

Context	*This activity includes continuous teaching of personnel in all areas of DSO.*
Purpose	*This activity keeps personnel up to date on all DSO-related issues.*
Overview	*Training, based on the latest process, is created, offered, and delivered to all team members as needed.*
Primary Actors	*This activity involves program managers and senior leadership.*
Activity Input(s)	*Inputs to this activity include new training materials and documented updates on DSO.*
Activity Outputs(s)	*Outputs from this activity include training videos, slides, books, meetings, and Q&A sessions.*
Other Resources	*Other resources include third-party trainers, various media with new training, and/or DSO changes.*
Tips, Tricks, and Wisdom	*Use training to keep personnel current on DSO technology and process refinements.* *Over training or too-early training can be detrimental.*

Key to DSO is the creation, usage, and—most importantly—updating of a readily accessible communication hub and repository. This hub/repository serves as the central location for the project's papers, notes, advisories, and announcements. At a higher level, the repository can also house the guidance, practice, and methods of SDLC along with video and other presentations regarding DSO, the pipeline, and related issues. Continuous documentation of all aspects of a project, including performance metrics and assessments, is critical. This body of knowledge will serve as a time capsule archive once the project is completed. The data collected can be used for studies performed across long periods of time or across multiple projects.

Regular, continuous, and ongoing DSO training is critical to adoption success and long-term sustainment. All training should be recorded and made available to all personnel for viewing at any time with no required approval or authorization, except for security reasons. Below is a list of topics for which training should be created:

- how the DSO pipeline works
- the steps in using the pipeline

- starting, continuing, and finishing projects
- how DevOps functions in the background

Training should be crafted for specific audiences by scoping the breadth and depth of knowledge. Business and finance operations will not require the same technical detail as software and quality analysis engineers, but both will likely benefit from some detail not required by mid-level management. Training should not be a "one-way street." Personnel should be given DSO-relevant exercises to carry out as homework that is graded. This is very important since it validates whether the trainees understand the DSO. Personnel should be taught the material from a theoretical and usability standpoint. It is sub-optimal to teach only how to use a pipeline without explaining DevSecOps from a theoretical, classroom-style perspective. Keep training fresh and up to date. As changes in any aspect of DSO occur, consider developing a new training module or updating existing modules to reflect those changes. During every training session, and especially at the conclusion, encourage feedback. If possible, hold a meeting with trainees after the training is completed to reflect and comment on its usability and potential additions and enhancements.

7 Concepts, Principles, and Tools

This section provides more information about concepts and tools supporting DSO adoption, development, and management. Because of the scope of DSO, adopters need to understand the basics of several key concepts and principles, especially those directly within the DSO domain. This section provides brief introductions to those basics for those unacquainted with them. More information can be found in Appendix B.

7.1 Technology Adoption and Culture Change

Adopting DSO means adopting a new technology and, as already discussed, changing your organization's culture. There is an extensive body of knowledge on culture change and technology adoption that can help you plan for and successfully accomplish DSO adoption. This section draws heavily from Miller and Turner [Miller 2006].

7.1.1 Difficulty of Change

Figure 17 illustrates the difference in difficulty associated with a set of factors that help to determine the scale and scope of an organizational change. It is adapted from Paul Adler's work [Adler 1990]. Note that culture requires a significantly higher level of learning.

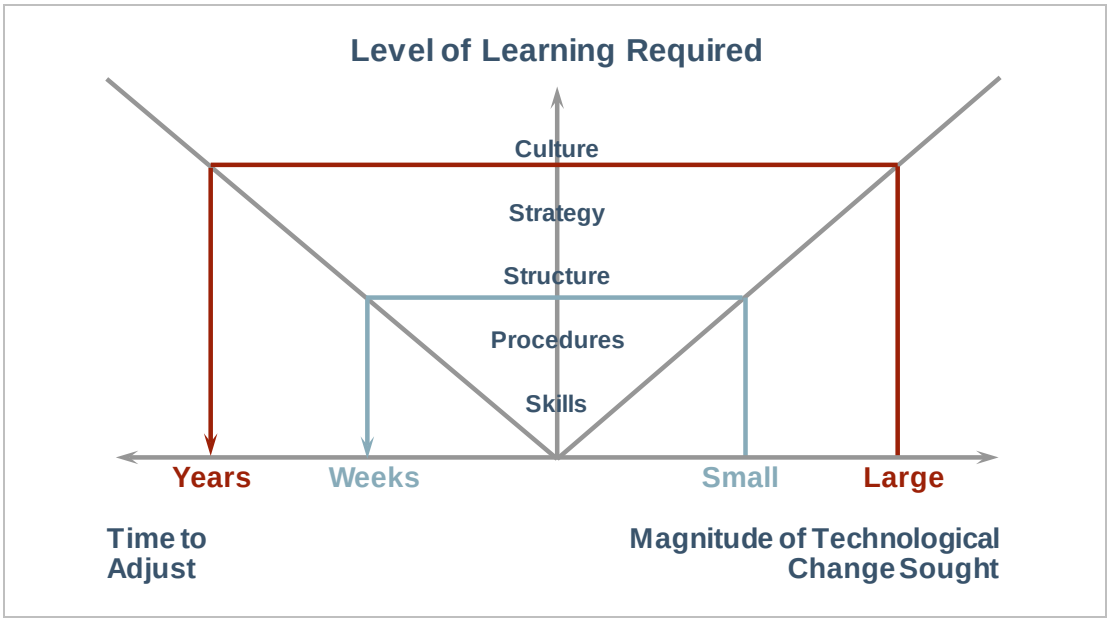


Figure 17: Difficulty of Change (Adapted from Adler [Adler 1990])

The following are definitions for various factors:

- **Skills:** The change with the least impact involves something where all that needs to change are the skills of the people adopting the new practices. The caveat here is an assumption that the new skill has some grounding in other skills the adopters are likely to have.

- **Procedures:** Procedures are next higher on Adler's scale. When procedures need to change, there is usually a chain of command that must be brought into the decision. Sometimes the organization's management is unaware of the procedural changes that adopting a new technology (e.g., an electronic health record system) will have. Changing procedures sometimes also involves changing where power resides within an organization and can lead to conflict.
- **Structure:** Beyond procedures, structure is the next higher item on Adler's scale. Structural changes almost always involve changes in power structures, which gets the attention of people who are not necessarily actual adopters of the new practices but are affected by the adopters (or are affected by the outcomes of new procedures). Any time power and its exercise are involved, passions will run high and resistance to changing the status quo is likely.
- **Strategy:** Strategy goes beyond structure to affecting the senior decision-makers in an organization. When business strategy changes, it often means that attention is paid to shifts in the markets, and there are implications for all the factors below it on Adler's scale.
- **Culture:** Culture is at the top of the change-difficulty scale from Adler. When culture is expected to change, it impacts people's values and their assumptions about what behavior is acceptable and not acceptable within the organization. Often these assumptions and values are not explicit; they are discovered primarily by violating one or more of the organization's norms.

7.1.2 A Change Model (Satir)

There are several ways of representing the cycle of responses human beings make to change. The SEI has found the Satir Change Model best fits both process improvement and technology adoption. It is useful for both because it is descriptive (i.e., it explains the symptoms often seen in organizations going through change) and it is somewhat prescriptive (i.e., it provides ideas for helping people navigate the cycle of change). The SEI bases its understanding of the Satir model as presented and explained in Weinberg's *Quality Software Management Volume 4: Anticipating Change* [Weinberg 1997].

Figure 18 presents a summary of the Satir Change Model. The individual or group starts at some level of performance, represented as the "old status quo." The introduction of a change intended to improve the individual or group's performance is treated by the group as a "foreign element." The group will have different reactions to the foreign element. Some of the possibilities include trying to

- ignore the foreign element
- find a way to accommodate the foreign element within their own current way of doing things
- explicitly reject the foreign element

The energy that goes into these reactions causes swings in the performance of the group that can be dramatic, depending on the character and size of the change being introduced. At some point, if the foreign element doesn't go away, most groups will find what Satir calls the "transforming idea" that will allow the group to integrate the change into their way of doing things and will allow them to move on.

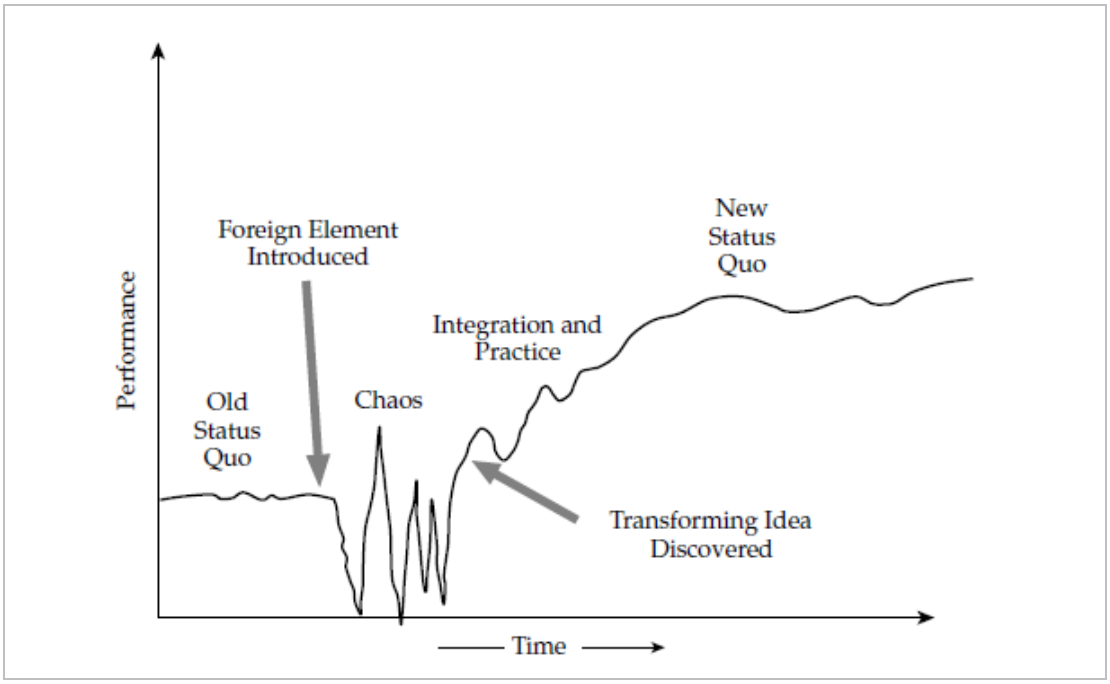


Figure 18: Graphical Summary of the Satir Change Model (Adapted from Weinberg)

When the group has found and accepted a transforming idea, it integrates the new behavior into its routines by practicing the new behavior. During this time, the group's performance starts to improve; however, this increased performance can occur only if there is opportunity to practice the new behavior.

After the new behavior is integrated into the group's behavior, it becomes the "new status quo," and whatever performance increases have been achieved are likely to continue.

