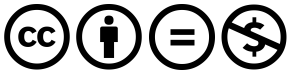
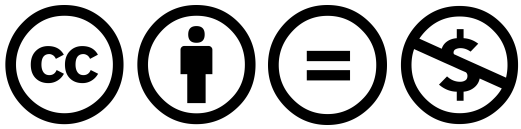
Implementing DevOps Learning Outcomes



ICAgile®
International Consortium for Agile

LICENSING INFORMATION

The work in this document was facilitated by the International Consortium for Agile (ICAgile) and done by the contribution of various Agile Experts and Practitioners. These Learning Outcomes are intended to help the growing Agile community worldwide.



SPECIAL THANKS

ICAgile would like to thank the contributors to the Implementing DevOps Learning Outcomes:
Bryon Brewer • Colin Garlick • Dominica DeGrandis • Gene Gotimer • Tim Guay • Chris Knotts

CONTENTS

HOW TO READ THIS DOCUMENT

This document outlines the Learning Outcomes that must be addressed by accredited training organizations intending to offer ICAgile's Implementing DevOps certification.

Each LO follows a particular pattern, described below.

0.0.0. Learning Outcome Name

Additional Context, describing why this Learning Outcome is important or what it is intended to impart.

The Learning Outcome purpose, further describing what is expected to be imparted on the learner (e.g. a key point, framework, model, approach, technique, or skill).

LEARNING OUTCOMES

1. PLANNING THE PIPELINE

1.1. DEFINING THE OUTCOMES (THE "WHY" BEHIND IMPLEMENTING DEVOPS)

1.1.1. The Value Stream

DevOps must encompass the entire value stream.

Illustrate the complete value stream, from the greenfield or brownfield request to the day-to-day support of the production application and its users. Identify non-value add activities and excessive bureaucracy that can be reduced or eliminated with value stream mapping. Also recognize that the DevOps pipeline is only part of the value stream, and that the whole value stream must be understood and optimized.

1.1.2. Dangers of Over-Controlling Change

Make it easy and fast to get value into the hands of the user.

List the dangers of unnecessary constraints around change and deployment. Discuss the advantages of practices such as peer review of changes over change control boards, and pair programming over traditional code reviews.

1.1.3. Shared Responsibility

Reduce outages and enable faster recovery from any outages.

Explain the importance of shared development and operations ownership as foundational to implementing DevOps. Explore good practices for sharing that responsibility and reducing the separation of duties.

1.2. CONTINUOUS INTEGRATION (CI) AS A PREREQUISITE TO DEVOPS

1.2.1. Developers Have Full Control Over Their Environments

Remove unnecessary barriers to developers doing their job.

Review the importance of developers having full control over their environments, and the dangers of imposing bureaucratic processes for gaining access to tools, services and permission to deploy. Discuss how automated self-service access to services, tools and deployments supports this.

1.2.2. Same Tools in Pre-Production and Production

Reduce the risks related to promoting to production.

Illustrate the danger of having differences in tooling and configuration between pre-production and production environments. Show how this naturally leads to expanding the definition of done to running in production environments, or as close to production as possible.

1.2.3. Automated Build Systems

Make the build process repeatable and reliable.

Discuss the capabilities of build tools, including running tests, and show how this can increase confidence in the system being built.

1.2.4. Continuous Integration

Verify and validate the system continually.

Show how regular builds can result in faster development and more reliable systems.

1.2.5. Stop the Line

Don't live with a broken build.

Show how not fixing a failing continuous integration job leads to more operational issues. Identify the culture and practices that are needed to avoid this.

1.3. DESIGN AND ARCHITECT APPLICATIONS FOR DEVOPS

1.3.1. Enterprise and Software Architecture

Know the big picture.

Show how enterprise and domain architecture places constraints on the pipeline. Describe how the solution and software architecture determines what is possible regarding both automation and deployment. Describe some architectural patterns such as monolithic applications, client-server and microservices, and how they can support and/or constrain DevOps implementation.

1.3.2. Qualitative Requirements

Identify which non-functional assessments need to be incorporated in the pipeline.

Review how software architecture supports achieving the non-functional requirements, and identify how these will be verified, whether manually or with automation. Determine what technology is required to support this verification (e.g., what environments and databases are needed).

1.3.3. Security

Design for security from day one.