7.1.3 Adoption Commitment Curve (Patterson-Conner)

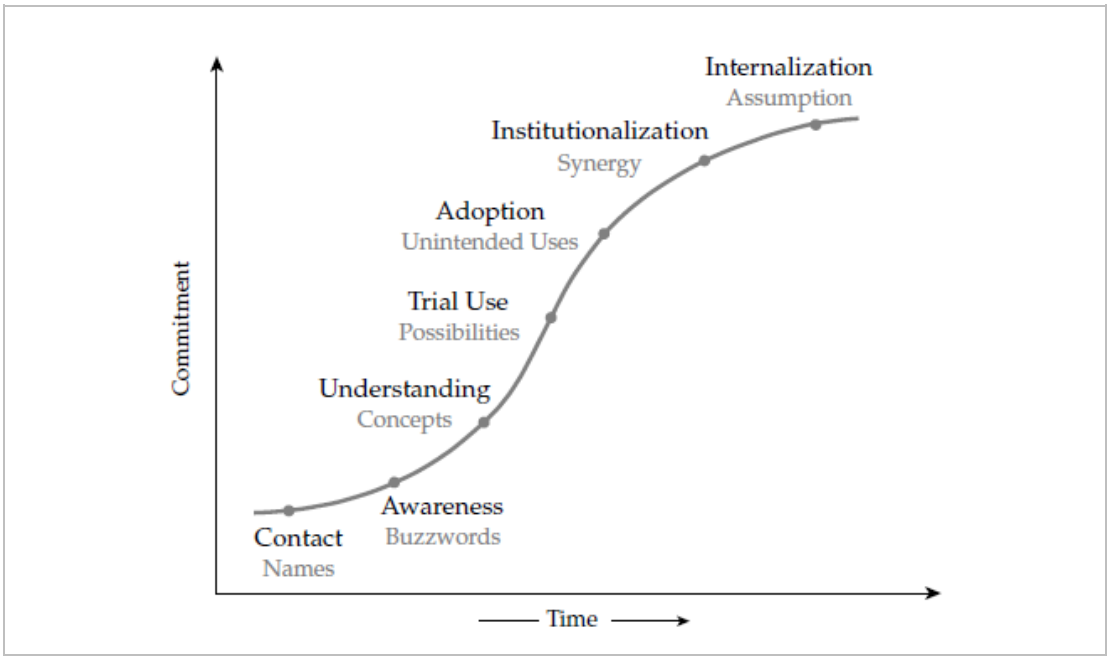


Figure 19: Patterson-Conner Adoption Commitment Curve (Adapted from Patterson and Conner)

The type of support people in different groups need to accelerate their adoption of the new practices depends on how committed they are to the change. The SEI uses a slightly modified version of the Patterson-Conner Adoption Commitment Curve (shown in Figure 19) to identify the stages that most individuals and groups go through when approaching adoption of a new set of practices or a new technology [Patterson 1982].

Figure 20 overlays Satir over this model; they work well with each other and provide two viewpoints from which to understand the process of change.

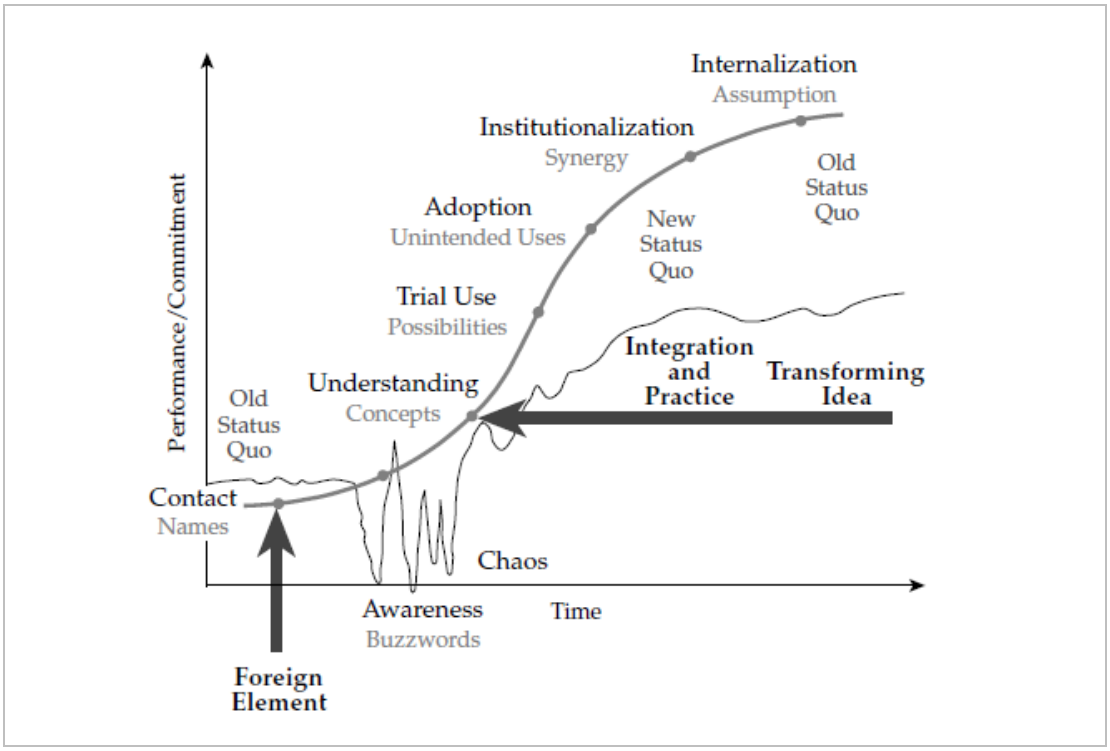


Figure 20: Satir Model Integrated Into the Adoption Commitment Curve [Miller 2006]

7.1.4 Finding/Selecting Pilot Projects

In general, there are two categories of pilots: *technical feasibility* pilots and *adoptability* pilots.

Technical feasibility pilots are useful if you are uncertain about the soundness of a new process. This type of pilot is suited to projects that are not on your organization's critical path with a team considered to be innovators or early adopters. Essentially, technical feasibility pilots determine if the technical components of the new process are performable and correct.

Adoptability pilots are performed when technical feasibility of the new process has been demonstrated. They evaluate whether what you've developed to support it—checklists, training, procedures—will work well with your mainstream organizational population. In this case, you are generally looking for a project that contains people who represent the general population to which you intend to deploy the new process. For an adoptability pilot, you do not want innovators; you want pragmatists, who need a reason to try something new. If the adoptability pilot works with them, chances are that the support products will work with the rest of the organization.

7.1.5 Adopter Analysis

Adopter analysis is a technique that comes from technology adoption. The idea is that individuals have some predisposition toward adopting a new technology or set of practices based on many different factors. The factors themselves aren't that important because most of the time, if you describe the "thing" to be adopted and the characteristics of several general adoption categories, most people can tell you where they would fit in relation to whatever technology/practice you

want them to adopt. The categories come from Everett Rogers' work on technology adoption [Rogers 2003] but are actually easier to understand based on their popularization in Geoffrey Moore's book *Crossing the Chasm* [Moore 2002]. Table 8 contains brief descriptions of the adopter categories used by Rogers and Moore.

Table 8: Rogers and Moore Adopter Categories [Rogers 2003, Moore 2002]

Adopter Category	Distinguishing Characteristics
Innovator	Gatekeepers for any new technology Appreciate technology for its own sake Appreciate architecture of technology Will spend hours trying to get technology to work Very forgiving of poor documentation, slow performance, incomplete functionality, etc. Helpful critics
Early Adopter	Dominated by a dream or vision Focus on business goals Usually have close ties with techie innovators Match emerging technologies to strategic opportunities Look for breakthrough Thrive on high-visibility, high-risk projects Have charisma to generate buy-in for projects Do not have credibility with early majority
Early Majority	Do not want to be pioneers (prudent souls) Control majority of budget Want percentage improvement (incremental, measurable, predictable progress) Not risk averse, but want to manage it carefully Hard to win over but are loyal once won
Late Majority	Avoid discontinuous improvement (revolution) Adopt only to stay on par with the rest of the world Somewhat fearful of new technologies Prefer preassembled packages with everything bundled
Laggard	Naysayers Adopt only after technology is not recognizable as separate entity Constantly point at discrepancies between what was promised and what is delivered

Adopter analysis helps identify individuals or groups that will be useful to you in different aspects of an adoption task. Innovators and Early Adopters are likely to volunteer for tasks in areas that affect them. However, they will probably be satisfied with a partial solution that might not satisfy other adopter types. For an adoption feasibility pilot, you should engage Early Majority participants. If you want to know what kind of transition mechanisms (i.e., things that help with the

communication or implementation of the new practices) need to be built over the longer term, you should engage Late Majority or Laggard participants.

Different adopter types typically move through change cycles (for example, the Satir cycle) at different speeds, so it is probable that you will find situations where some of the people in a group enthusiastically embrace a new set of practices and others drag their feet.

Finally, adopter type is not the only characteristic that is useful in choosing people to participate in different aspects of the adoption effort. You can also look at where they fit in your value network (see Section 7.4).

7.2 Lean and Agile

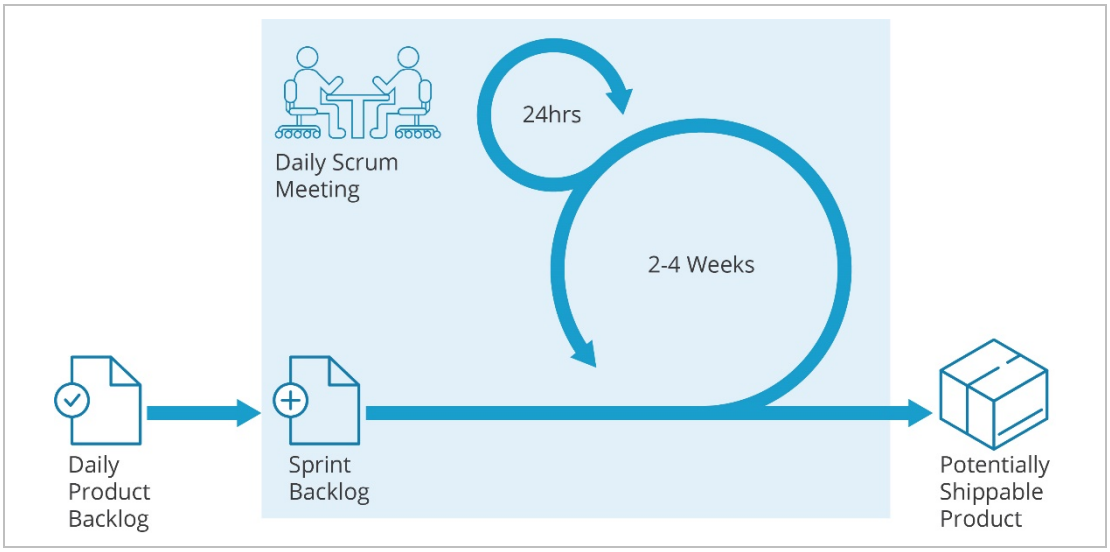


Figure 21: Scrum—Most Commonly Used Agile

Agile development approaches have been around for decades and were originally focused on team-management concepts for a single (small) software development team. Key Agile principles include ongoing involvement of the users, considering change as expected and positive, iterative development with short learning cycles that lead to evidence-based reviews, face-to-face communication, self-organizing teams, and continuous improvement through retrospective analysis. Additional methods and techniques that grew out of the original Agile Manifesto[3] include test-driven development, relative-effort estimation, and service-based architecture.

Scrum is the most commonly used Agile technology (Figure 21). When issues arose scaling these concepts for applications at systems level, the Agile concepts from software development were combined with the Lean concepts of flow, including value stream analysis, limiting work in progress, small batch sizes, and queue management to construct frameworks for working at scale.

[3] See www.agilemanifesto.com.

7.2.1 Principles

Merged Lean and Agile principles, as articulated in the SAFe Agile Framework,[4] provide a foundation for the implementation of DevSecOps that includes an Lean and Agile mindset. These principles are described in the following sections.

7.2.1.1 Take an Economic View

Most decisions are made by comparing values in some form, whether clearly stated or unconsciously considered. While the business world usually applies profit margin or return to the investor as primary measures of economic value, there are other types of value in systems development that allow decisions to be made in an economic framework. Delivering value early and often has a significant effect on the value to many types of users, which can be measured by considering the "cost of delay" in terms of missed productivity, lower quality of service, or higher mission success probability over time. Value can be a factor in prioritization and sequencing of work. Service orientation is based on delivering specific value under an agreed to level of service.

In highly regulated environments, economic value often includes an element related to the value of explicit compliance to the regulations that constrain the environment.

7.2.1.2 Apply Systems Thinking

It is important to consider the product to be developed as a system. Systems thinking also helps you understand the full value stream in acquisition and development organization. It considers a much broader set of factors than those related to requirements or how the product system interacts in operational context; it enables understanding the socio-technical system that encompasses the product and its context.

For HREs, a particular aspect of systems thinking is consideration of the regulatory system in which the products and operational context reside. The regulatory system for financial systems is governed by a different set of factors than the regulatory system for nuclear power plants, for example. Using the system factors of a nuclear power plant for a financial system is not likely to yield useful constraints.

7.2.1.3 Assume Variability; Preserve Options

In general, humans are not skilled at predicting the future. Threats, political landscapes, economics, and technology are changing too fast. Locking in a single detailed description of a system that will take a decade to build (even two or three years is difficult) can become a barrier as soon as anything changes. Acquirers and developers must acknowledge that variability and uncertainty are facts of life. It makes sense to invest in options where possible and make decisions at the last responsible moment (but no later).

[4] For richer definitions and a wealth of associated information see the Scaled Agile Framework website: https://www.scaledagileframework.com/

It is critical to ban the false-positive feasibility associated with deadline-driven, single-path approaches. False-positive feasibility occurs when we invest significant time and emotion into a description of a projected future state (THE requirements document, THE architecture diagram) and then resist any challenges to the "rightness" of the projection, even if that challenge comes from attempted implementation of the description that highlights areas where it fails to account for implementation realities. We falsely retain the view that the original description is feasible, despite implementation evidence to the contrary. Enlightenment usually comes too late in the traditional development process to allow for much change.

HREs are known for *not* having options along dimensions where the regulations are applied. Understanding where there are options versus where there are no options is a critical aspect of analysis in highly regulated environments.

7.2.1.4 Build Incrementally With Fast, Integrated Learning Cycles

Building incrementally with fast, integrated learning cycles helps to continuously focus on the most valuable work and receive feedback on your predictions (assumptions) quickly enough to eliminate much of the high cost of rework. It also supports the economic view associated with value delivery.

In HREs, this is a particularly important principle because learning what will and will not satisfy regulatory authorities needs to happen as the system evolves, not only at the end when a huge amount of rework could result from misunderstanding any number of constraints imposed by the regulatory environment.

7.2.1.5 Base Milestone Completion on the Objective Evaluation of Working Systems

Traditionally, milestones are treated as gates through which the development must pass to be allowed to continue forward. In traditional developments, they tend to be far apart in time and can involve a large group meeting with a large number of presentations, each with a large number of slides based on the results of the analysis of a large volume of documents by even more people. These reviews are often the only opportunity stakeholders have for providing input. Given the economic viewpoint and the principles of increments with fast learning cycles, this type of review is antithetical to those principles, not to mention expensive, cumbersome, and resulting in little learning about the system itself. Lean and Agile are based on the concept of objective, evidence-based reviews performed often, usually with some form of a working system. In the case of early learning, more and more dependence on evolving models of the system play a role in providing early system learning.

For HREs, mission threads are a common means of objective evaluations. Each milestone achieved extends the mission thread. Achieving capability-related requirements objectively measures mission-thread work progress.

7.2.1.6 Visualize and Limit WIP, Reduce Batch Sizes, and Manage Queue Lengths

Lean is based on the foundation of how work flows through the development process. This brings in a number of measures based on queueing and information theory as well as cognitive and

behavioral science. Visualizing and limiting work in progress regulates the number of tasks that are being worked on at any one time. It also keeps the human resources from an overwhelming number of context switches between tasks. Managing queue lengths supports the focus on WIP with the principle of "stop starting and start finishing," since the user gets value only with completed work, and work waiting in a queue is a waste. Don't start what you don't have the resources to finish. Small batch sizes mean that the work should be in small enough chunks that scheduling issues can be quickly resolved, and value is delivered in a timely fashion. Overall, while the flow is maximized, the organization is working at its full capacity and delivering the most value possible.

In an HRE—due to its typical siloed and isolated nature—full work-in-progress visualization can be a challenge. Security concerns determine access to all data. This can strongly limit full stakeholder visualization of any work in progress and in the queue.

7.2.1.7 Apply Cadence and Synchronize with Cross-Domain Planning

Uncertainty is a fact in system development, and predictive or "push" scheduling usually ignores this fact. Of course, management and users would like reasonable estimates for a variety of reasons. Setting cadences and synchronizing across the various teams and activities is the Lean answer to bounding the uncertainty. Cadences provide a predictable cycle of results and feedback opportunities as well as a foundation for useful, comparable metrics. Setting a synchronized cadence for the organization's work helps convert unpredictable events into predictable ones. It also forces developers to think in smaller batch sizes and allows for the orderly addition of new work. Finally, it improves the ability to understand, resolve, and integrate multiple teams' work as well as multiple stakeholder perspectives at the same time.

Due to the constant changes in mission-related scenarios, uncertainty is a large factor in HREs. Changes in priorities rapidly impact schedules, needs, and requirements. Agile's short development cycles coupled with DSO's continuous integration and operational insight provide rapid response to priority changes on a steady cadence.

7.2.1.8 Unlock the Intrinsic Motivation of Knowledge Workers

Unlike many workers, knowledge workers are usually much more capable and have a better understanding of the work they do than those who would try to "manage" them. The necessity is to help them achieve by creating an environment where they are most likely to thrive. Motivation comes not necessarily through command, control, or salary (although that is a factor), but through less-tangible things like autonomy, mission, and minimum constraints. Respect and a willingness to collaborate are also motivation strategies.

In an HRE, these principles hold true. The mission is undoubtedly the main motivator for knowledge workers. Also note that the structured top-down design of management can hinder progress of these workers. They may sense a "caged in" environment disallowing the open exploration of ideas.

7.2.1.9 Decentralize Decision Making

Decentralized decision making is a key component for achieving the shortest sustainable value delivery time. Decisions that require a chain of command elevator can destroy cadence, delay progress, and often are lower quality, particularly if the information needed to make the decision must be loaded in the elevator. Strategic decisions are more effective if centralized, but all others should be delegated to the level closest to the information involved. That may mean that some information previously seen as "privileged" should be pushed out to the edge, so that those closest to the issue can make more informed decisions—thus freeing up the managers/executives to focus on the strategy.

HREs are well known for having the "chain-of-command elevator" and can cause most, if not all, of the issues mentioned above. In small groups of developers, management can be flattened to an immediate supervisor and thus allow rapid progress and help team members flourish. It is best for these supervisors to work the elevator while the developers progress with "approval pending" status.

7.3 Systems Engineering

More and more acquisitions of all sizes and domains are seeking the benefits of Lean, Agile, and DevSecOps (LADSO) principles. While these principles are usually associated with software engineering [McQuade 2019], implementing them requires some significant changes in the way systems engineering is performed [Wrubel 2014]. Systems engineering as generally practiced in large, complex cyber-physical systems development and is rooted in waterfall-based, plan-driven, low-uncertainty, highly predictable programs. LADSO is built around iterative, high-uncertainty, rapidly changing threat and STEEP (social, technological, economic, environmental, and political) factors. These different assumptions force changing the fundamental nature of systems engineering when supporting LADSO projects. Table 9 illustrates some of these differences.

Table 9: Fundamental Differences Between Traditional and LADSO SE Environments (Adapted from Wrubel 2014)

Systems Engineering as Generally Practiced	LADSO-Based Systems Engineering
Large-batch processing (products, documents, events)	Small batch processing (products, documents, events)
Single-pass lifecycle (all requirements done before the design is initiated; all design done before implemented)	Incremental, iterative multi-pass lifecycle (small batches of products and their artifacts built/tested iteratively, delivered incrementally)
Single-point design	Set-based design
Solution intent fixed early (all requirements defined in detail early)	Most of solution intent variable early (only near-term requirements in detail; others are higher level with details based on learning)
Fixed point, large-batch integration (components all "done" before integration occurs)	Cadence-based, small-batch integration used as frequently as feasible; integrate as available to prevent rework (for software, may be daily)
Centralized, command-and-control leadership	Mix of centralized and decentralized leadership; "servant leadership"

Systems Engineering as Generally Practiced	LADSO-Based Systems Engineering
Detailed, allocated baseline early; high overhead change management practices in play for the rest of development	Allocated baseline level of abstraction allows learning-based change throughout development; no high-overhead change processes
Hardware and software treated separately, integrated late	Hardware and software treated together, integrated early and frequently
Large-batch model-based engineering used to improve the detail of requirements and design prior to implementation; often abandoned after design	Model-based engineering moves between large- and small-batch modeling activities; models and simulations flow with implementation and support the full lifecycle, development through sustainment
Projective (to be) requirements and design documentation dominates early discussion and activities	Projective documentation takes second place to working prototypes and demos; used to guide, not specify; documentation is as-built, not to-be.
Systems engineering function separate from hardware and software development functions	Systems engineering function integrated into capability-focused teams that include all disciplines needed (HW, SW, UX, reliability, etc.)
Component-based work breakdown structure	Capability-based work breakdown structure
Systems engineering primarily as artifact transformation (e.g., Requirements->Architecture->Design)	Systems engineering as a service (facilitation of artifact transformation; focus on communication, coordination, conflict resolution, collaboration)
System architecture decisions neutral to development approach	System architecture decisions strongly support loosely coupled components/subsystems, especially for software capabilities
Assumption that early work is correct and that late failure is a surprise	Assumption that early work is inherently flawed, and learning from early failure feeds the evolution of knowledge about the system
System and software architecture frozen early	Intentionally extendable and iteratively evolving architecture throughout development and sustainment
User participation only early and late	User participation continuous throughout lifecycle

As is evident from the table, systems engineering in LADSO environments is significantly different in a number of ways. There is a definite tension between the flexibility and adaptability of software and the built-in consideration of milestones, baselines, and controls in systems engineering. While software-only systems may face less resistance than the DSO and similar approaches, embedded systems and safety-critical systems have been mostly skeptical of CI/CD in general. The one place where systems folks may be able to support DevOps is in the field of architecture.