Explain how security needs to be designed in at the beginning, rather than tested at the end. Plan how this will be verified in the pipeline.

1.4. IDENTIFYING THE STAGES OF THE PIPELINE

1.4.1. Co-Create with Development and Operations

Both Development and Operations are responsible for the flow, and both sets of concerns must be addressed.

Explain the importance of equal involvement of both Development and Operations in the design of the pipeline, and that both groups' concerns carry equal weight. Explore how to ensure that both groups have equal say, and how to encourage cooperation and address conflicting requirements.

1.4.2. Structuring and Scheduling Tests

Automated testing fosters trust in the pipeline.

Categorize the different types of tests that can be performed, and identify which can be automated and which will be manual.

1.4.3. Document the Pipeline

The system must undergo various assessments before being allowed to be deployed.

Produce the design for, and order the different stages of, assessment that the system being built will need to go through in order to build sufficient confidence to allow it to be deployed to production. Identify which steps are manual and which are automated, concurrent steps and manual triggers.

2. BUILDING THE PIPELINE

2.1. PROVISIONING ENVIRONMENTS

2.1.1. Provisioning Environments

Environments should be managed as one of many, not individually crafted.

Discuss how manually configuring environments results in unreliable systems and an inability to scale. Show that a hands-off approach is needed to achieve reliability and scalability.

2.1.2. Infrastructure as Code

Automated provisioning and configuration enables a self-service model.

Review the capabilities of provisioning and configuration tools, and show how this supports a self-service model for Development.

2.2. DEPLOYING SYSTEMS

2.2.1. Automated Deployment

Deployments should be repeatable, reliable and easy to troubleshoot.

Identify how automated deployments are integral to building confidence in the deployment pipeline. Illustrate how manual deployments remove the ability to deploy frequently.

2.2.2. Release Patterns

Teams should have a strategy for releasing as well as for rolling forward or back.

Review various patterns for releasing systems, such as blue-green deployment, discussing the benefits and challenges of each. Explain the impact of each on the database, including schema migrations and transactions in-flight.

2.2.3. Orchestration

Managing the complete workflow, not just the individual parts.

Show how coordinating and automating the overall process is just as important as automating the individual steps. Describe how exception conditions, including rollbacks, can and should be included in the process.

2.3. THE PIPELINE AS CODE

2.3.1. Code the Pipeline

Store the pipeline in the project's repository.

Develop the pipeline as code, and explain why it should be committed to the project's repository.

2.3.2. Feed the Machine

The pipeline should run after every commit.

Combine the pipeline with CI, and demonstrate how practices such as stop-the-line support achieving the goals of the pipeline.

3. MONITORING THE PIPELINE

3.1. CYCLE TIME

3.1.1. Measure Cycle Time

An understanding of cycle time and how to measure it is key to identifying waste.

Explain how they are normalized in an organization. Explain the difference between cycle time and lead time. Explain the concepts of Value-Added, Value-Enabling and Non-Value Added activities, as well as the concept of 'Inventory.' List the 7 deadly wastes and show how they are normalized in an organization.

3.1.2. Theory of Constraints

The Theory of Constraints is a valuable tool for identifying and eliminating bottlenecks.

Explain the Theory of Constraints and the Five Steps to Exploiting a Constraint. Be aware of the dangers of optimizing the whole. Show the use of Cycle Efficiency and Cumulative Flow Diagrams to monitor the pipeline as well identify bottlenecks. Learn how to drive the discussions on eliminating a bottleneck.

3.2. LOG MANAGEMENT

3.2.1. Log Types

Different log types and log levels are revelant to different phases of the pipeline.

Show the various log types and explain which log type and level are used for each phase of the pipeline.

3.2.2. Organizing and Communicating Log Data

Organize log data in ways that facilitate insights and communication.

Explain the options for collecting, indexing and displaying log data. Show which tools can be used to facilitate these activities.

3.2.3. Dealing with Sensitive Information and PII

Attention must be paid and special strategies applied when logging sensitive and Personally Identifiable Information (PII).

Explain the legal and security issues with logging sensitive and PII. Show the strategies available for protecting this data, such as encryption or redaction. Address the compliance and auditing issues and practices.

3.3. TELEMETRY

3.3.1. Types of Telemetry

Different types of telemetry and instrumentation apply to different phases of the pipeline.