7.3.1 Architecture

The architecture of the system under development can enable or impede deployability goals. Steffany Bellomo and other SEI researchers are convinced that architecture design decisions and tradeoffs can impact the feasibility of DevOps practices; poor decisions can lead to the infeasibility of critical activities such as continuous build integration, automated test execution, and operational support [Bellomo 2014, Bass 2015]. For example, they observed cases where a tightly coupled component architecture became a barrier to continuous integration because small changes required an increasingly time-consuming rebuild of the entire system. Similar issues are

- A system doesn't have architectural interfaces for test automation and manual tests are slow; tests are skipped.

- The architecture creates deployment complexity, and error-prone manual steps prevent release; weeks/months pass without release.

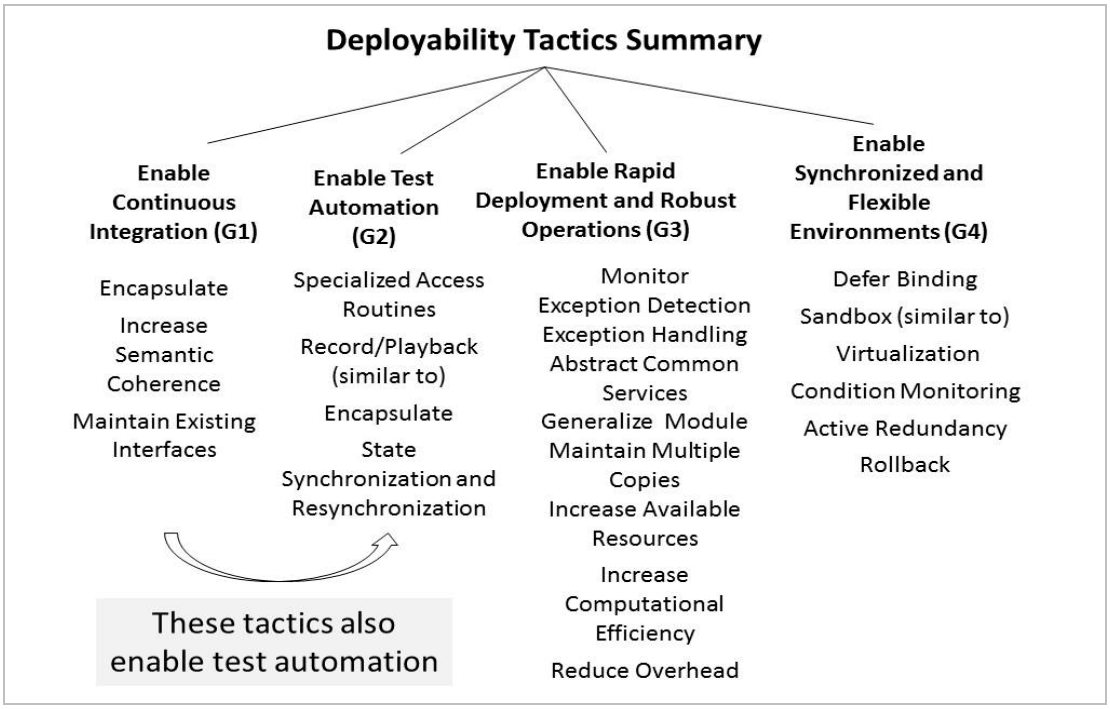


Figure 22: Deployability Architecture Tactics Tree

Re-architecting the system to fix problems such as these can require significant work and, in some cases, become a form of technical debt, resulting in high expenditures of time, money, and effort release after release [Kruchten 2012]. Architecting for deployability can lead to greater benefits from adopting DevOps practices. Criteria and best architectural practice for deployability are still being researched; the Deployability Tactics Summary seen in Figure 22 was one of the products of the research.[5]

[5] For more information see a video of Stephany Bellomo presenting *Architectural Implications of DSO* at https://youtu.be/AWA-oN8rOgo.

7.4 Value Stream and Network Visualizations

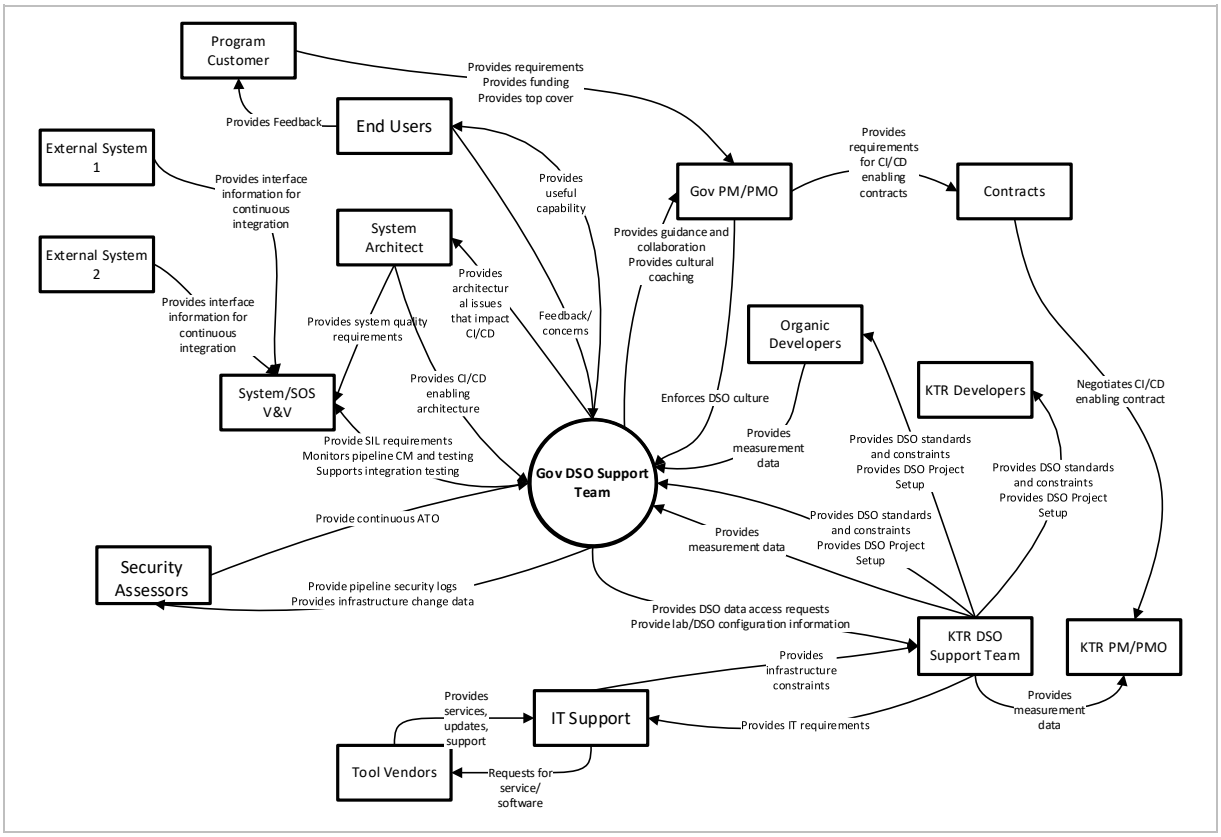


Figure 23: Example of a Value Network

Value networks capture the value flow through the organization. A value network starts with your team in the middle of a network diagram. Additional nodes in the network represent different groups or individuals that interact with the team. The arcs capture values exchanged among the different nodes and the team. The kinds of value exchanged may remind you about different kinds of data, resources, decisions, special skills, measurements, or other services or demands. Figure 23 is an example of a value network for DSO adoption.

A value stream gives an ordering to many of the arcs in the value network that shows the general addition of values as the product evolves. Keeping the value stream in concert with changes in the process would provide evidence for how the DSO pipeline is improving outcomes (particularly if you can simultaneously show measurable improvements in product quality).

Creating the value network is an iterative process usually accomplished by a number of people engaging in a series of group work sessions. Information gathered in the previous sections is captured using nodes and arcs as illustrated in Figure 23. Using sticky notes of two colors (one for nodes and one for arcs) arranged and connected on a whiteboard provides an easy way to collaboratively evolve the network. Photos can easily capture ongoing development as more nodes and arcs are identified or repositioned at each subsequent meeting; once there is an agreement, any graphics software tool can be used to "formalize" the resulting network.

7.5 Policy

The Defense Innovation Board's Software Acquisition and Practices (SWAP) study identifies a long trail of recommendations and reports from studies financed by the DoD to address software development and acquisition [McQuade 2019]. More importantly, they provided a comprehensive way forward that includes many enablers for DevOps, SecDevOps, and other CI/CD-related technology. The following appears in the Defense Innovation Board Ten Commandments of Software:[6]

> *Commandment #4. Adopt a DevOps culture for software systems. "DevOps" represents the integration of software development and software operations, along with the tools and culture that support rapid prototyping and deployment, early engagement with the end user, and automation and monitoring of software. These techniques should be adopted by the DoD, with appropriate tuning of approaches used by the Agile/DevOps community for mission-critical, national security applications. Open source software should be used when possible to speed development and deployment, and leverage the work of others. Waterfall development approaches (e.g., DOD-STD-2167A) should be banned and replaced with true, commercial Agile processes. Thinking of software "procurement" and "sustainment" separately is also a problem: software is never "finished" but must be constantly updated to maintain capability, address ongoing security issues, and potentially add or increase performance* [McQuade 2019].

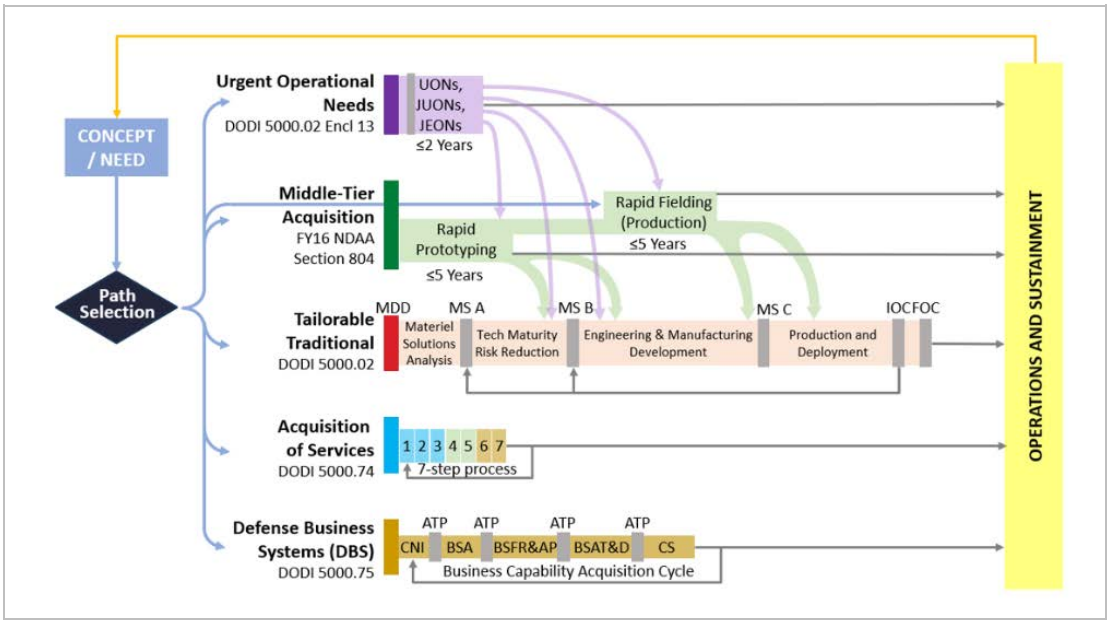


Figure 24: Adaptive Acquisition Framework (https://aaf.dau.edu/aaf/)

While not guaranteed to be acted upon, the current OSD and Service acquisition goal is to make significant changes in acquisition and development policies. OSD is recrafting DoD 5000 to

[6] All of the DIB work on software in the DoD can be found at https://innovation.defense.gov/software/ .

address multiple acquisition paths as shown in Figure 24. The work is continuing as this guide is written, and information can be found at the website in the caption.

Some issues are outside of the DoD span of control and include legal concerns such as those related to USC Title 10 and the Clinger-Cohen Act. However, the moving of most acquisition decision authority from OSD to the individual Services provides additional changes that can be made in the policies within the services.

Finally, the amount of work on software acquisition and development and the desire to support proven best practices with policy rather than preventing them, seem to be a positive indicator for greater application of Lean, Agile, DevOps, DevSecOps and other adaptive techniques.

Appendix A: Collected Activity Summaries

Objective: Establish your vision.

Activity: Identify/build your vision.

Context	*This activity is the first step in adoption. Without a shared vision the odds of successful adoption are significantly reduced.*
Purpose	*The purpose of this activity is to achieve a vision that supports readiness and fit analysis and adoption planning activities, maintains the long view, and is validated and agreed to by stakeholders and practitioners.*
Overview	*Develop a common understanding of a desirable outcome of the adoption activities. What will success look like?*
Primary Actors	*This activity involves leadership, management, and representative practitioners (organic or contracted).*
Inputs	*Inputs to this activity include decision information, survey questions, access to personnel, and means of capturing the information gathered.*
Outputs	*Outputs from this activity include documentation of the information gathered and the vision identified.*
Resources	
Tips, Tricks, and Wisdom	*Use visualizations (e.g., value networks, organizational structures, process diagrams) to promote participation and reduce unnecessary "wordsmithing."*

Objective: Determine the readiness to adopt.

Activity: Understand your context.

Context	*This is the first step in adoption. Understanding the current environment is critical to how this activity relates to overall adoption.*
Purpose	*This activity supports readiness, fit analysis, and adoption planning activities.*
Overview	*Collect and validate information about context.*
Primary Actors	*This activity involves everyone.*
Inputs	*Inputs to this activity include access to personnel and means of capturing information gathered.*
Outputs	*Outputs from this activity include documentation of the information gathered.*
Resources	*Section 7.4*
Tips, Tricks, and Wisdom	*Using visualizations (e.g., value networks, organizational structures, process diagrams), promote participation and reduce unnecessary "wordsmithing."*

Activity: Implement the readiness and fit analysis process.

Context	***This activity establishes the enablers and barriers in your organization associated with adopting DSO. Having the value map and profiles make this an easier task. While this can produce significant concern, particularly if the barriers outweigh the enablers, it is critical to manage expectations and conduct rational planning.***
Purpose	***This activity captures the current organization's readiness to adopt DSO in terms of risks, opportunities, barriers, and enablers. It is a significant planning asset.***
Primary Actors	***This activity involves the manager, teams, S-CSs, and the culture change coach.***
Relevant/Key Events	***Events include the decision to adopt DSO and a DSO Posture Assessment.***
Activity Input(s)	***Inputs to this activity include the results of 4.1.1 and 4.2.1.***
Activity Outputs(s)	***Outputs from this activity include an adoption risk assessment with identified mitigation approaches and proposed adoption progress measures.***
Other Resources	***Other resources include the RFA White Paper [Miller 2014], RFA Presentation slides, and RFA Forms [Miller 2014].***
Tips, Tricks, and Wisdom	***A workshop approach to this analysis is faster but requires more coordination. It is just as important to identify enablers as risks.***

Objective: Develop an adoption/transition strategy.

Activity: Identify your DSO adoption goal(s).

Context	***The DSO adoption goals are the core guidance for all of your strategic and tactical planning. They continue to be evaluated and evolved throughout the adoption and management process.***
Purpose	***This activity produces a set of goals aligned with the DSO principles that identify the specific outcomes desired from the adoption of DSO along with broad indicators of accomplishment.***
Primary Actors	***This activity involves team lead(s), customers/users, developers, Operations-Deployment, IT, Security, and the pipeline (PL) architect.***
Activity Inputs	***Inputs to this activity include a list of success-critical stakeholders (S-CSs).***
Activity Outputs	***Outputs from this activity include the initial goals statement.***
Other Resources	***Other resources include the blog post DevOps and Your Organization: Where to Begin (*** https://insights.sei.cmu.edu/devops/2014/12/devops-and-your-organization-where-to-begin.html***) and the webinar Three Secrets to Successful Agile Metrics (https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=507850).***
Tips, Tricks, and Wisdom	***The goals can (and most likely will) evolve as the adoption progresses. Give priority to cultural outcomes and stakeholder pain points. Review the goals and their measures regularly.***

Activity: Establish the initial adoption scope.

Context	*The DSO adoption goals are most likely visions for the future. There needs to be an identified scope for initial adoption. Is it one team, one organization, or an enterprise? The answer to this question will determine how you will size the increments and will be highly dependent on the resources available over time.*
Purpose	*This activity identifies the specific goals to be addressed in the current adoption effort.*
Primary Actors	*This activity involves team lead(s), customers/users, developers, Operations-Deployment, IT, Security, and the pipeline (PL) architect.*
Activity Inputs	*Inputs to this activity include the overall adoption goals.*
Activity Outputs	*Outputs from this activity include specific goals to be addressed in the initial effort.*
Other Resources	*Other resources include Section 7.1.4 and the CMMI Survival Guide [Miller 2006].*
Tips, Tricks, and Wisdom	*This is where understanding technical feasibility pilots and adoption feasibility pilots can be useful.*

Activity: Propose change (transition) mechanisms.

Context	*Change is not a passive activity. There must be specific actions taken to reach out to stakeholders and practitioners to enable and reinforce change.*
Purpose	*This activity produces a set of transition mechanisms that are tailored to the scope and target of the adoption effort.*
Primary Actors	*This activity involves team lead(s), customers/users, developers, Operations-Deployment, IT, Security, and the pipeline (PL) architect.*
Activity Inputs	*Inputs to this activity include specific (scoped) goals.*
Activity Outputs	*Outputs from this activity include a set of communication and implementation mechanisms and actions that propel changes in technology and culture.*
Other Resources	*Other resources include the CMMI Survival Guide [Miller 2006] and Section 7.1.3.*
Tips, Tricks, and Wisdom	*Two types of failure modes are frequently seen: (1) focusing only on communication mechanisms (the "train people to death" failure mode) and (2) the opposite—providing new procedures, measures, and other implementation mechanisms before enough communication has occurred for staff to understand what the goal of the adoption is.*

Objective: Plan your next adoption activities.

Activity: Identify resources.

Context	*Plans without appropriate resources are worthless. Iterative planning based on realistic availabilities is necessary for success.*
Purpose	*This activity identifies the resources (e.g., skilled staff, facilities, materials) that are needed to accomplish the agreed-upon goals.*
Primary Actors	*This activity involves team lead(s), customers/users, developers, Operations-Deployment, IT, Security, the pipeline (PL) architect, and the financier.*
Activity Inputs	*Inputs to this activity include the identified scope and DSO adoption goals.*
Activity Outputs	*Outputs of this activity include a list of resources and when and for how long they are needed.*
Other Resources	
Tips, Tricks, and Wisdom	*The amount of "free stuff" on the Internet in topic areas related to DSO is staggering. Don't ignore free training and communication resources that are relevant to your setting.*

Activity: Develop a backlog and initial increment map.

Context	*There is nothing like using the techniques you are espousing to help your team and organization understand that this effort is serious.*
Purpose	*This activity produces an initial backlog of items that need to be accomplished within (1) the next increment and (2) a breakdown of those items into those that can be accomplished in the next iteration.*
Primary Actors	*This activity involves team lead(s), customers/users, developers, Operations-Deployment, IT, Security, and the pipeline (PL) architect.*
Activity Inputs	*Inputs to this activity include the adoption strategy, adoption goals, and adoption resources.*
Activity Outputs	*Outputs from this activity include a roadmap for the next increment and the backlog of both high-level and more granular product backlog items.*
Other Resources	*Other resources include Section 7, Appendix B, and any number of books or websites describing fundamental Lean and Agile software development practices.*
Tips, Tricks, and Wisdom	*User stories are used in some settings; they may work if the environment is already accustomed to them, but may be awkward for some service-oriented tasks.*

Activity: Develop a communications plan.

Context	***Communication is the lifeblood of change management. Without a well-thought-out plan for how information is distributed, how questions are answered, and how progress and stories are captured, inertia will take over and the adoption will fail.***
Purpose	***This activity produces a communication plan tailored to the adoption activities and the environment and culture of the change target.***
Primary Actors	***This activity involves team lead(s), customers/users, developers, Operations-Deployment, IT, Security, and the pipeline (PL) architect.***
Activity Inputs	***Inputs to this activity include vision goals, scoped goals, and RFA results.***
Activity Outputs	***Outputs from this activity include the communications plan.***
Other Resources	***Other resources include the blog post DevOps and Your Organization: Where to Begin (***https://insights.sei.cmu.edu/devops/2014/12/devops-and-your-organization-where-to-begin.html***)***
Tips, Tricks, and Wisdom	

Objective: Change the culture.