Show what types of telemetry are available for each phase of the pipeline as well as how to implement them. Show how to integrate telemetry with logging.

3.3.2. Self-Service Telemetry

Allow developers and operations personnel to create their own telemetry.

Explain the advantages of allowing developers and operations personnel to instrument their code and processes, and have self-service access to telemetry. Show how to create production metrics as part of daily work.

3.3.3. Diagnostics with Telemetry

Use telemetry output to diagnose problems and improve the pipeline.

Show how to detect abnormalities and diagnose the underlying cause using telemetry output. Show how to set control bands and threshholds to avoid false positives.

3.3.4. Security Monitoring

Monitor the pipeline for security abnormalities.

Show the basics on monitoring each phase of the pipeline for security abnormalities. Explain and detect the types of security abnormalities to monitor.

3.4. PRODUCTION SUPPORT

3.4.1. Shared Responsibility

Development and Operations share responsibility for production support.

Show the importance of shared development and operations as foundational to implementing DevOps. Explore best practices for sharing that responsibility.

3.4.2. Release Patterns

Multiple release patterns exist and can be used towards the goal of achieving zero-downtime deployments.

Introduce the different release patterns (e.g., environment-based release patterns, application-based release patterns, Canary Release Pattern, Immune System Release Pattern). Discuss methods to decouple deployments from releases and the role of dark launches. Distinguish the pros and cons of different release patterns. Learn how to implement the various release patterns, how to address database changes, and how automated testing makes fixing forward safe.

3.4.3. Scaling and Virtualization

It is important to consider scaling and virtualization in context of deployment and production support.

Explain the deployment process for various forms of virtualization, including dynamic infrastructure. Benefits of SOA and containerization for supporting scaling and virtualization. Monitoring environmental demands. Explore the pros and cons of the different approaches to provisioning. Monitoring environmental demands. Exploiting scaling and virtualization to adapt to operational demands.

3.4.4. Managing Data

Managing data and databases in a DevOps environment brings unique challenges.

Show the challenges of managing relational, free-form and hybrid Big Data in a DevOps environment. Strategies for optimizing databases for DevOps, such as creating micro-databases that support a single service, feature flags and feature routers. Addressing security and privacy challenges.

3.5. HEARING THE VOICE OF THE CUSTOMER

3.5.1. Gathering Customer Data

Gathering relevant customer data is essential to continuous improvement.

Show various proactive and reactive methods for gathering customer data as well as their pros and cons. Explain the importance of engaging the customer early and continuously.

3.5.2. Analyzing and Communicating the Voice of the Customer

Techniques for understanding and communicating the Voice of the Customer.

Show some techniques for analyzing and communicating customer data, such as Customer Journey, Kano Models, CTQ charts, POV MadLib and Empathy Mapping. Know the pros and cons of each technique.

4. MATURING THE PIPELINE

4.1. DEPLOYMENT ARCHITECTURES

4.1.1. Alternative Architectures

Use services and non-traditional systems to compose environments.

Describe how traditional architectures are often unable to provide the level of confidence required for continuous deployment, and explain how alternatives (e.g., containers, microservices, cloud services) can address these concerns.

4.1.2. Evolutionary and Sacrificial Architecture

The current architecture is not the target, but a step on the journey.

Explain that the architecture will change and evolve throughout the life of any system, and that there may come a point where it has to be sacrificed in favor of a new and improved architecture. Describe how this may take many steps to achieve, requiring a roadmap for bringing about this change over a long period of time.

4.2. CONTINUOUS DEPLOYMENT

4.2.1. Revise the Pipeline

Ensure the pipeline is still relevant.

Review the pipeline in terms of the current goals, comparing its performance and results with the desired outcomes. Revise it where there is a discrepancy.

4.2.2. Ensure Compliance

Satisfy legal, regulatory, audit and other compliance needs.

Identify stakeholders in the development and deployment process, and verify that any changes to the pipeline comply with their requirements.

4.2.3. Confirm Governance

Improve the process as well as the product.

Analyze the pipeline, looking for stages that can be removed or have their impact minimized without jeopardizing the confidence that the stakeholders have in the process.

4.2.4. Engineer the Pipeline

Treat the pipeline as production code.

Develop processes to allow the pipeline to be modified, upgraded and maintained without disrupting delivery. Explain why this requires that changes to the pipeline be tested, consistency be maintained across projects and changes be traceable.