Activity: Monitor cultural change progress.

Context	***This activity is a continuous monitoring of the organizational culture to understand progress with respect to the measures adopted in response to the Adoption Culture Risk Assessment.***
Purpose	***This activity captures the current state of the supporting DSO culture in terms of risks, opportunities, barriers, and enablers.***
Overview	***The cultural profile is established and monitored at an appropriate cadence. Information comes from the culture- and risk-related measures collected periodically. This activity is often associated with technical milestones.***
Primary Actors	***This activity involves the manager, teams, S-CSs, and the culture change coach.***
Activity Input(s)	***Inputs to this activity include the current Adoption Culture Risk Assessment, adoption measures, and the current Culture Action Plan.***
Activity Outputs(s)	***Outputs of this activity include culture change information, the revised Adoption Culture Risk Assessment, the revised Culture Action Plan, and new culture awards.***
Other Resources	***Other resources include the RFA White Paper [Miller 2014], change management literature, and case studies of similar organizations.***
Tips, Tricks, and Wisdom	***Having leadership exhibit the appropriate behaviors (or not) is a significant measure of progress.***

Activity: Influence change.

Context	*As cultural goals and risks are addressed and evaluated, actions are continuously taken to improve and maintain a DSO-supportive culture.*
Purpose	*This activity supports DSO culture acceptance and maintenance.*
Overview	*Change mechanisms are used to improve specific concerns or problems identified in monitoring. The measurement strategy and metrics may be adjusted to more accurately capture progress around the specific issues.*
Primary Actors	*This activity involves the manager, teams, and S-CSs.*
Activity Input(s)	*Inputs to this activity include current cultural needs, barriers, or enablers.*
Activity Outputs(s)	*Outputs from this activity include specific culture-related actions added to the increment plans and feedback from actions.*
Other Resources	*Other resources include the mechanisms identified in the adoption strategy and Table 4. Typical transmission mechanisms by Adoption Commitment Curve Stages.*
Tips, Tricks, and Wisdom	*Be innovative in responding to issues; don't overuse one or two mechanisms. Enlist leadership to exhibit and reinforce needed behaviors.*

Objective: Build a DSO pipeline.

Activity: Consolidate pipeline requirements.

Context	*This activity uses the information gathered in adoption preparation to capture the software and hardware requirements for the pipeline. Pipeline construction can be iterative or incremental. The requirements may evolve, but there are specific questions that need to be answered before construction begins.*
Purpose	*This activity captures the initial requirements for the pipeline based on information gathered in Epic 1 and a set of questions provided.*
Overview	*This activity draws on the information developed in preparation activities and establishes the requirements for a DSO pipeline that meets the context, readiness profile, and strategy of the organization.*
Primary Actors	*This activity involves the manager, team leads, and the pipeline (PL) architect.*
Activity Input(s)	*Inputs to this activity include the goals statement, DSO Adoption Strategy and Plan, DSO Posture Assessment Report, Technical Inventory, and Security Profile.*
Activity Outputs(s)	*Outputs from this activity include the pipeline requirements.*
Other Resources	*Other resources include the Pipeline Requirements Questionnaire.*
Tips, Tricks, and Wisdom	*If selecting open source components, verify they provided in-help guides connectivity scripts to other components. Many components are commonly used together in pipelines; leverage this to reduce scripting and configuration needs.*

Activity: Identify and acquire needed components.

Context	*Identifying and acquiring the infrastructure and components are a critical part of achieving DSO benefits. It should be approached collaboratively and consider the stakeholders' expectations and the available technical and security environment. The components may be selected and acquired in a single activity, or they can be incrementally acquired as resources become available. The DSO Adoption Strategy and Plan should reflect the approach.*
Purpose	*This activity provides the building blocks for the DSO pipeline.*
Overview	*This task translates the requirements into a set of ordered software and hardware components that will make up the pipeline infrastructure.*
Primary Actors	*This activity involves the manager, team leads, the pipeline architect, Procurement, and IT.*
Activity Input(s)	*Inputs to this activity include the Technical Profile, Goals Statement, and Pipeline Requirements.*
Activity Outputs(s)	*Outputs from this activity include pipeline components.*
Provided Work Aids	*Work aids provided include Pipeline Component Considerations and the Pipeline Design Template.*
Other Resources	
Tips, Tricks, and Wisdom	*Making pipeline decisions should be aligned with the culture assessment and strategy and never be the first activity undertaken. Without culture change, the pipeline will have only a minimal effect on overall SDLC performance.*

Activity: Install and launch the pipeline.

Context	*Technical implementation of the pipeline involves integrating the pipeline software and hardware, understanding the connectivity and creating links, and capturing the process as defined by the selected components.*
Purpose	*This activity creates the pipeline infrastructure and process.*
Overview	*This activity integrates the pipeline component in an iterative manner, identifying and documenting the roles and responsibilities of the pipeline components.*
Primary Actors	*This activity involves the pipeline (PL) architect.*
Activity Input(s)	*Inputs to this activity include pipeline requirements and pipeline components.*
Activity Outputs(s)	*Outputs from this activity include an operational pipeline with a validated use process.*
Other Resources	*Other resources include online resources associated with the components to support development of component communication scripts as needed.*
Tips, Tricks, and Wisdom	*In most installations, the default setup for any component will suffice minus scripts to connect with other components.*

Activity: Test the pipeline.

Context	*Before trial use, the pipeline should be tested end to end.*
Purpose	*This activity validates the overall functionality of the installed pipeline.*
Overview	*This activity tests the overall functions of the pipeline as an integrated tool. Concurrently, it captures the user role responsibilities to create a documented process.*
Primary Actors	*This activity involves software engineers, operations engineers, pipeline (PL) architect*
Activity Input(s)	*Inputs to this activity include the Installed pipeline and the test project.*
Activity Outputs(s)	*Outputs from this activity include test results.*
Other Resources	*Other resources include information you can obtain by following each component's official online documentation for recommended testing.*
Tips, Tricks, and Wisdom	*Use a simple project that includes aspects that use every component of the pipeline, including testing (manual and automatic).* *Your very first run will likely have connectivity problems. This is normal. Resolve and continue testing. Repeat testing until you complete one successful run of the entire pipeline.*

Activity: Reassess your DSO posture.

Context	*This activity happens at each step of the adoption process.*
Purpose	*This activity quantifies the impact of the pipeline on providing your desired SDLC.*
Overview	*This activity determines if the pipeline addresses its associated DSO technical goals and provides insight into progress.*
Primary Actors	*This activity involves software engineers, QA, requirements engineers, and the program manager.*
Activity Input(s)	*Inputs to this activity include a detailed pipeline architecture with performance metrics and testing results, the questionnaire below, and results from the initial DSO Posture Assessment.*
Activity Outputs(s)	*Outputs from this activity include answers to the questionnaire, a determination of progress, and potential remediation action tasks.*
Other Resources	
Tips, Tricks, and Wisdom	*Repeat pipeline testing if remediation action items are put forth. Continue with this testing until no action items are given. Be aware that the pipeline may introduce new issues not previously present; remediate and repeat test in this case.*

Objective: Conduct trial use.

Activity: Select pilot tasks/projects/work.

Context	*Selecting pilots is the first step in trial use. It attempts to maximize the information gained from Trial Use.*
Purpose	*This activity creates a list of tasks or projects to act as pilots.*
Overview	*Trial use pilot(s) selections are made collaboratively, using established criteria for considering the projects available.*
Primary Actors	*This activity involves the program manager and the pipeline (PL) architect.*
Activity Input(s)	*Inputs to this activity include DSO Goals, the latest version of the DSO Posture Assessment, the installed and tested pipeline, and validated process guidance.*
Activity Outputs(s)	*Outputs from this activity include a selected pilot project.*
Other Resources	*Other resources include Sections 7.1.4 and 7.1.5.*
Tips, Tricks, and Wisdom	*Senior leadership should buy in on selected project.* *Don't choose a multi-year project for the trial run. It may be too long before remediation can occur.*

Activity: Conduct pilot tasks/projects/work.

Context	*The pilots will identify technical, process and culture conflicts, mismatches, errors, and improvements.*
Purpose	*This activity supports and monitors the pilots.*
Overview	*The pilots are conducted on the operational pipeline using the validated use process.*
Primary Actors	*This activity involves the program manager, the pipeline (PL) architect, customers, end users, software engineers, requirements engineers, the quality assurance engineer, the operations engineer, and the security engineer.*
Relevant/Key Events	*Events include the commencement of project, schedules milestones, final delivery into production, and handover to the customer.*
Activity Input(s)	*Inputs of this activity include selected pilots, requirements, personnel, the process of using the pipeline, the schedule, milestones.*
Activity Outputs(s)	*Outputs of this activity include the delivered product to the customer.*
Other Resources	
Tips, Tricks, and Wisdom	*Before the project starts, hold meetings and talks to get personnel thinking in the DSO style, so they mentally plan and make decisions conducive to DSO.*

Activity: Reassess your DevOps posture.

Context	*This happens at every step of the adoption process.*
Purpose	*This activity is a full assessment of post-trial use.*
Overview	*This is a final check on the initial pipeline deployment.*
Primary Actors	*This activity involves all stakeholders.*
Activity Input(s)	*Inputs to this activity include the DevOps posture questionnaire, trial-use issues, met and failed schedule dates, and milestones.*
Activity Outputs(s)	*Outputs from this activity include questionnaire answers and remediation action items.*
Other Resources	*Other resources include questionnaire answers for all previous posture assessments related to this project.*
Tips, Tricks, and Wisdom	*This was the first run; it won't be perfect. Learn from it, make changes to improve posture, and move on to the next project.*

Objective: Monitor the ecosystem.

Activity: Establish a measurement program.

Context	*This activity is a long-term performance metric tracking pipeline, process, and culture.*
Purpose	*This activity determines and creates long-term metrics for DSO effectiveness.*
Overview	*Senior leadership and program managers determine the best metrics for long-term monitoring of DSO effectiveness across the organization.*
Primary Actors	*This activity involves senior leadership, program management, and software engineers.*
Activity Input(s)	*Inputs to this process include formalization goals.*
Activity Outputs(s)	*Outputs of this activity include a list of desired data for long-term analysis of DSO.*
Other Resources	*Other resources include the webinar Three Secrets to Successful Agile Metrics.*
Tips, Tricks, and Wisdom	*Remember your analysis should focus on DSO culture and process. Don't get stuck only on the pipeline performance.*

Activity. Regularly reassess your DSO technical and cultural posture.

Context	*This activity is a routine assessment to gauge long-term impact and effectiveness of DSO.*
Purpose	*This activity assures DSO impact and effectiveness trends have not fallen to near unacceptable levels and fixes those that may be declining.*
Overview	*This activity is a progress assessment that looks at all aspects of the DSO adoption effort.*
Primary Actors	*This activity involves all stakeholders.*
Activity Input(s)	*Inputs to this activity include results of all previous DSO assessments and the assessment questionnaire in 2.1.5.*
Activity Outputs(s)	*Outputs to this activity include answers to the questionnaire, the current DSO posture, and remediation action items provided to incremental planning activities.*
Other Resources	
Tips, Tricks, and Wisdom	*Watching assessment trends over time will clearly indicate cultural acceptance and/or defiance.*

Objective: Extend DSO (Institutionalize).

Activity: Establish formalization goals.

Context	*These goals will institutionalize DSO as regular practice within your organization.*
Purpose	*This activity establishes the use of DSO on all projects, organization-wide.*
Primary Actors	*This activity involves senior leadership and program managers.*
Overview	*Senior leaders and program managers collaboratively identify goals that set the standard of practice in using DSO for all future projects.*
Activity Input(s)	*Inputs to this activity include trial usage final reports, DSO post-implementation assessment findings, and the DSO process and pipeline.*
Activity Outputs(s)	*Outputs from this activity include an organization-wide mandate detailing the mechanics of DSO usage on all projects.*
Other Resources	
Tips, Tricks, and Wisdom	*Several pilots will be needed for the whole organization to accept DSO as "the way we do things."*

Activity: Document and train personnel.

Context	*This activity includes continuous teaching of personnel in all areas of DSO.*
Purpose	*This activity keeps personnel up to date on all DSO-related issues.*
Overview	*Training, based on the latest process, is created, offered, and delivered to all team members as needed.*
Primary Actors	*This activity involves program managers and senior leadership.*
Activity Input(s)	*Inputs to this activity include new training materials and documented updates on DSO.*
Activity Outputs(s)	*Outputs from this activity include training videos, slides, books, meetings, and Q&A sessions.*
Other Resources	*Other resources include third-party trainers, various media with new training, and/or DSO changes.*
Tips, Tricks, and Wisdom	*Use training to keep personnel current on DSO technology and process refinements.* *Over training or too-early training can be detrimental.*

Appendix B: Additional SEI DSO Resources

DevOps and DevSecOps [Waits 2015]

Blog Post: Introduction to DevOps

This blog post is a short overview of the DevOps concept in industry and the evolution of DevSecOps.
https://insights.sei.cmu.edu/devops/2014/03/an-introduction-to-devops.html

Webinar: DevSecOps Implementation in the DoD: Barriers and Enablers

Today's DoD software development and deployment is not responsive to warfighter needs. As a result, the DoD's ability to keep pace with potential adversaries is falling behind. In this webcast, panelists Hasan Yasar, Eileen Wrubel, and Jeff Boleng discuss potential enablers of and barriers to using modern software development techniques and processes in the DoD or similar segregated environments. These software development techniques and processes are commonly known as DevSecOps.
https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=544953

Video/Podcast: Agile DevOps

In this video, the SEI's Eileen Wrubel and Hasan Yasar discuss how Agile and DevOps can be deployed together to meet organizational needs. “Continuous delivery is already part of Agile principles. In a DevOps world, we are seeing continuous delivery. We are seeing continuous integration. We are talking about continuous deployments. These are the key principles of DevOps. As techniques, these make all of these Agile principles achievable.”
https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=517941

Technical Report: Infrastructure as Code

This report concludes work on a research project that explores the feasibility of infrastructure as code, summarizing the problem addressed by the research, the research solution approach, and results. *Infrastructure as code* (IaC) is a set of practices that use code to set up virtual machines and networks, install packages, and configure environments. Successful IaC adoption by software sustainers requires a broad set of skills and knowledge. This project addresses the problem of accelerating IaC adoption among software sustainment organizations.
https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=539327

Blog Post: Information Visualization as a DevOps Monitoring Tool

In this blog post, the first in a series on Information Visualization in DevOps, researcher Luiz Antunes explores how visual graphics can assist in the DevOps process.
https://insights.sei.cmu.edu/devops/2017/05/information-visualization-as-a-devops-monitoring-tool.html

Blog Series: Implementing DevOps Within Highly Regulated Environments

This blog post series is based on *Implementing DevOps Practices in Highly Regulated Environments*, a paper by José Morales, Hasan Yasar, and Aaron Volkmann.
https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=531308

The series expands on the paper and discusses the process, challenges, approaches, and lessons learned in implementing DevOps in the software development lifecycle (SDLC) within highly regulated environments (HREs).

First Post: Challenges to Implementing DevOps in Highly Regulated Environments
https://insights.sei.cmu.edu/devops/2019/01/challenges-to-implementing-devops-in-highly-regulated-environments-first-in-a-series.html

Second Post: Expectations for Implementing DevOps in a Highly Regulated Environment
https://insights.sei.cmu.edu/devops/2019/02/expectations-for-implementing-devops-in-a-highly-regulated-environment.html

Third Post: Establishing the Pre-Assessment DevOps Posture of an SDLC in a Highly Regulated Environment
https://insights.sei.cmu.edu/devops/2019/04/establishing-the-preassessment-devops-posture-of-an-sdlc-in-a-highly-regulated-environment.html

Fourth Post: Performing the DevOps Assessment
https://insights.sei.cmu.edu/devops/2020/01/performing-the-devops-assessment-fourth-in-a-series.html

Fifth Post: Formalizing DevOps Assessment Findings and Crafting Recommendations
https://insights.sei.cmu.edu/devops/2020/02/formalizing-devops-assessment-findings-and-crafting-recommendations-fifth-in-a-series.html

Technology Adoption

White Paper: Is your organization ready for Agile?

This white paper addresses the factors that should be considered when adopting Agile practices and processes in regulated environments (e.g., DoD, IRS, FDA, other government agencies). The paper is based on the SEI's Readiness and Fit Analysis process. All software engineering and management practices are based on cultural and social assumptions. When adopting new practices, leaders often find mismatches between those assumptions and the realities within their organizations.
https://resources.sei.cmu.edu/asset_files/WhitePaper/2014_019_001_90981.pdf

Security

Webinar: Security Practitioner Perspective on DevOps for Building Secure Solutions

Software security often evokes negative feelings in developers because it is linked with challenges and uncertainty on rapid releases—especially for the Agile development process. The growing concept of DevOps can be applied to improve the security of applications. Applying DevOps principles can have a big impact on software resiliency and secure solutions. This webinar covers the perspectives of security practitioners on building secure software using the DevOps development process and modern security approach.
https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=474101

Video/Podcast: How Risk Management Fits into Agile and DevOps in Government

In this podcast, Eileen Wrubel, technical lead for the SEI's Agile-in-Government program, leads a roundtable discussion into how Agile, DevOps, and the Risk Management Framework can work together. The panelists include Tim Chick, Will Hayes, and Hasan Yasar.
https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=514190

Blog Post: Improving Security and Stability by Using DevOps Strategies

This blog post explores some basic DevOps practices that will improve application security while helping to maintain a stable operating environment.
https://insights.sei.cmu.edu/devops/2018/03/improving-security-and-stability-by-using-devops-strategies.html

Video/Podcast: 10 Types of Application Security Testing Tools and How to Use Them

In this podcast, Thomas Scanlon, a researcher in the SEI's CERT Division, discusses the different types of application security testing tools and provides guidance on how and when to use each tool.
https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=539820

Lean and Agile [Palmquist 2013]

Webinar: Practical Considerations in Adopting Lean and Agile in Government Settings

This webinar summarizes much of what the SEI has learned in its eight years of researching and facilitating the adoption of Lean and Agile methods in software-reliant systems in government. Suzanne Miller and Eileen Wrubel focus on how Lean and Agile principles can be interpreted for government settings and provide an overview of resources published by the SEI to support government organizations who are now, or are contemplating, adopting Agile or Lean methods for the software-reliant systems acquisitions and sustainment.
https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=502861

Podcast: Agile Software Teams and How They Engage with Systems Engineering on DoD Acquisition Programs

In this podcast, Acquisition researchers Eileen Wrubel and Suzanne Miller offer insights into how systems engineers and Agile software engineers can better collaborate when taking advantage of Agile as they deliver incremental mission capability.
http://resources.sei.cmu.edu/library/asset-view.cfm?assetid=427580

Podcast Series: Agile Adoption in the DoD

This is a series of podcasts by Suzanne Miller and Mary Ann Lapham that explores the application of the 12 Agile principles in the Department of Defense. Each podcast focuses on one of the Agile principles and includes google/apple podcasts, audio only, and transcript versions.

First Principle: "Our highest priority is to satisfy the customer through early and continuous delivery of valuable software."
http://resources.sei.cmu.edu/library/asset-view.cfm?assetid=294551

Second Principle: "Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage."
http://resources.sei.cmu.edu/library/asset-view.cfm?assetid=57578

Third Principle: "Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale."
http://resources.sei.cmu.edu/library/asset-view.cfm?assetid=58970

Fourth Principle: "Business people and developers must work together daily."
http://resources.sei.cmu.edu/library/asset-view.cfm?assetid=294306

Fifth Principle: “Build projects around motivated individuals.”
http://resources.sei.cmu.edu/library/asset-view.cfm?assetid=294212

Sixth Principle: “The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.”
http://resources.sei.cmu.edu/library/asset-view.cfm?assetid=91708

Seventh Principle: “Working software is the primary measure of progress.”
http://resources.sei.cmu.edu/library/asset-view.cfm?assetid=300353

Eighth Principle: “Agile processes promote sustainable development.”
http://resources.sei.cmu.edu/library/asset-view.cfm?assetid=424806

Ninth Principle: "Continuous attention to technical excellence and good design enhances Agile."
http://resources.sei.cmu.edu/library/asset-view.cfm?assetid=435068

Tenth Principle: “Simplicity—the art of maximizing the amount of work done—is essential.”
http://resources.sei.cmu.edu/library/asset-view.cfm?assetid=435352

Eleventh Principle: “The best architectures, requirements, and designs emerge from self-organizing teams.”
http://resources.sei.cmu.edu/library/asset-view.cfm?assetid=435441

Twelfth Principle: “At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.”
http://resources.sei.cmu.edu/library/asset-view.cfm?assetid=438821

Architecture

Webinar: Architectural Implications of DevOps

The Agile movement began as a reaction to frustration over the slow delivery of software, which often didn't sufficiently meet user needs. DevOps picks up what Agile started. Software development velocity has improved in many cases, yet we see deployment-related delays due to issues such as the inability to integrate continuously (or even frequently), resulting in late discovery of costly integration issues, challenges completing automated testing within an increment/build cycle, and uncertainty about whether a build is stable and secure enough for external release. To avoid problems such as these, SEI researcher Stephany Bellomo suggests it is critical for teams to make design decisions that align with their deployment goals such as reduced deployment cycle time and continuous delivery.
https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=298324

Webinar: What Makes a Good Software Architect (2019 Edition)?

The architect's role continues to evolve; in this webcast, we revisit the question in the context of today's roles and responsibilities. Researchers John Klein, Ipek Ozkaya, Felix Bachmann, and Suzanne Miller explore the challenges of working in an environment with rapidly evolving technology options, such as the serverless architecture style, and the role of the architect in Agile organizations using DevSecOps and Agile architecture practices to shorten iterations and deliver software faster.
https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=543614

Video: How do you integrate software architecture into Agile/DevOps environments?

SEI Researchers Andrew Kotov and John Klein respond to "How do you integrate software architecture into Agile/DevOps environments?"
https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=553008

Systems of Systems

Blog Post: Mission Thread Analysis Using End-to-End Data Flows - Part 1

The first post in a series by researcher Donald Firesmith on mission thread analysis using end-to-end data flows. This post identifies engineering challenges caused by the lack of an E2E mission thread analysis and provides an overview of an effective way of addressing these challenges: the E2E Mission thread Data flow Analysis (EMDA) method.

https://insights.sei.cmu.edu/sei_blog/2019/08/mission-thread-analysis-using-end-to-end-data-flows---part-1.html

Blog Post: Mission Thread Analysis Using End-to-End Data Flows - Part 2

This second blog post discusses the process used to create and verify the method's work products, the benefits of the method, the challenges that must be addressed while implementing the method, and lessons learned during the use of this method on a U.S. military program.
https://insights.sei.cmu.edu/sei_blog/2019/08/mission-thread-analysis-using-end-to-end-data-flows---part-2.html

References

URLs are valid as of the publication date of this document.

[Adler 1990]
Adler, P. & Shenhar, A. Adapting Your Technological Base. *Sloan Management Review*. Volume 32. Issue 1. October 1990. Pages 27-36. https://sloanreview.mit.edu/issue/fall-1990/#issue-loop

[Bass 2015]
Bass, Len; Weber, Ingo M.; & Zhu, Liming. *DevOps: A Software Architect's Perspective*. Addison-Wesley Professional. 2015.

[Bellomo 2014]
Bellomo, Stephany; Ernst, Neil; Nord, Robert; & Kazman, Rick. *Toward Design Decisions to Enable Deployability: Empirical Study of Three Projects Reaching for the Continuous Delivery Holy Grail*. Software Engineering Institute, Carnegie Mellon University. 2014. https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=298735

[Brownsword 2006]
Brownsword, Lisa; Fisher, David; Morris, Edwin; Smith, James; & Kirwan, Patrick. *System-of-Systems Navigator: An Approach for Managing System-of-Systems Interoperability*. CMU/SEI-2006-TN-019. Software Engineering Institute, Carnegie Mellon University. 2006. http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=7921

[CircleCI 2019]
Puppet, CircleCI, and Splunk. *2019 State of DevOps Report: Presented by Puppet, CircleCI, and Splunk.* 2019. https://www2.circleci.com/2019-state-of-devops-report.html

[Cockburn 2002]
Cockburn, Alistair. *Agile Software Development*. Addison-Wesley. 2002.

[Dahmann and Baldwin 2008]
Dahmann, J. S. & Baldwin, K. Understanding the Current State of U.S. Defense Systems of Systems and the Implications for Systems Engineering. *Proceedings of the2nd Annual IEEE Systems Conference.* 2008.

[DoD 2001]
Joint Publication 1-02 (JP 1-0). *Department of Defense Dictionary of Military and Associated Terms*. 2001 (updated in 2020). https://www.jcs.mil/Portals/36/Documents/Doctrine/pubs/dictionary.pdf

[DoD 2004]
U.S. Department of Defense. System of Systems Engineering. In *Defense Acquisition Guidebook*. Chapter 4. 2004.

[DoD 2006]
Department of Defense. Joint Publication 3-0 (JP 3-0). *Joint Operation*. 2006 (updated in 2018). https://www.jcs.mil/Portals/36/Documents/Doctrine/pubs/jp3_0ch1.pdf?ver=2018-11-27-160457-910

[DoD 2008]
Office of the Under Secretary of Defense (Acquisition, Technology and Logistics). *Systems Engineering Guide for Systems of Systems*. 2008.

[DTRA 2007]
Defense Threat Reduction Agency. Joint Improvised-Threat Defeat Organization (JIDO). *SecDevOps Concept of Operations*. Version 1.0. 2017.

[Fazal-Baqaie 2017]
Fazal-Baqaie, M.; Güldali, B.; & Oberthür, S. *Towards DevOps in Multi-Provider Projects*. 2nd Workshop on Continuous Software Engineering. February 2017. Hannover, Germany, 2017.

[Kim 2016]
Kim, Gene; Humble, Jez; Debois, Patrick; & Willis, John. *The DevOps Handbook*. IT Revolution Press. 2016.

[Klein and Reynolds 2019]
Klein, John & Reynolds, Doug. *Infrastructure as Code: Final Report*. Software Engineering Institute, Carnegie Mellon University. 2019. https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=539327

[Kruchten [2012]
Kruchten, Philippe; Nord. Robert L.; & Ozkaya, Ipek. Technical Debt: from Metaphor to Theory and Practice. *IEEE Software*. Volume 29. Number 6. Pages 8-21. 2012.

[Leonard-Barton 1988]
Leonard-Barton, Dorothy. Implementation as Mutual Adaptation of Technology and Organization. *Research Policy*. Volume 17. Number 5. Pages 251-267. 1988.

[Maier 1998]
Maier, Mark. Architecting Principles for Systems-of-Systems. *Systems Engineering*. Vol. 1. Number 4. Pages 267-284. 1998.

[Martinez 2018]
Martinez, Manuel Perez; Laszlo, Timea; Pataki, Norbert; Rotter, Csaba; & Szalai, Csaba. *Multi-vendor Deployment Integration for Future Mobile Networks*. SOFSEM 2018. January 2018. https://link.springer.com/content/pdf/10.1007%2F978-3-319-73117-9_25.pdf

[McCarthy 2015]
McCarthy, Matthew A.; Herger, Lorraine M.; Khan, Shakil M.; & Belgodere, Brian M. *Composable Devops*. Presented at IEEE International Conference on Services Computing. 2015.

[McQuade 2019]
McQuade, J. Michael; Murray, Richard M.; Louie, Gilman; Medin, Milo; Pahlka, Jennifer; & Stephens, Trae. *Software Is Never Done: Refactoring the Acquisition Code for Competitive Advantage*. Defense Innovation Board. 2019.

[Miller 2006]
Garcia-Miller, Suzanne & Turner, Richard. *CMMI® Survival Guide: Just Enough Process Improvement.* Addison-Wesley Professional. 2006.

[Miller 2014]
Miller, Suzanne. *The Readiness & Fit Analysis: Is Your Organization Ready for Agile?* Software Engineering Institute, Carnegie Mellon University. 2014. https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=90977

[Moore 2002]
Moore, G. *Crossing the Chasm: Marketing and Selling Disruptive Products to Mainstream Customers*. Harper Business Essentials. 2002.

[Morales 2018]
Morales, Jose A.; Yasar, Hasan; & Volkmann, Aaron. Implementing DevOps Practices in Highly Regulated Environments. In *Proceedings of International Workshop on Secure Software Engineering in DevOps and Agile Development.* SecSE 2018. https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=517148

[Palmquist 2013]
Palmquist, Steven; Lapham, Mary Ann; Garcia-Miller, Suzanne; Chick, Timothy; & Ozkaya, Ipek. *Parallel Worlds: Agile and Waterfall Differences and Similarities*. CMU/SEI-2013-TN-021. Software Engineering Institute, Carnegie Mellon University. 2013. http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=62901

[Patterson 1982]
Patterson, Robert W. & Conner, Daryl. Building Commitment to Organizational Change. *Training & Development Journal*. Volume 36. Number 4. Pages 18-30. 1982.

[Rogers 2003]
Rogers, E. *Diffusion of Innovation*. 5th ed. Simon & Schuster. 2003.

[Waits 2015]
Waits, Todd & Volkmann, Aaron. Webinar: *Culture Shock: Unlocking DevOps with Collaboration and Communication*. Software Engineering Institute, Carnegie Mellon University. 2015. https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=442589.

[Weinberg 1997]
Weinberg, Gerald. *Quality Software Management Volume 4: Anticipating Change*. Dorset House Publishing. 1997.

[Wrubel 2014]
Wrubel, Eileen; Miller, Suzanne; Lapham, Mary Ann; & Chick, Timothy. *Agile Software Teams: How They Engage with Systems Engineering on DoD Acquisition Programs*. CMU/SEI-2014-TN-013. Software Engineering Institute, Carnegie Mellon University. 2014. http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=295943

[Yasar 2018]
Yasar, Hasan & Lackey, Zane. *Security Practitioner Perspective on DevOps for Building Secure Solutions*. Webinar. https://youtu.be/U8972_RR9p0.

[Zmud 1992]
Zmud, R. & Apple, L. E. Measuring Technology Incorporation/Infusion. *Journal of Product Innovation Management*. Volume 9. Number 2. Pages 148-155. 1992.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
(Leave Blank)	April 2020	Final

4. TITLE AND SUBTITLE	5. FUNDING NUMBERS
Guide to Implementing DevSecOps for a System of Systems in Highly Regulated Environments	FA8702-15-D-0002

6. AUTHOR(S)

Jose Morales, Richard Turner, Suzanne Miller, Peter Capell, Patrick Place, David James Shepard

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)	8. PERFORMING ORGANIZATION REPORT NUMBER
Software Engineering Institute Carnegie Mellon University Pittsburgh, PA 15213	CMU/SEI-2020-TR-002

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSORING/MONITORING AGENCY REPORT NUMBER
SEI Administrative Agent AFLCMC/AZS 5 Eglin Street Hanscom AFB, MA 01731-2100	n/a

11. SUPPLEMENTARY NOTES

12A DISTRIBUTION/AVAILABILITY STATEMENT	12B DISTRIBUTION CODE
Unclassified/Unlimited, DTIC, NTIS	

13. ABSTRACT (MAXIMUM 200 WORDS)

DevSecOps (DSO) is an approach that integrates development (Dev), security (Sec), and delivery/operations (Ops) of software systems to reduce the time from need to capability and provide continuous integration and continuous delivery (CI/CD) with high software quality. The rapid acceptance and demonstrated effectiveness of DSO in software system development have led to proposals for its adoption in more complex projects. This document provides guidance to projects interested in implementing DSO in defense or other highly regulated environments, including those involving systems of systems.

The report provides rationale for adopting DSO and the dimensions of change required for that adoption. It introduces DSO, its principles, operations, and expected benefits. It describes objectives and activities needed to implement the DSO ecosystem, including preparation, establishment, and management. Preparation is necessary to create achievable goals and expectations and to establish feasible increments for building the ecosystem. Establishing the ecosystem includes evolving the culture, automation, processes, and system architecture from their initial state toward an initial capability. Managing the ecosystem includes measuring and monitoring both the health of the ecosystem and the performance of the organization. Additional information on the conceptual foundations of the DSO approach is also provided.

14. SUBJECT TERMS	15. NUMBER OF PAGES
DevOps, DevSec,Ops, continuous delivery, system of systems, ecosystem	111

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL