Continuous Delivery

Streamline the Path from Idea to Value

Version 7.1

Workshop Overview

Module 0

Course Agenda

- Module 01: What is Continuous Delivery?
- Module 02: Prerequisites to Continuous Delivery
- Module 03: Implementing Continuous Delivery
- Module 04: Continuous Integration
- Module 05: CD and Your Environments
- Module 06: The Deployment Pipeline
- Module 07: Design of a Strong Foundation

Agenda

- Module 08: Automated Testing
- Module 09: Developing on Trunk
- Module 10: Telemetry, Logging & Monitoring
- Module 11: Security in CI/CD
- Module 12: Code Stability
- Module 13: The Planning Paradigm
- Module 14: Setting Goals and Getting Started

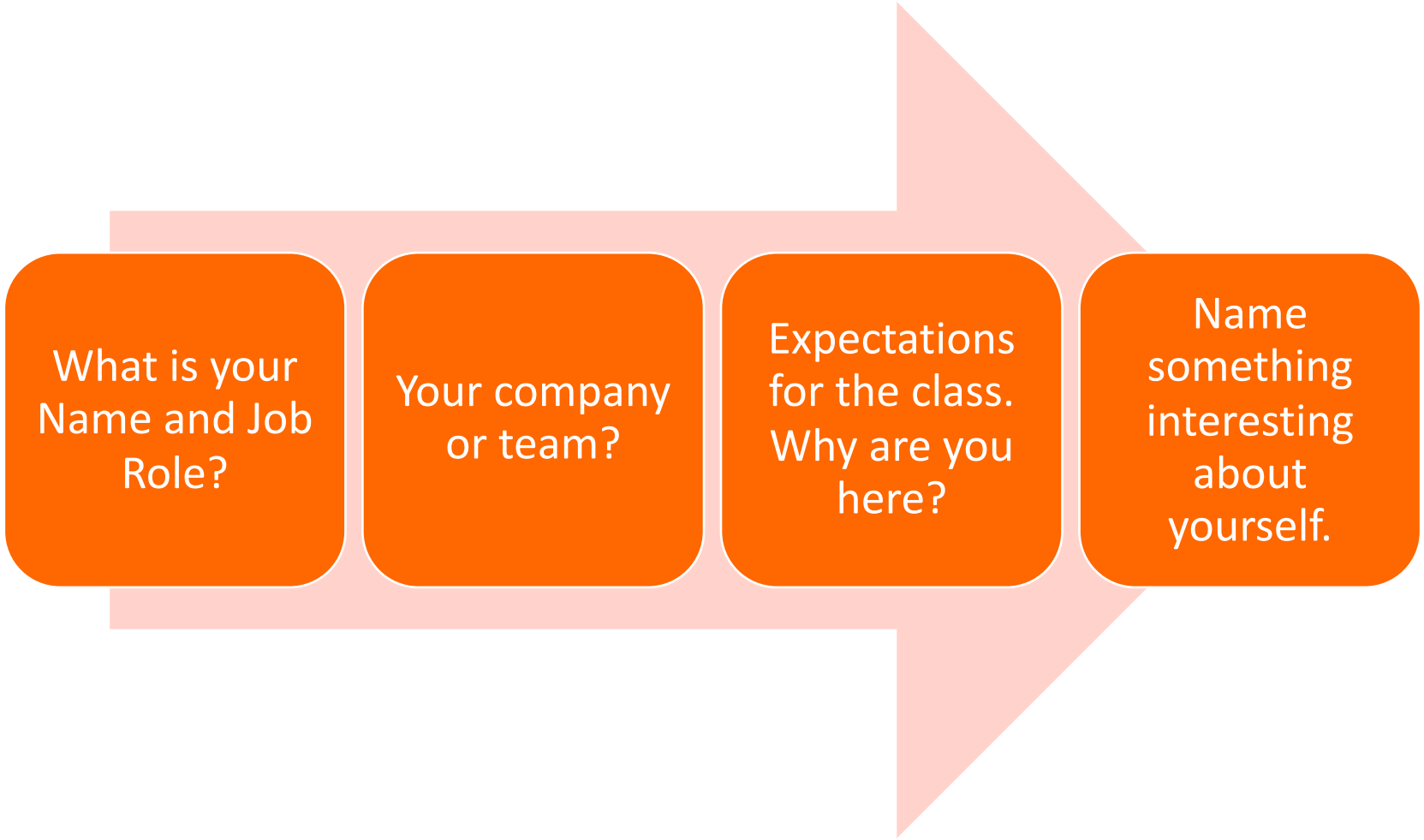
Who is this workshop for?

- Developers and their leaders
- Teams responsible for:
  - Code
  - Infrastructure
  - QA, test and security
  - Operations
- Anyone involved in creating or deploying software solutions – and the outcomes

Related Recourses

- ***Continuous Delivery: Reliable Software Releases Through Build, Test, and Deployment Automation*** – by Jez Humble and David Farley
- ***Leading the Transformation: Applying Agile and DevOps Principles at Scale*** – by Gary Gruver and Tommy Mouser
- ***The DevOps Handbook*** – by Gene Kim, Jez Humble, Patrick DeBois, John Willis, and John Allspaw
- ***Toyota Kata*** – by Mike Rother

Introductions



IC Agile ICP-IDO Certification

ICAgile Certification Video

What is Continuous Delivery?

Module 1

Module 1: What is Continuous Delivery?

- *Introduction to Continuous Delivery*
- *How does your team deliver software?*
- *Where does Continuous Delivery fit in the DevOps Landscape?*
- *What are the benefits of using Continuous Delivery?*
- *Anti-patterns*

Continuous Delivery

"Continuous Delivery is a software development discipline where you build software in such a way that the software can be released to production **at any time."**

– *Martin Fowler*

Introduction to Continuous Delivery

- Ready to deploy at any time
- Able to deploy any version
- Able to deploy to any supported platform

- *It doesn't mean you have to deploy continuously – it means you perform your engineering so you can if you want to*

Able to Deploy at any time

Deployment should be fast

- Less dependencies
- Less Hand-offs
- No long lived branches
- Maintain error free trunk
- Well tested
  - Automated unit test
  - Automated integration test

Able to Deploy any Version

- Use low risk deployment architectures
  - Blue green deployment
  - Canary releases
- Store Deployment package in Version Control
- Store all required artifacts
  - Deployment Package
  - Deployment Script
  - Anything required to deploy and run the solution

Deploy to any Platform

- Use of Configuration Management
- Platform management
  - Use of containers
  - Store supported OS .iso in Package Repository
- Store all required artifacts for supported Platforms
  - OS, Patches, Supporting Software and Utilities

Continuous Delivery – "Classic" Core Principles

- "10+ Deploys per day"
- If it hurts do it more!
- Automation becomes a requirement
- Mean time to change decreases
- Mean time to recovery decreases
- Confidence increases and risk is reduced

You are succeeding with continuous delivery when:

1. Your software is deployable throughout its lifecycle
2. Your team prioritizes keeping the software deployable over working on new features
3. Anybody can get fast, automated feedback on the production readiness of their systems whenever somebody makes a change to them
4. You can perform push-button deployments of any version of the software to any environment on demand.

Anti-Patterns

Some of the most common obstacles to effective CD practices are:

- **Manual Deployment**
- **Manual Testing**
- **Organizational Silos**
- **Manual Configuration Management**

Value Stream Mapping

A **Lean method** for analyzing a Value Stream

(or a system)

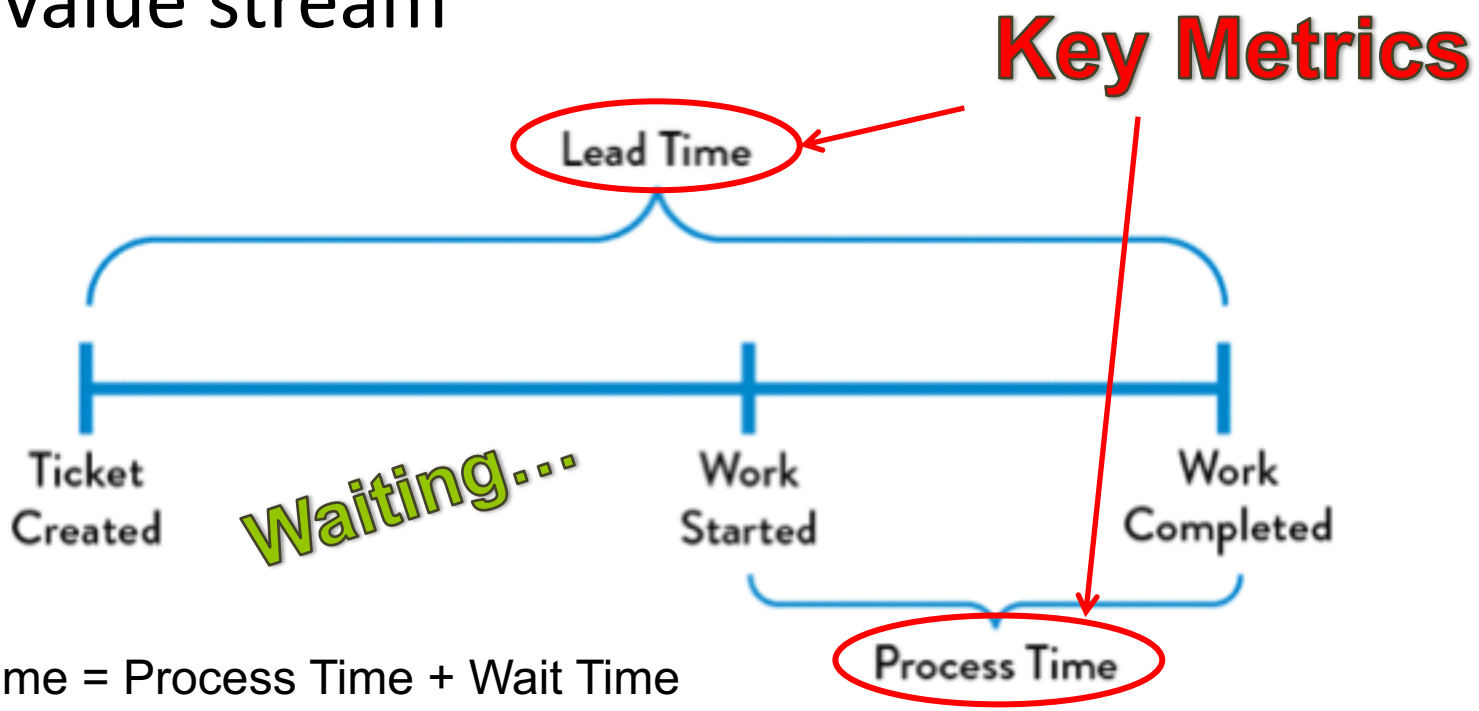
Value Stream: A series of steps that take a product or service from its beginning through to the customer.

Include
Waiting and **Process** time

Lead Time

The time required to complete a task:

Task can be entire value stream or one step in the value stream



Lead Time = Process Time + Wait Time

Measuring Rework

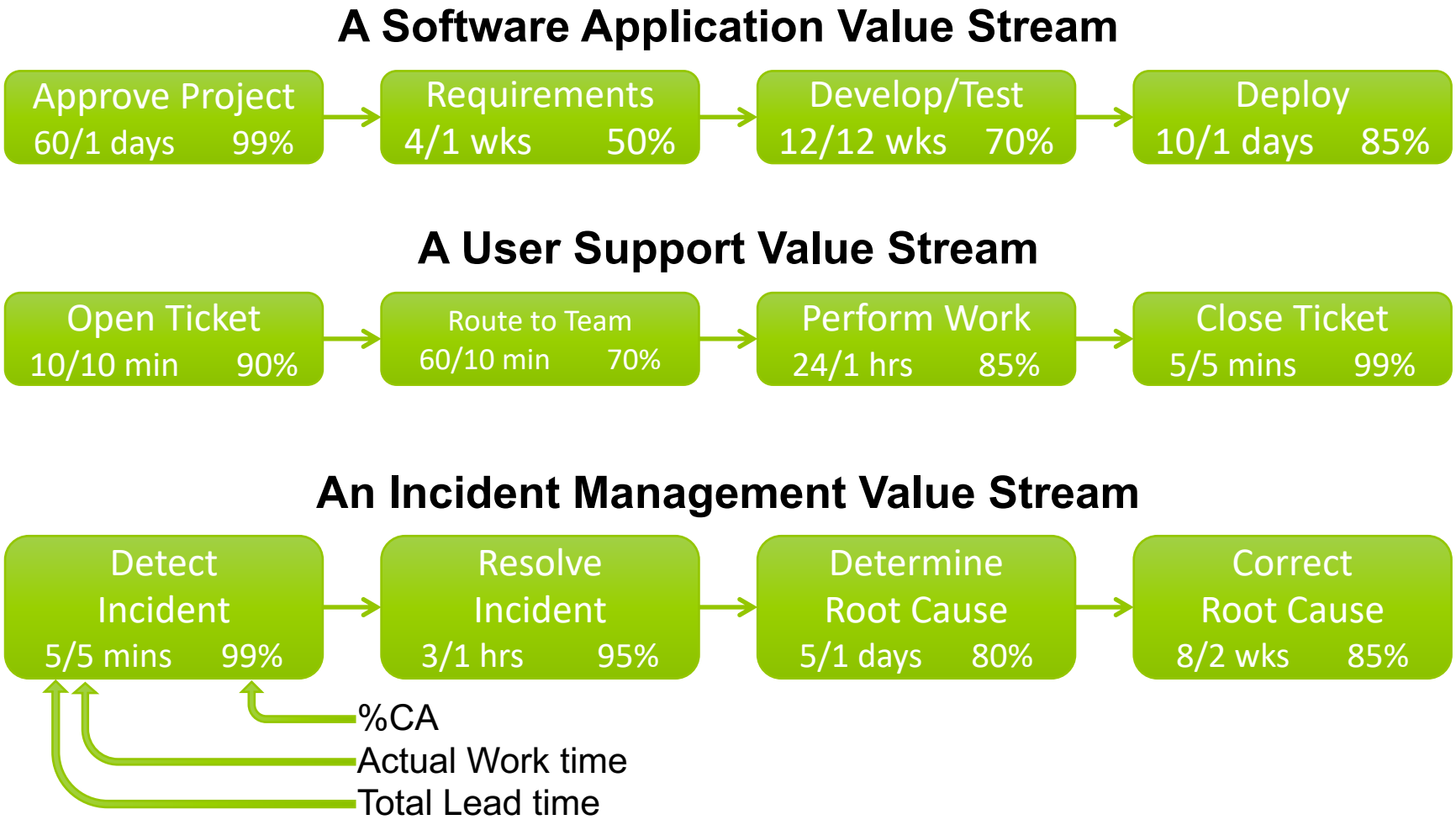
3rd Key Metric

%CA = For each Value Stream Step Percentage of output that is Complete and Accurate

- **Usable downstream as is**
- **Requiring No rework**

$$\%CA = \frac{(\text{Total Step Output}) - (\text{Step Output Requiring Rework})}{(\text{Total Step Output})} \times 100\%$$

Value Stream Examples with Metrics

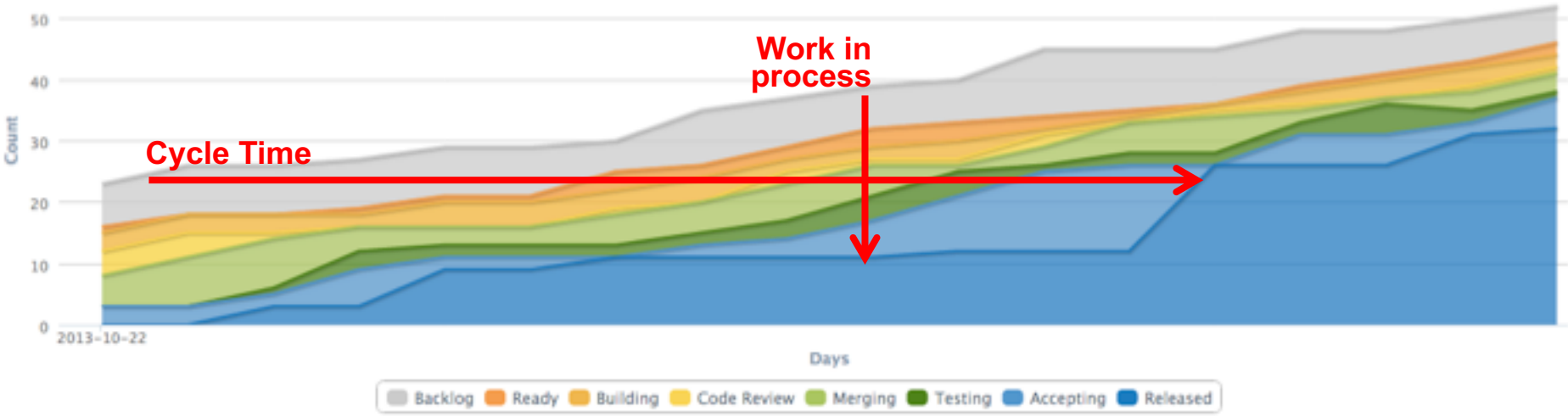


Exercise
Map Your Current Deployment Process

- **Create a Value Stream Map for your deployment process as it currently is**
  - **Draw a diagram of the Value Stream**
  - **For each step of the Value Stream, make your best estimate of the three key metrics**
    - Total Lead time
    - Actual Work time
    - %CA (Percent Complete and Accurate)

Cumulative Flow Diagram

Like a Burn-Up chart, but not just totals



Bottleneck Activities:

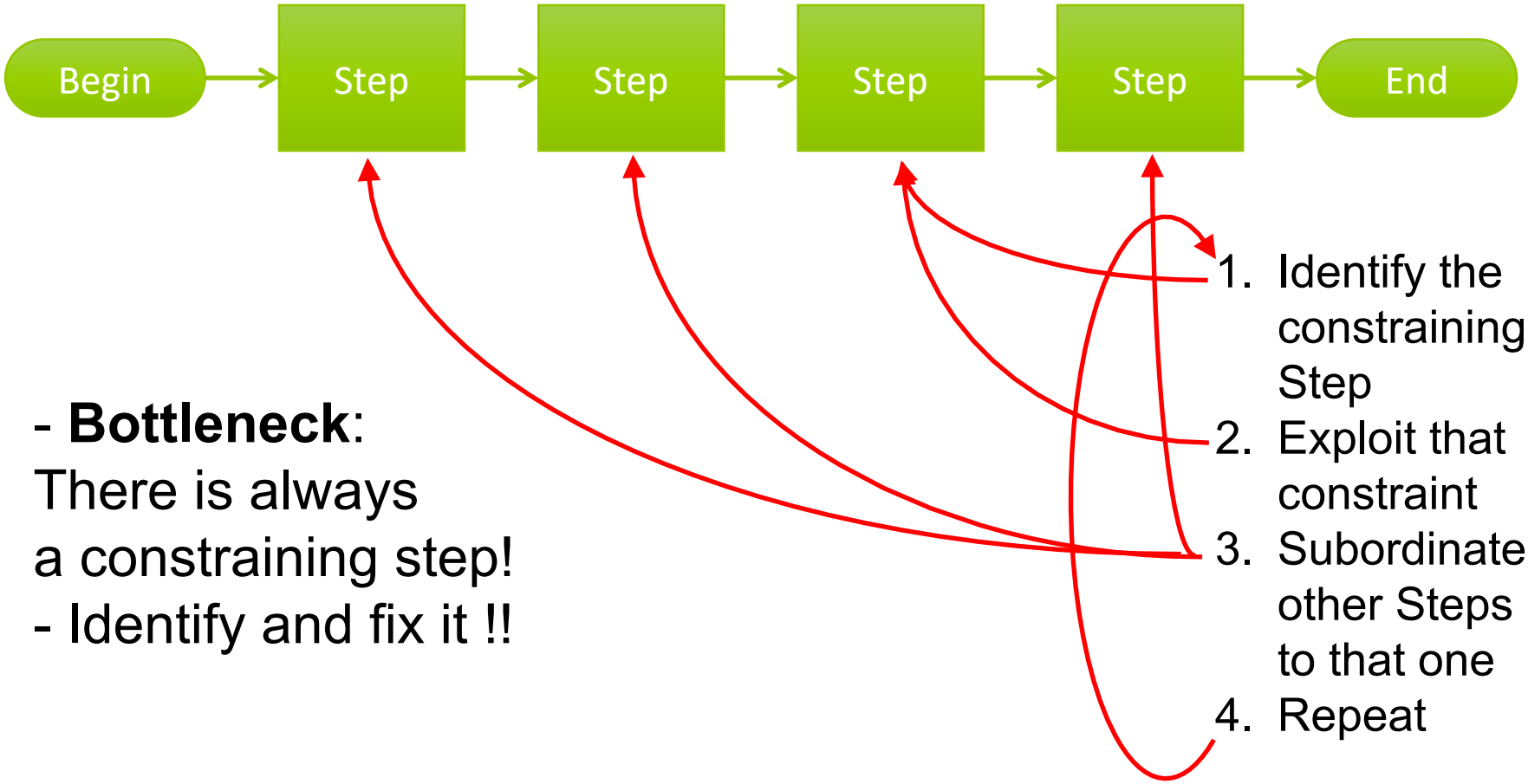
- Excessive WIP
- Excessive Cycle Times (e.g. slow moving items)
- Uneven Growth

Where are the probable bottlenecks in this example?

Identifying the constraining steps

- Look at the cycle and WIP graphs
- Identify the constraining steps
  - Steps that takes lot of time
  - Steps that are repeated but can be automated
  - Steps that pass on errors downstream

Theory of Constraints



- **Bottleneck**:
There is always
a constraining step!
- Identify and fix it !!

Exercise: Identify Deployment Bottlenecks

- Based on your Value Stream Analysis:

  Identify the primary Constraints

  in your Deployment process

  - Manual
  - Long Running
  - Error Prone
  - Executed Frequently

Prerequisites to a Continuous Delivery Practice

Module 02

Prerequisites to a Continuous Delivery Practice

- Commitment from leadership
- Commitment to an Agile mindset, principles, and practices
- Commitment to automation
- Commitment to empowering fast deployment and fast feedback on every functional role, from development to security

Let’s have a conversation about Agility

- An iterative approach to software development where detailed planning is only done for the next 1-30 days
- Shorter time scales?
- Evaluating process and progress at the end of a time block or sprint
- Minor improvements / corrections made during sprints – continuous improvement. This means experiments!

Engaging the Customer

- Agile Value: Customer Collaboration
- Agile Principle: Satisfy the Customer

Engaging the customer:

- During development
- Before deployment
- After deployment

Capacity for Change

At the heart of an agile practice, and certainly a CI/CD practice, is the ability to deploy change rapidly and productively.

- **Technical Solutions**
  - Represent a smaller portion of the effort (most Orgs focus here)
- **Enterprise Change Management**
  - Priority at the high level
  - Drive change management
- **Cultural environment**
  - Collaborative workspaces
  - Having clear definition of done
  - Constructive feedback – Blameless Postmortems

Do you have a plan for always deployable code?

- Agile Principles at Scale
- Developers have a lot of input in defining team-level processes
- Is security involved?
- What is your monitoring/telemetry strategy?

Continuous Improvement requires everybody to have ownership of success

Lab 1
Explore a TFS CI Configuration

- Exercise 1: Explore TFS Key Features
- Exercise 2: Explore Azure Key Features

Lab guide: Page 2 to 9

Exercise 1: Explore CI Settings in TFS

- Login to TFS Account
- View the Team Project Configuration
- Explore Team Project menus

Exercise 2: Explore Azure DevOps key Features

- View Sample Project Source Control Settings
- View Check-in Settings
- View Build Configuration

Azure DevOps Documentation

https://docs.microsoft.com/en-us/azure/devops/pipelines/?toc=%2Fazure%2Fdevops%2Fpipelines%2Ftoc.json&bc=%2Fazure%2Fdevops%2Fboards%2Fpipelines%2Fbreadcrumb%2Ftoc.json&view=azure-devops

Implementing Continuous Delivery:

Core Expectations

Core Expectations

- Continuous Integration
- Scripted Environments
  - Infrastructure as Code
- Scripted Deployments
- Evolutionary Database Design
- Deployment Pipeline
- Orchestration
- Post Deployment Validation
- Embrace the Cultural Shift
- Align IT and Business Objectives

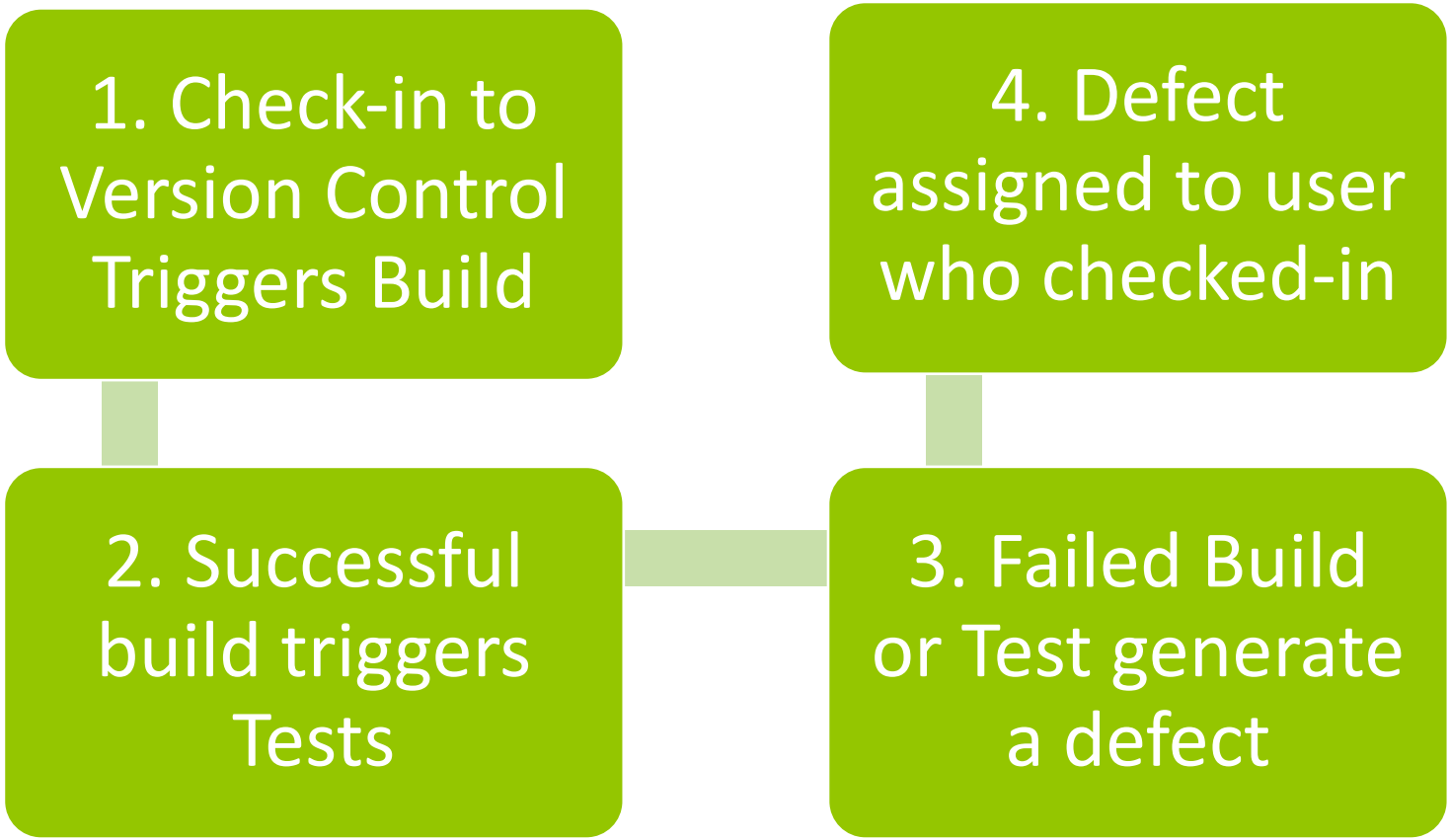
Continuous Integration

Module 03

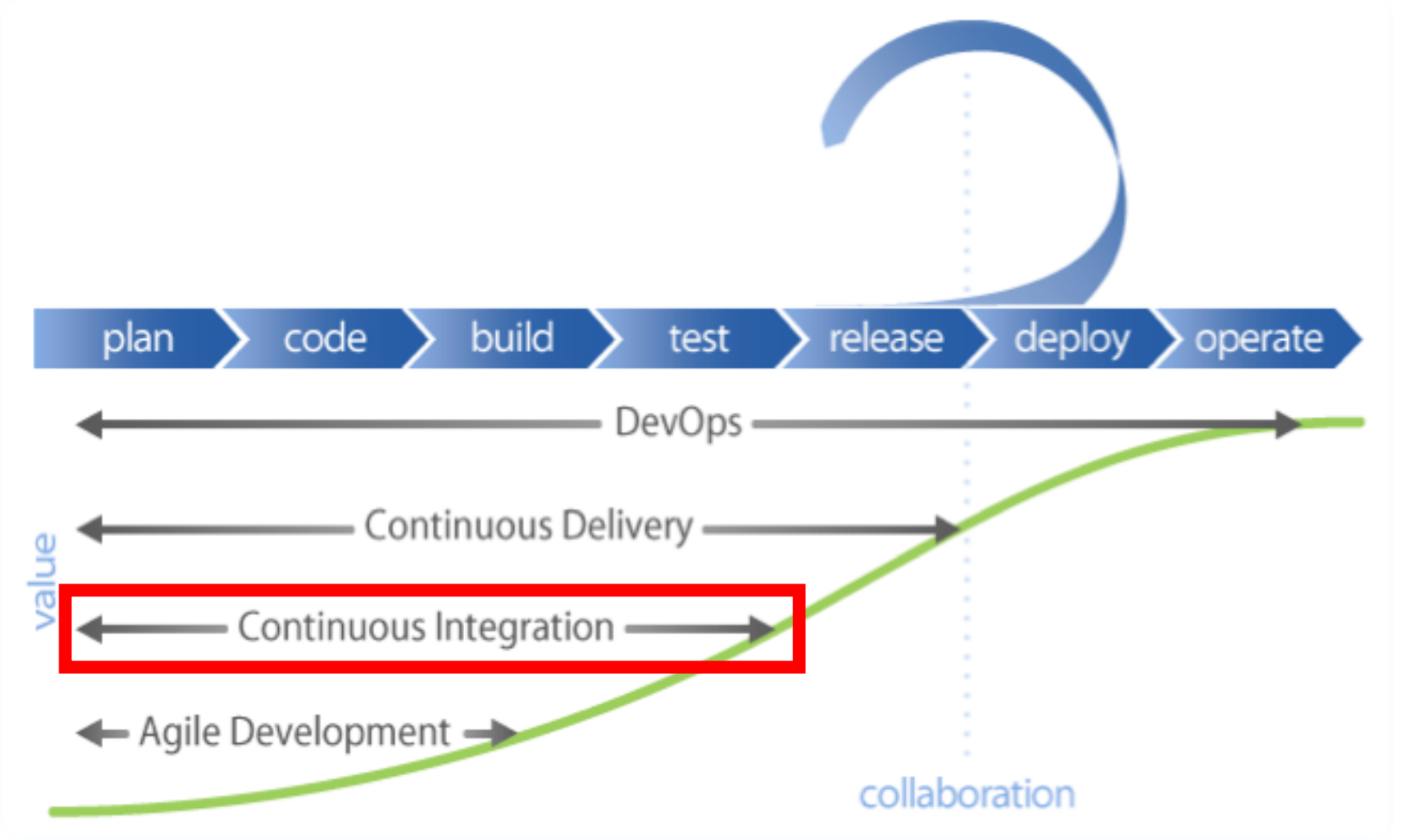
Module 3: Continuous Integration

- What is Continuous Integration?
- Where does Continuous Integration fit in the DevOps Landscape
- How does Continuous Integration work?
- Continuous Integration Essentials
- Common Continuous Integration Practices
- Benefits of Continuous Integration
- Continuous Integration to Continuous Delivery

What is Continuous Integration?



Where CI fit into the DevOps Landscape?



Continuous Integration Essentials

- Automated Build
- Automated Tests
- Automated platform creation
- Deployment Package Creation
- Deployment Package Storage

Benefits of Continuous Integration

- Feedback early and often
  - Integrated Build Results
  - Test Results
  - Focus on smaller task items and change set sizes
- If you are going to fail, fail early
- Enables automated testing
- A necessary milestone on the path to Continuous Delivery and DevOps

Continuous Integration Vs Delivery Vs Deployment

- **Integration**: Integrate code into a main branch early and often, instead of building out features in isolation and integrating them at the end of a development cycle
- **Delivery**: Automate the software delivery process so that teams can easily and confidently deploy their code to production at any time; by ensuring that the codebase is always in a deployable state
- **Deployment:** Automatically deploys each build that passes the full test cycle; instead of waiting for a human gatekeeper to decide what and when to deploy to production

For the next lab (Code Coverage)

- Metric that can help to understand how much of the source is tested. Code coverage tools will use one or more criteria to determine how the code was exercised or not during the execution of the test suite.
  - **Function coverage:** how many of the functions defined have been called
  - **Statement coverage:** how many of the statements have been executed
  - **Branches coverage**: how many of the branches of the control structures (if statements for instance) have been executed
  - **Condition coverage:** how many of the boolean sub-expressions have been tested for a true and a false value
  - **Line coverage:** how many of lines of source code have been tested

For the next lab (Triggers)

- **CI Trigger:** If a CI build fails due to bad code being committed, the code is still committed and reside in the source control
- **Gated check-in Trigger**: TFS creates a *shelveset* containing the code that's being validated, then runs a build of that code. Only if that code builds successfully and all configured unit tests pass does the code actually get committed
  - Benefits:- It prevents checking in bad code
  - Drawbacks:- Makes the process slower :- Others have to wait, until the code passes gated check-in trigger

Lab 2
Configure Team Services for Continuous Integration

- Exercise 1 Create a Build Pipeline
- Exercise 2 Enable Continuous Integration
- Exercise 3 Configure Work Item Creation

Lab guide: Page 10 to 19

Exercise 1: Create Build Pipeline

- Create Build Pipeline
- Modify Test Settings

Exercise 2 Enable Continuous Integration

- Enable CI Build Trigger
- Enable Gated Check-in option

Exercise 3 Configure Work Item Creation

- Enable Bug Task on Build Failure
- Assign Bug to Requester

Exercise
Identify Deployment Improvements

- Update your Value Stream Map based to reflect using Continuous Integration
  - Estimate new Lead Times and %CAs
- Identify how CI improves your Deployment process

Continuous Delivery and Your Environments

Module 04

Continuous Delivery and Your Environments

- Infrastructure Management
- Communicating with the Operations Team
- Configuration Management
- Infrastructure in the Cloud
- Infrastructure Maintenance

Infrastructure Management

Infrastructure Challenges

- Cycle Time – Can take too long and therefore costly
- Manually versioning environments is costly

Solution: Managing Infrastructure as Code

- Manage infrastructure as code for reliable, repeatable configurations
- Ensure speedy and reliable deployments with a proven DevOps platform
- Manage all the complexity of the cloud by treating infrastructure as code.
- Enables continuous delivery practices like version control & continuous integration.

Communicating with Operations Team

- **Lack of communication can lead to resistance to change**

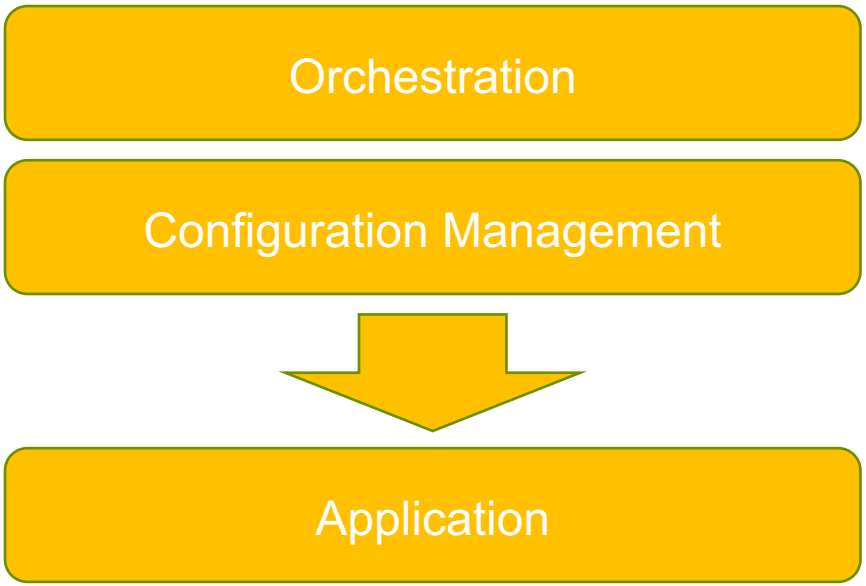
Common objections to change

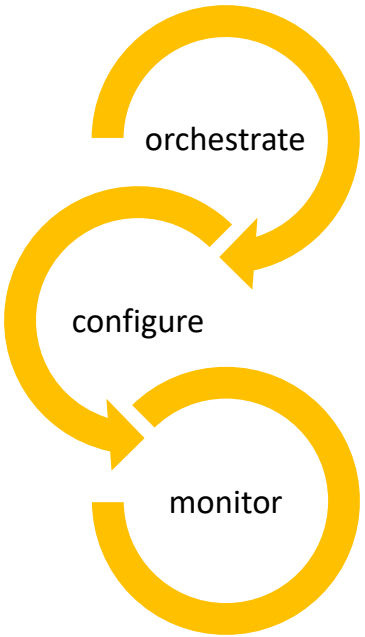
- Because they fear the consequences, which are unknown
- Because they do not trust
- Because they believe it is a fad and not permanent
- Because it threatens them in some way, perhaps by losing status or lack of skills
- Because they do not know why the change is needed

Common strategies to overcome resistance

- Establish throughout the organization that change is needed
- Communicate clearly the reason behind the change
- Offer support resources such as training when applicable
- Be consistent, well researched, and follow through with any committed changes
- Start with small victories and build on them

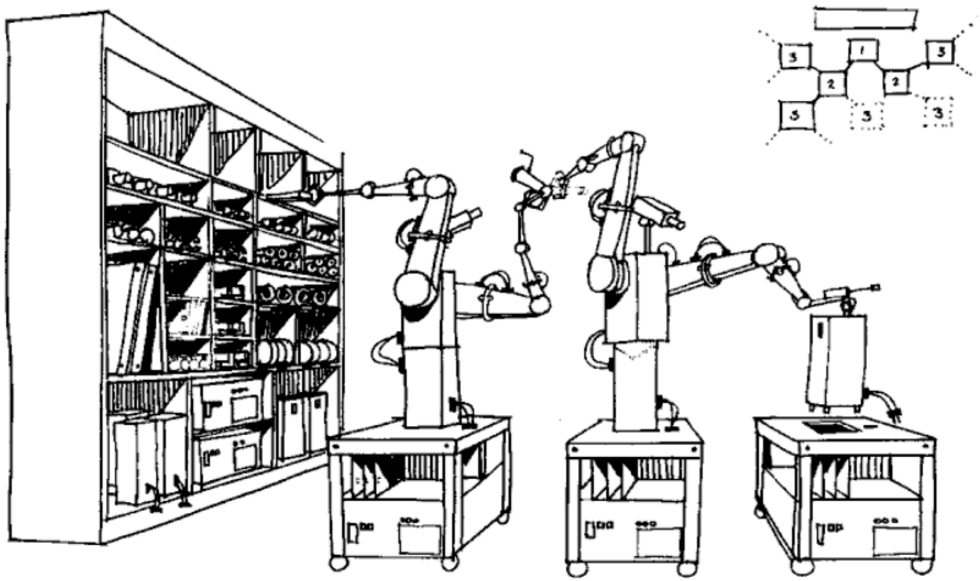
Modern Automation Stack





Orchestration

- Not the same as configuration management
- Infrastructure deployment
- Critical for infrastructure automation



We Automate, What/Why/Orchestrate…?

We have been automating things forever. Why is orchestration so hot these days?

What is different today is:

- Cloud computing, containers, Microservices, and horizontally scaling applications.
- The underlying tooling has changed, AND architectures are made up of many more moving parts than ever before.
- We need to dynamically build and tear down infrastructure on demand.

https://www.virtualizationpractice.com/the-race-to-own-orchestration-37184/

Deployment vs Provisioning vs Orchestration

'Deployment' is the process of putting a new application, or new version of an application, onto a prepared application server.

'Provisioning' is normally used by Ops to refer to getting computers or virtual hosts ready to use, and installing needed libraries or services on them.

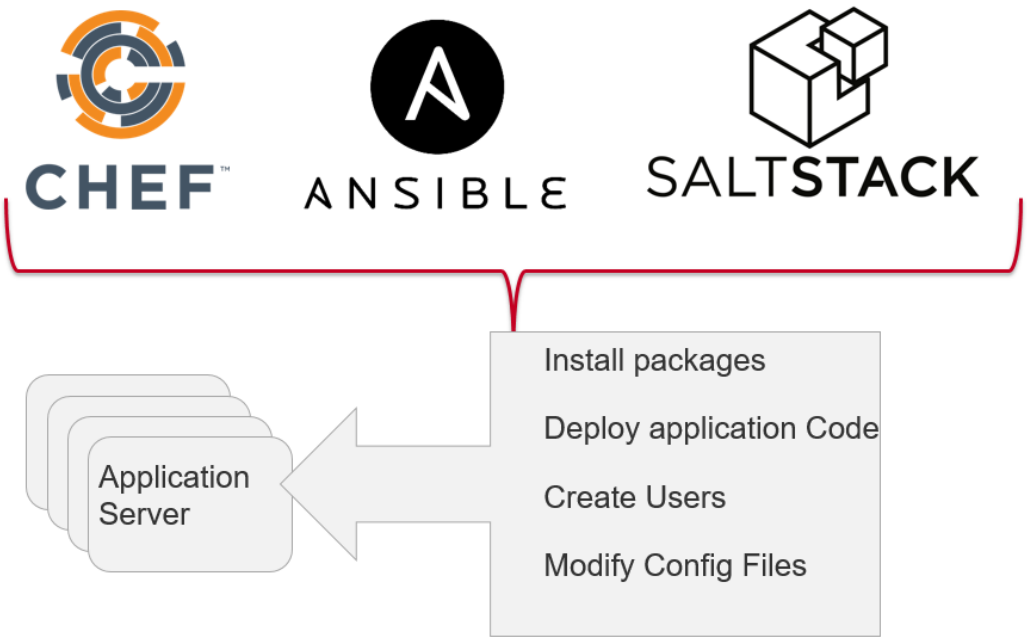
- **The thing to remember is that 'deployment' does not, as a rule, include 'provisioning'.**
- Developers often use Chef, Ansible or Puppet for server provisioning.

'Orchestration' means arranging or coordinating multiple systems. It's also used to mean "running the same tasks on a bunch of servers at once, but not necessarily all of them."

http://codefol.io/posts/deployment-versus-provisioning-versus-orchestration

Configuration Management

- Process for establishing and maintaining consistency of a product's performance, functional, and physical attributes with its requirements, design, and operational information throughout its life.
- Usually does not ORDER automation steps
- Usually acts on existing infrastructure



Configuration Management

Benefits of Configuration Management

- Quick Provisioning of New Servers
- Quick Recovery from Critical Events
- No More Snowflake Servers
- Version Control for the Server Environment
- Replicated Environments

Configuration Management Tools

www.chef.io/chef

Chef is a configuration management tool written in Ruby, and uses a pure Ruby DSL for writing configuration "recipes". These recipes contain resources that should be put into the declared state. Chef can be used as a client–server tool, or used in "solo" mode.

Configuration Management Tools

puppetlabs.com

Puppet allows you to define the state of your IT infrastructure, then automatically enforces the correct state. Puppet automates tasks that sysadmins often do manually, freeing up time and mental space so sysadmins can work on the projects that deliver greater business value.

Configuration Management Tools

www.ansible.com

Ansible is just about the simplest solution for operating system configuration management available. It's designed to be minimal in nature, consistent, secure, and highly reliable, with an extremely low learning curve for administrators, developers, and IT managers.

Configuration Management Tools

www.saltstack.com

Salt or SaltStack is a Python-based open source configuration management and remote execution application. Supporting the "infrastructure-as-code" approach to deployment and cloud management, it competes primarily with Puppet, Chef, and Ansible.

Infrastructure in the Cloud

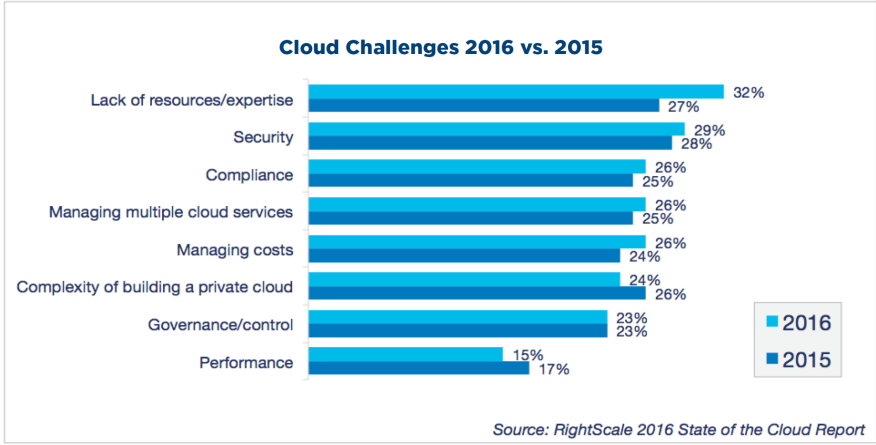
A typical Infrastructure in the cloud can deliver the following features and benefits:

- Scalability
- No investment in hardware
- Utility style costing
- Location independence
- Physical security of data center locations
- No single point of failure

Public Cloud Challenges

Top Public Cloud Challenges

- Lack of Resources/Expertise
- Security
- Compliance
- Managing Multiple Cloud Providers
- Managing Costs
- Complexity of Building a Private/Hybrid Cloud



Exercise
Identify Deployment Improvements

- Update your Value Stream Map based to reflect using Infrastructure as Code
  - Estimate new Lead Times and %CAs
- Identify how Infrastructure as Code improves your Deployment process

Lab 3
Create and Configure a Project

- Create an ASP.Net MVC Project
- Run Unit Tests
- Debug the ASP.MVC project

Take instruction from the notes

Exercise 01: Create an ASP.Net MVC Project

- Create a new project with ASP.Net template
- Configure Project Authentication

Exercise 02: Execute Unit Tests

- **Execute Existing Unit Tests**
- **Write a Failing Test**

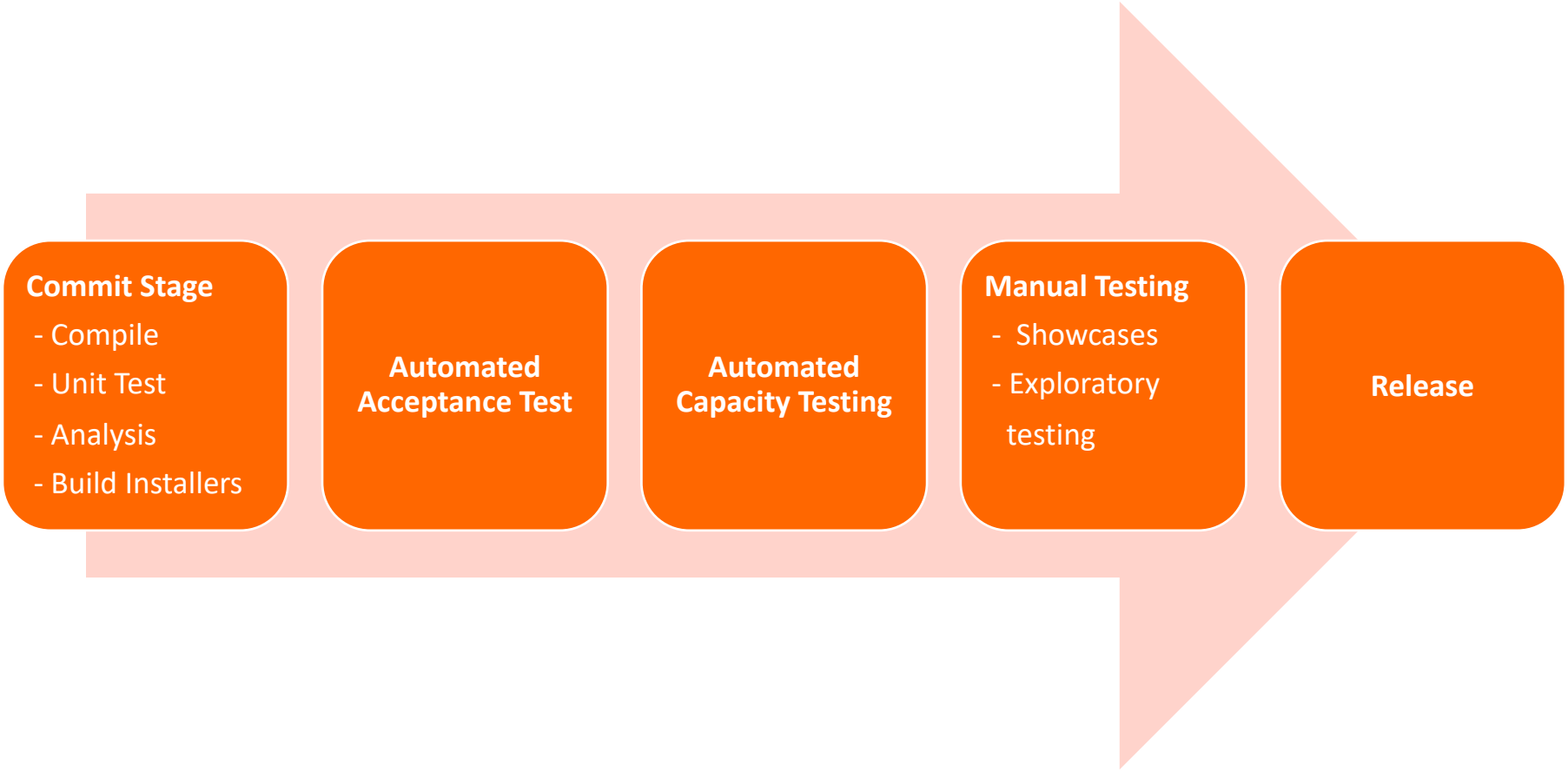
Exercise 03: Debug Project

- Debug Solution
- Register a new user
- Review created membership database

Deployment Pipeline and Scripting

Module 05

What is a Deployment Pipeline?



Source: *Continuous Delivery,* Humble & Farley

The Deployment Pipeline & Scripting: Design & Development

- Pipeline Design Principles
- Test layers
- Test stages
- Acceptance Testing
- The Enterprise System

Pipeline Design Principles

- Automation
- Testing
  - Locating the origin of bad code
    - Immediate feedback for the developers
  - Qualify & Integrate Components
    - Strategy that localizes the feedback to a smaller team while finding as many of the integration and operational issues as possible

Testing Layers

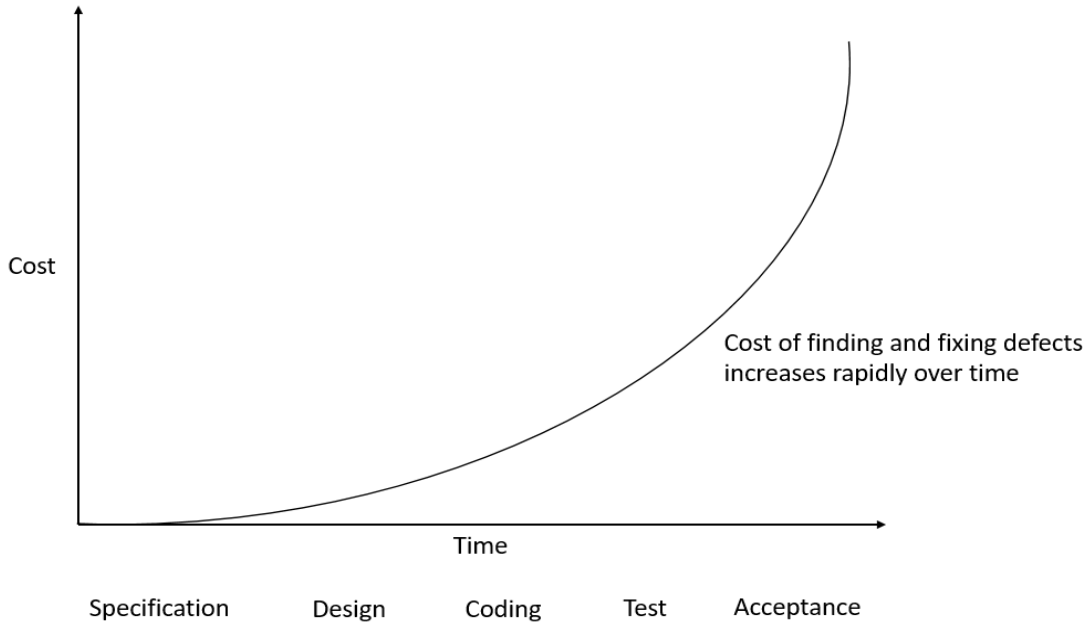
- Unit Testing & Static Code Analysis
  - Properly design unit tests
  - Unit test coverage
- Service Layer Testing
  - Test the service or components
- Integration Testing

Designing Test Stages

- Localize Feedback to Developers
  - Unit test -> feedback at the first layer
- Service Virtualization
  - Breaking up large systems into smaller
  - Localize issues -> easier to debug
- Create a Prioritized Backlog
  - Optimize speed by implementing Agile at team level

Testing Stages

The longer you wait to test, the more expensive it is to find and fix a bug. If you have multiple bugs, well!



Build Acceptance Testing

- Build Acceptance Test Strategy
  - Let it come from the acceptance criteria
  - Use tools like Cucumber
- System tests with certain objectives
  - Each test exercises the functionality of corresponding components
- Enforcing Code Stability
  - Code always deployable
  - Gated check-in

Deployment Pipeline and Scripting: Implementation

- Understand Deployment Pipeline
- Committing Code
- Gated Acceptance Testing
- Automating Deployment
- Testing Stages
- Implementing a Deployment Pipeline
- Build Tools Overview
- Deployment Scripting
- Build Scripting
- Automating Tests

Committing Code

- Commit Often
- Don't commit broken code
- Run Local Builds before commits
- Fail the build if any test fails
- Fix broken builds immediately
- Don't check out code from a broken build

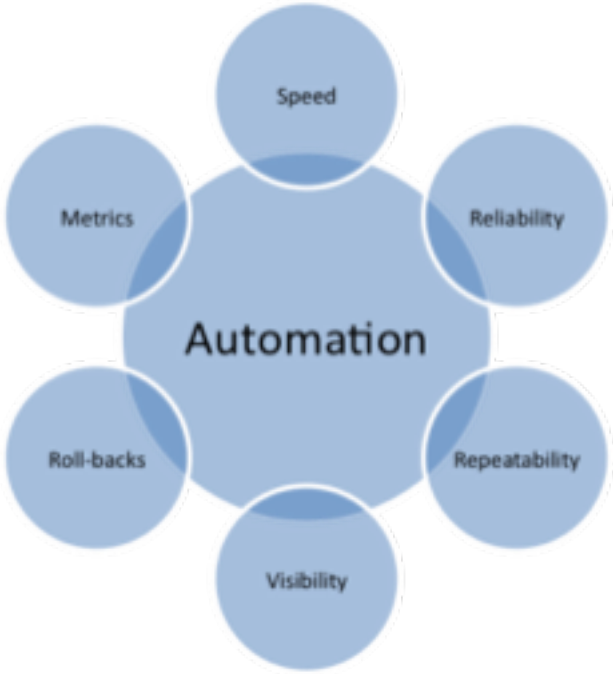
Gated Acceptance Testing

- Tests that are built from the acceptance criteria in user stories.
- Allows us to develop to established tests to help us define {done}
- Provides quick feedback on the quality
- Allows developers to measure working software (progress)

Automating Deployment

Benefits

1. Allows anyone to deploy the application
2. New environments can be readily supported
3. Less overhead from deployments
4. More frequent deployments are possible
5. More scalable due to less errors



Build Tools Overview

There are plenty of tools available in the market, including

- make
- ant
- nant
- Msbuild
- Maven
- ivy
- gradle

Lab: Automating Deployment

- **Exercise 01: Configure Build Tools**
- **Exercise 02: Create Database Deployment Project**
- **Exercise 03: Check Solution into Source Control**

Exercise 01: Configure Build Tools

- **Review Database Settings in Web.Config**

Exercise 02: Database Deployment Project

- **Create Database Deployment Project**
- **Build Database Deployment Project**

Exercise 02: Database Deployment Project (Continued)

Exercise 3: Add Solution to Source Control

- Add Solution to Source Control
- Commit Changes to Repository
- Test Gated Check-in

Design of a Strong Foundation

Module 06

07: Design of a Strong Foundation

- Adjusting the Architecture
- Ensuring the Build Process has independent parts
- Automated Testing
- Creating Test and other environments

❖ Adjusting the Architecture

- Existing architecture & DevOps Application
- Enabling Rapid development
- Testing and deploying components independently

Existing Architecture & DevOps Application

- Improving the Development Process with DevOps
- Well defined Architecture
  - Containerization
  - Configuration Management
- Smaller and loosely coupled Components

Evolving The Architecture

- Current architecture may prevent CD
- Moving to the right architecture takes time
- May require sacrificial interim steps

Discussion:

- What architectural changes will be needed?
- Will sacrificial interim steps be appropriate?

Enabling Rapid Development

- Creating a solid code Foundation
- Managing Different parts of Architecture
- Overcoming Integration Challenges
- Isolate products by loosely coupled components

Testing and Deploying Components *Independently*

- Testing Architecture Components Independently – by loosely coupled components
- Leveraging Code across a range of products
- Keeping the system Stable

❖ *Validate Build Process*

- Architectural diagram based on Artifacts
- Integrated System & Existing Components
- Building Dependencies & Component

  Ripple effects

Architectural Diagram Based on Artifacts

- Legacy Architecture
- Balancing Architectural changes & Process Improvements
- Structured approaches to Continuous Delivery

Integrated System & Existing Components

- Minimizing Product Differences
- Evaluating System Stability
- Integrating Existing Components

Building Dependencies & Component Ripple Effects

- Determining what makes a build go red
- Recovering from a bad build
- Getting back to a Green Build

❖ *Automated Testing*

- Unit and Service Level Testing
- When business logic is built into the UI

Unit and Service Level Testing

- Writing good test Automation
- Running Tests on a Daily Basis
- Dealing with large numbers of Tests efficiently

When Business Logic is Built into the UI

- The challenge of Code Stability
- Creating and adhering to a good plan
- Focusing on the Team: Developers working with executives

❖ *Creating Test Environments*

- Production Environment
- Designing Tests for sustainability
- Pairing Developers with QA

Production Environment

- Running Automated Tests Cost-effectively
- Increasing Test Coverage
- Test Simulators

Designing Tests for Sustainability

- Code Defect or Test Defect?
- Manual Testing vs Automated Testing
- Software Updates

Pairing Developers With QA

- Pairing Object oriented expertise with Manual Testing
- Creating the Automated Test Framework
- Increase the test coverage

❖ *Maintainability*

- Create a Test Results Database
- Design tests to discover localized defects
- Using component based testing
- Merge your specification and test documentation

Create a Test Results Database

- Managing Test Results
- Reporting Test Results
- Statistical Approach to Test Pass Rates

Design Tests to Discover Localized Defects

- Automating existing Manual Scripts
- Localizing the cause of failure
- Requirements for running thousands of tests

Using Component Based Testing

- Isolating Components of the System
- Testing designed for specific functionality
- Statistically Analyzing Test Metrics

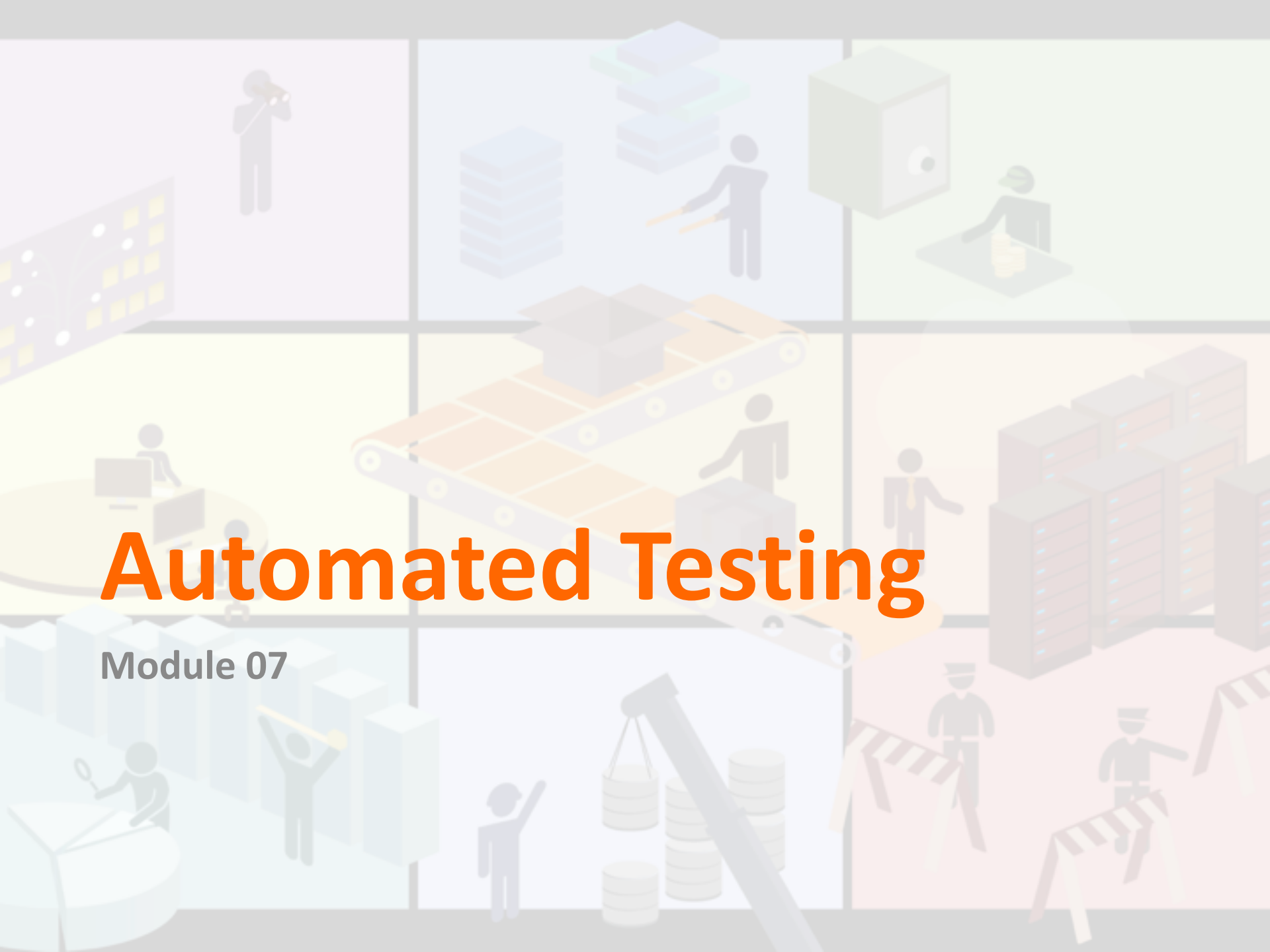
Merge Your Specification and Test Documentation

- Associating requirements with Code
- Creating Specifications in an executable form
- Testing to meet Audit Requirements

Exercise: Identify Deployment Improvements

- Update your Value Stream Map
  to reflect Automating Deployment
  - Estimate new Lead Times and %CAs
- Identify how Automating Deployment improves your Deployment process

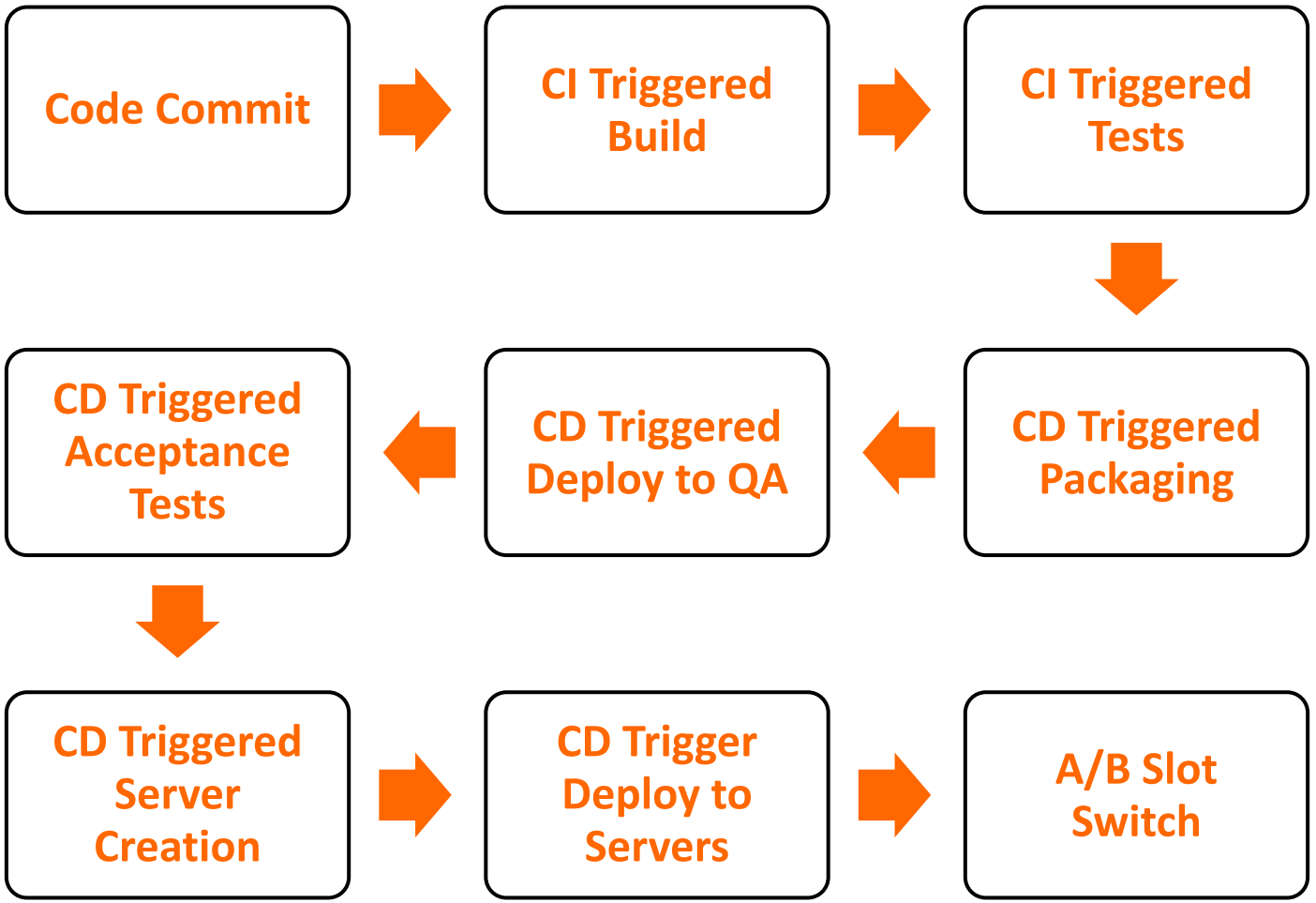
- *Architectural changes*
- *Automated build scripts*
- *Automated Infrastructure*
- *Automate deployment scripts*
- *Automate test*



Automated Testing

Module 07

Continuous Delivery Pipeline



Module 7: Automated Testing

- Creating Acceptance Tests
- Creating Unit Tests
- Automating Unit Tests
- Automating Capacity Testing
- Parallel Testing

Create Acceptance Test

Acceptance Testing Planning/Tasks

Acceptance Testing Planning/Tasks

- Acceptance Test Plan
- Acceptance Test Cases/Checklist
- Acceptance Test

When is it performed?

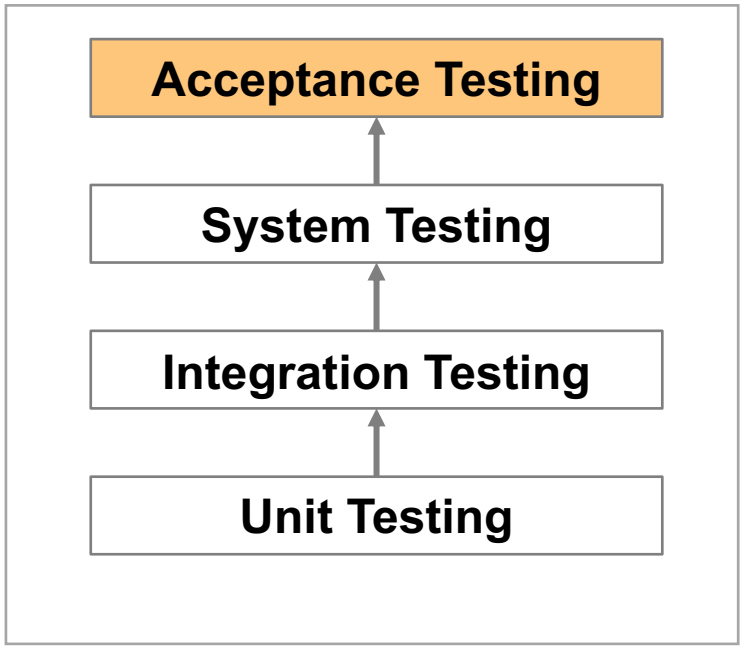
- After System Testing
- Before making the system available for actual use

Who performs it?

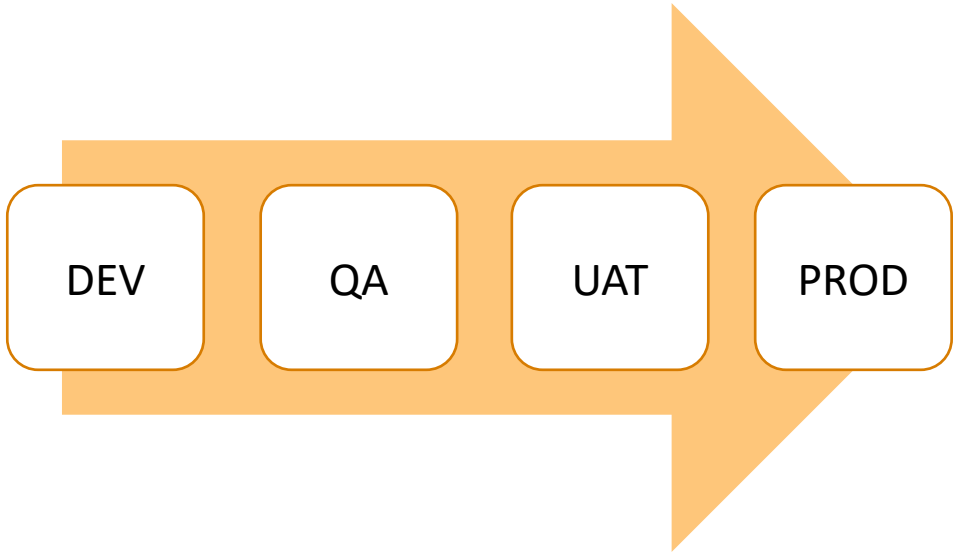
- **Internal Acceptance Testing** (Also known as Alpha Testing)
- **External Acceptance Testing** - is performed outside, non-developers of the software
- **Customer Acceptance Testing -** is performed by the customers of the organization
- **User Acceptance Testing** (Also known as Beta Testing) is performed by the end users of the software

Acceptance Test

Acceptance Testing Fundamentals



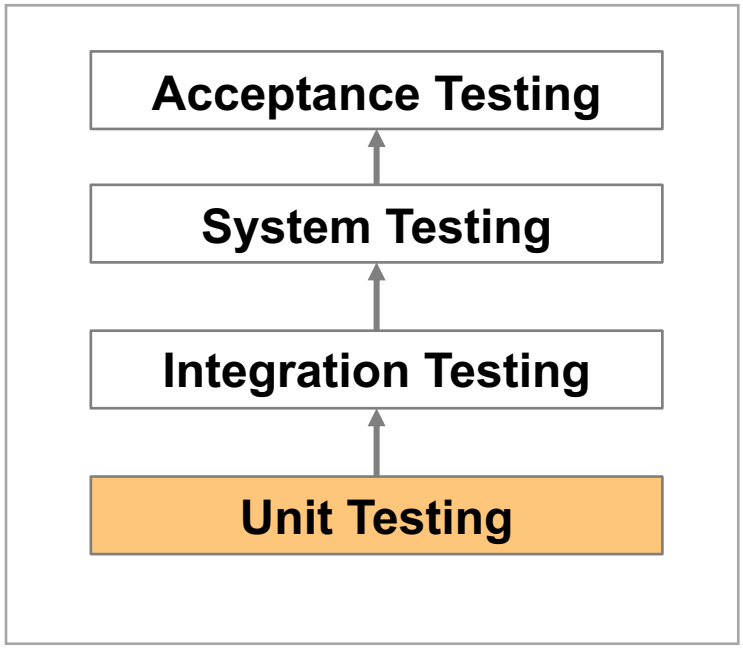
Where does AT fit in?



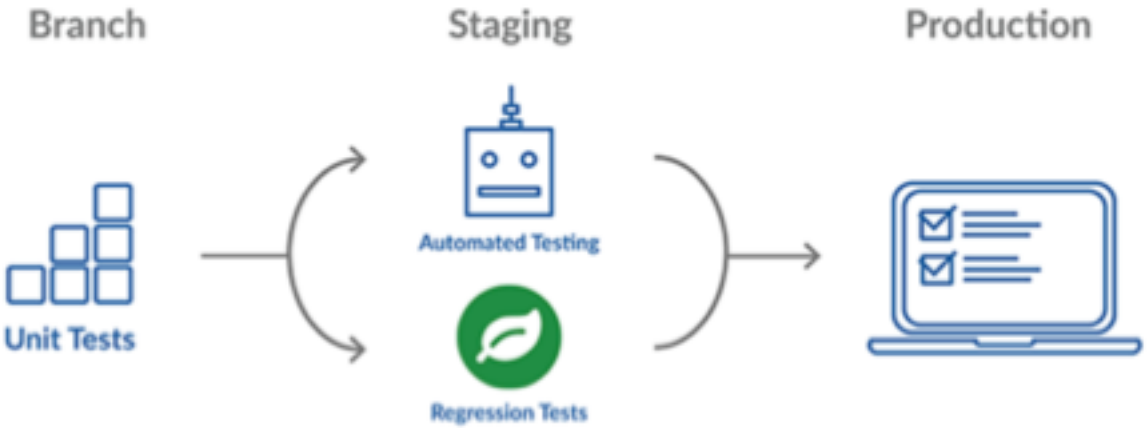
When is AT performed?

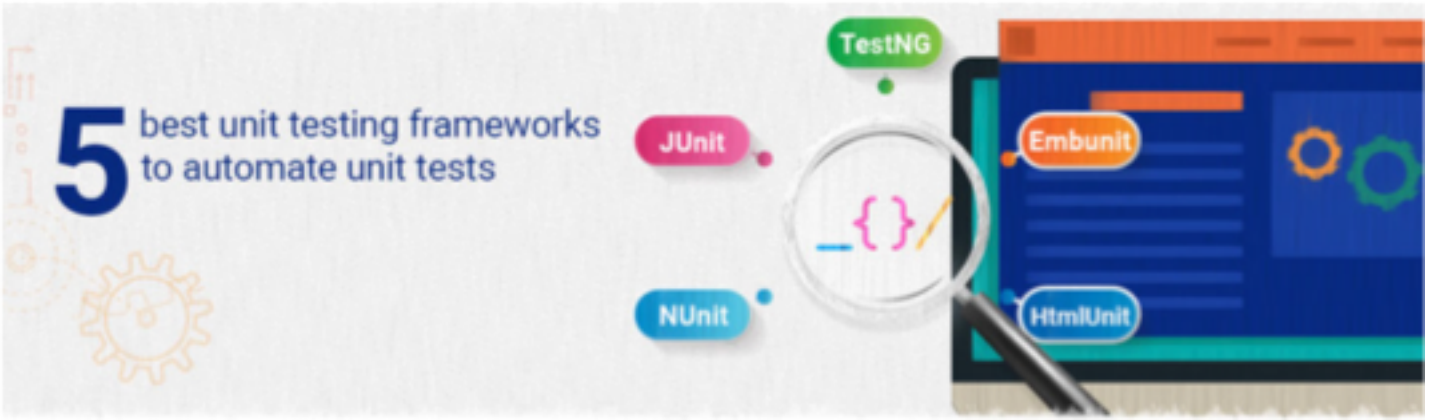
Unit Tests

Unit Testing Fundamentals



Automating Unit Tests





Capacity Testing

- The primary goal of testing is to establish the benchmark behavior
- The two types of capacity tests are
  - Performance and
  - Load/stress tests

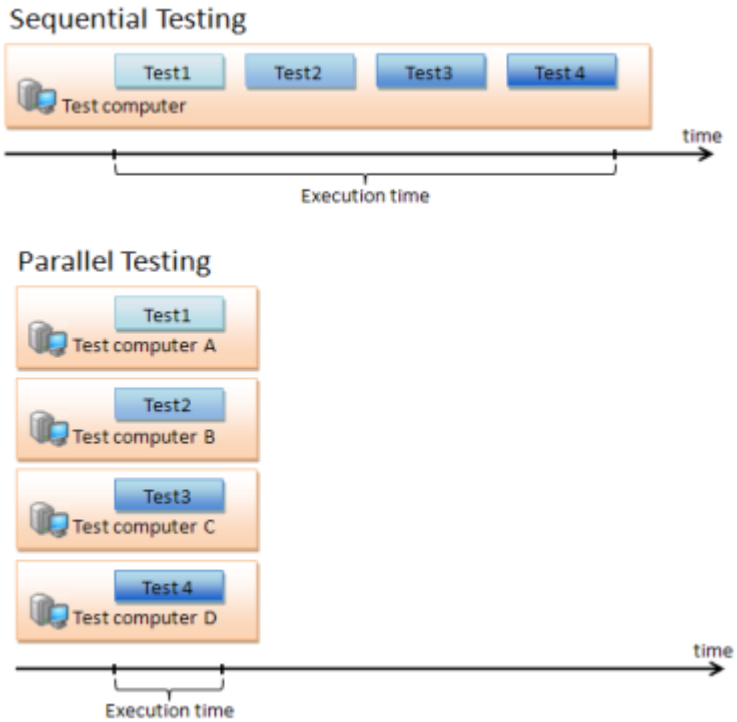
Automated Capacity Testing

Top 15 Performance Testing Tools

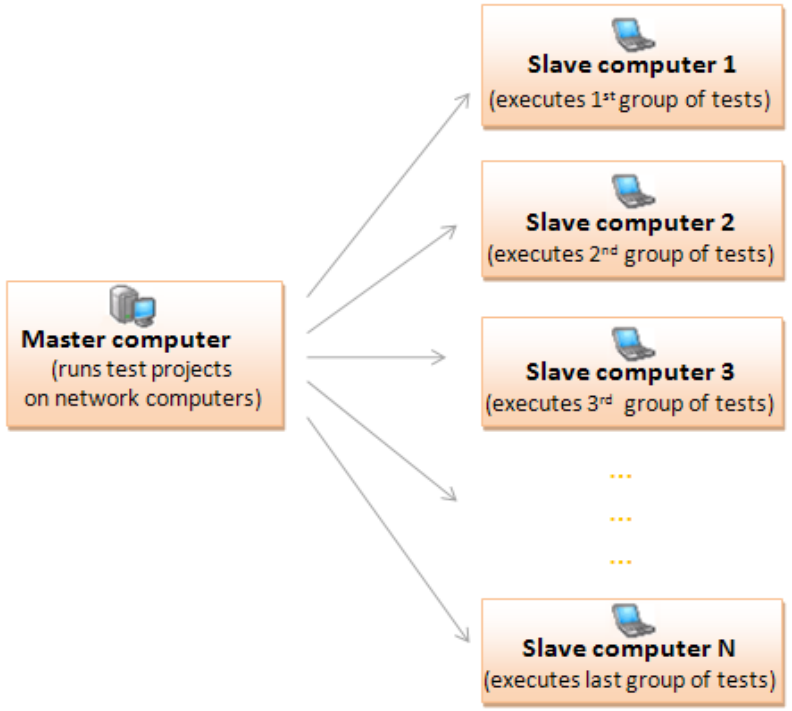
- WebLOAD
- LoadUI NG Pro
- Apica LoadTest
- LoadView
- Apache JMeter
- LoadRunner
- Appvance
- NeoLoad
- LoadComplete
- WAPT
- Loadster
- LoadImpact
- Rational Performance Tester
- Testing Anywhere
- OpenSTA
- QEngine (ManageEngine)
- Loadstorm
- CloudTest
- Httperf

Parallel Testing

Sequential vs Parallel Testing

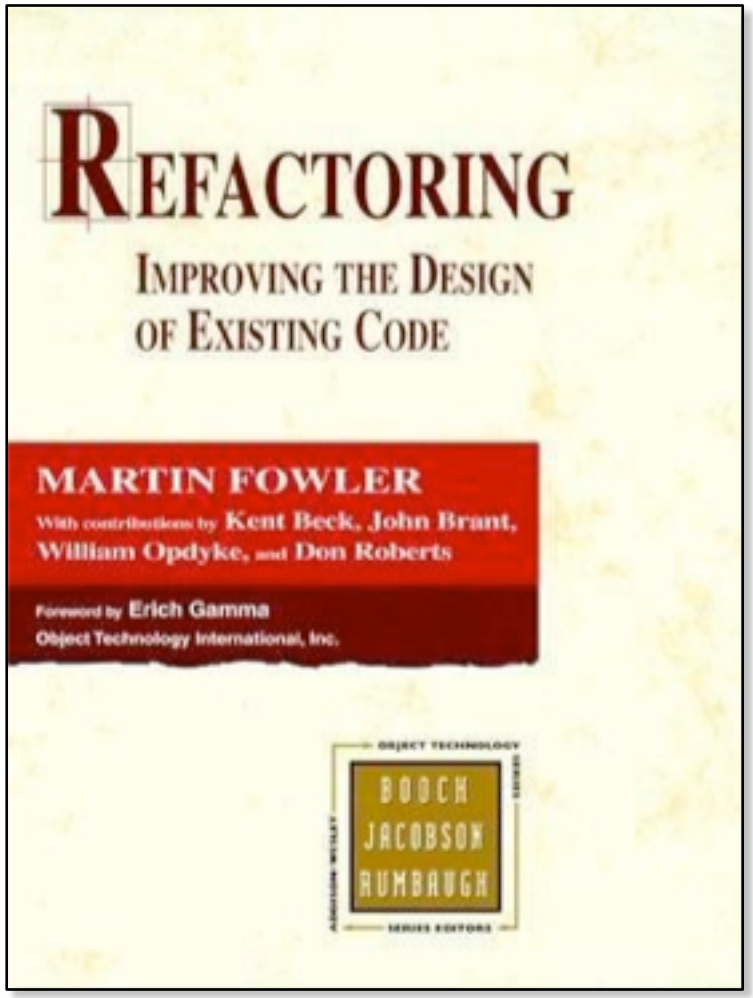


Sequential vs Parallel



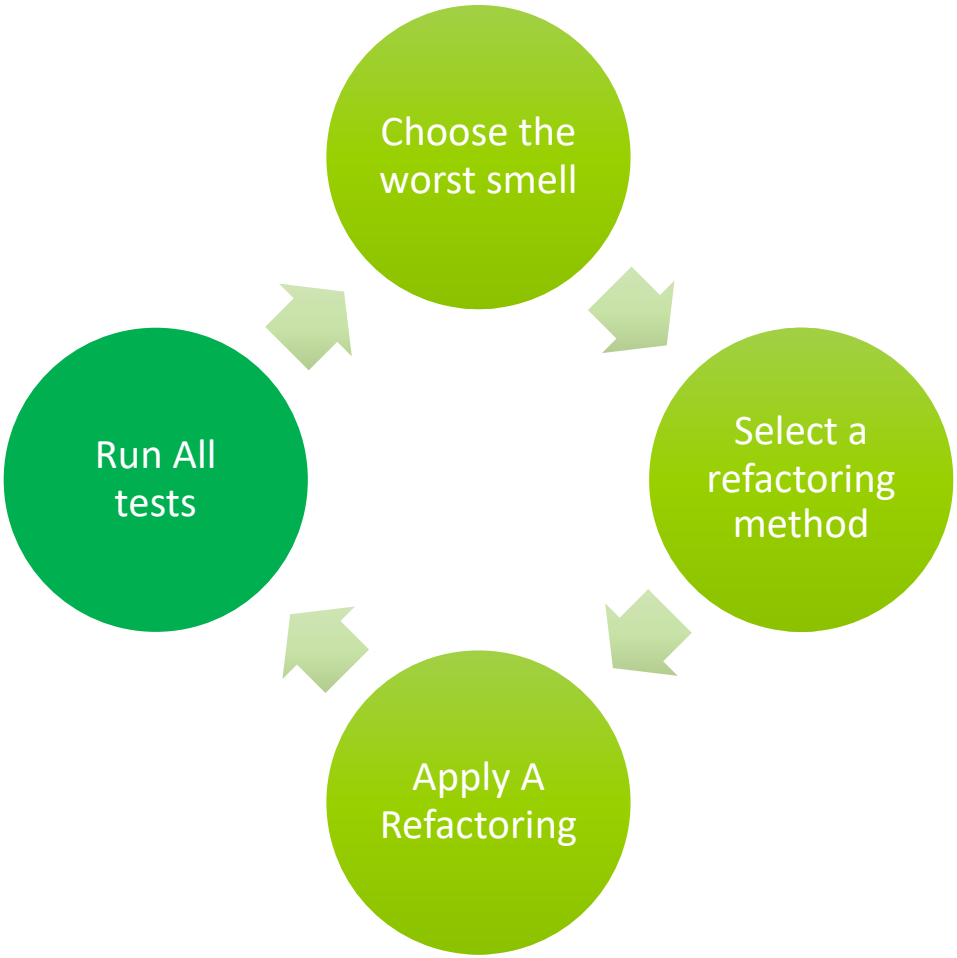
How Parallel is Performed

Refactoring



Refactoring is an ongoing activity of improving the internal structure of code without making externally visible changes to the functionality

Refactoring Cycle



Pre-conditions

- All Tests are Up-to-date, If tests need to be updated do so prior to refactoring
- All Tests are **Green**

Post-conditions

- Refactored code should be as good as the pre-existing code or better
- All Tests are **Green**

Refactoring Smells

Simple Smells

- Comments
- Large class
- Large methods
- Long parameter list

Smells in Name

- Uncommunicative Name
- Inconsistent Names

Smells in Conditional logic

- Null check
- Complicated Boolean Expression

When Are we Done?

- Passes all the tests.
- Has no duplicated logic.
- States every intention important to the programmers.
- Has the fewest possible classes and methods.

Lab: Automating Test Execution

- Review Code Coverage Settings
- Make a minor change
- Trigger Automated Build and Test

Exercise 01: Automating Unit Tests

- **Review Code Coverage Settings**

Exercise 02: Automating Unit Tests

- **Make a minor change**

Exercise 02: Automating Unit Tests (Continued)

Exercise 2B: Automating Unit Tests

- **Trigger Automated Build and Test**

Exercise 2B: Automating Unit Tests (Continued)

Developing on Trunk

Module 08

08: Trunk Culture

- Understanding Trunk Development
- Responsibilities of Team Leaders
- The New Mindset: No more Branching

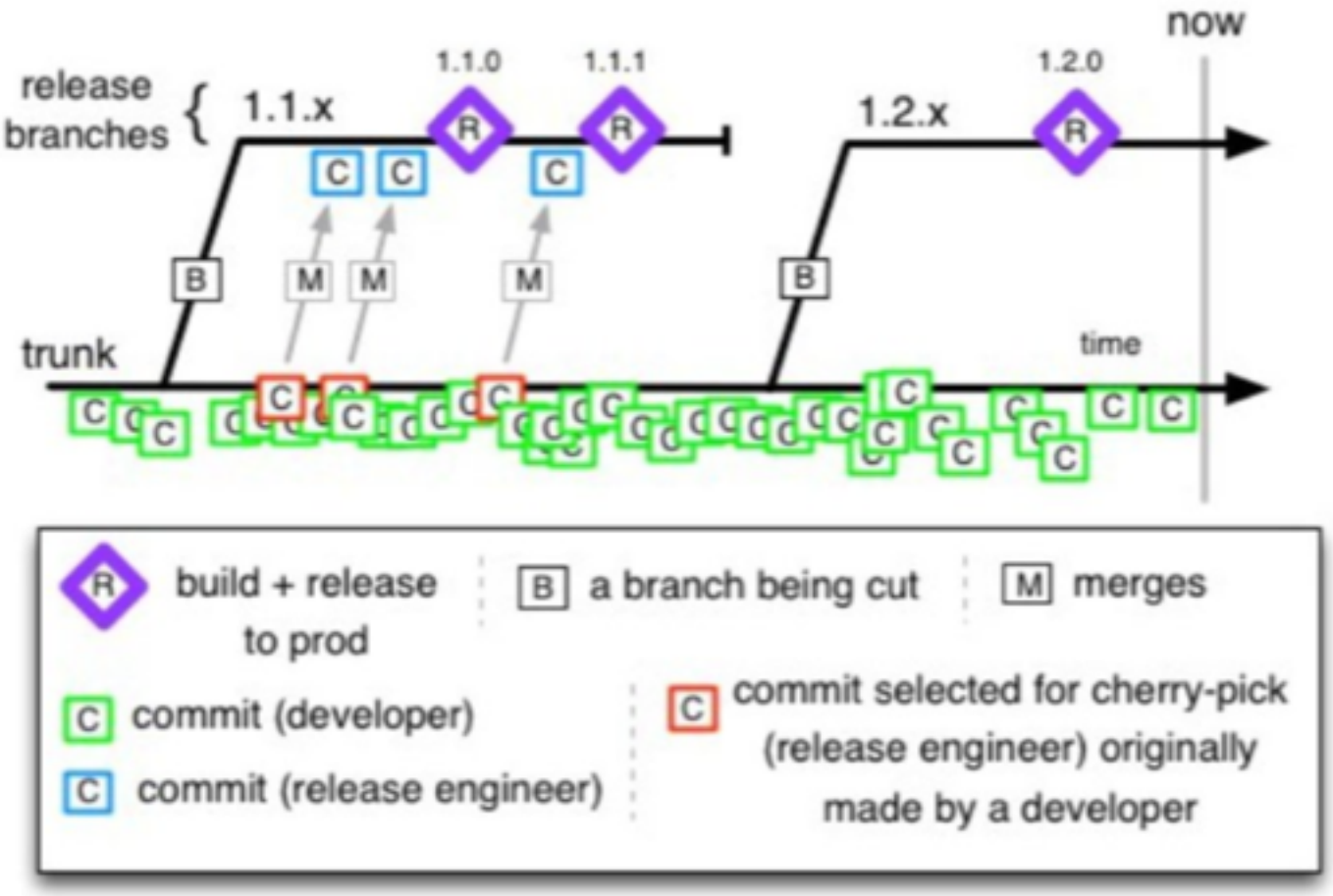
Understanding Trunk Development

- Executives need to embrace Trunk
- Trunk development Techniques
- Feature Flag Techniques

Trunk Development

- All developers work on a **single branch**
- Often it's the **master** branch. They commit code to it and run it
- In some cases, they create **short-lived feature branches**. Once code on their branch compiles and passess all tests, they merge it straight to **master**
- Ensures that development is continuous and prevents developers from creating merge conflicts that are difficult to resolve

Trunk Development



One Trunk – No More Branches!

- Prevent Duplicated Work in Branches
- Branching issues can be solved by developing on the trunk
- Multiple Branches create inefficiency and duplicate costs
- Look for process changes that eliminate the need for branches

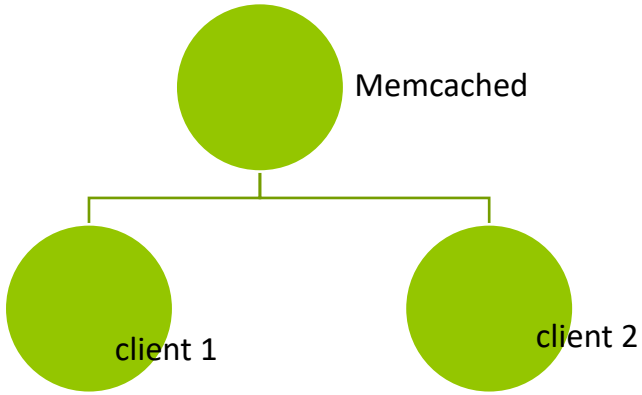
When does this work and when doesn't

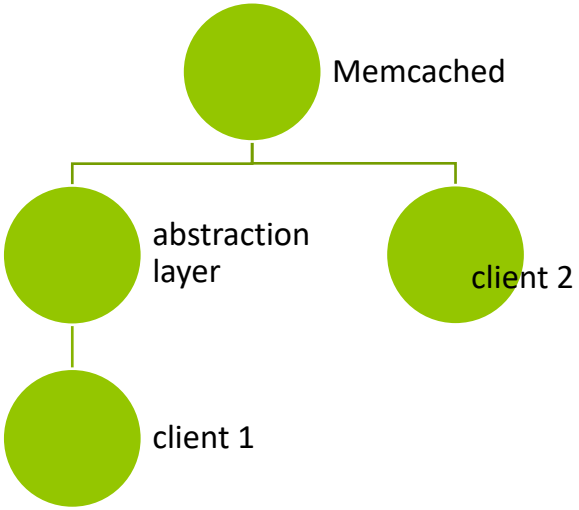
- **When it works best**
  - When you are starting up
  - When you need to iterate quickly
  - When you work with more senior developers
- **When it doesn't work**
  - When you have lot of junior developers
  - When you work with large established teams
  - You want strict control over who can commit when

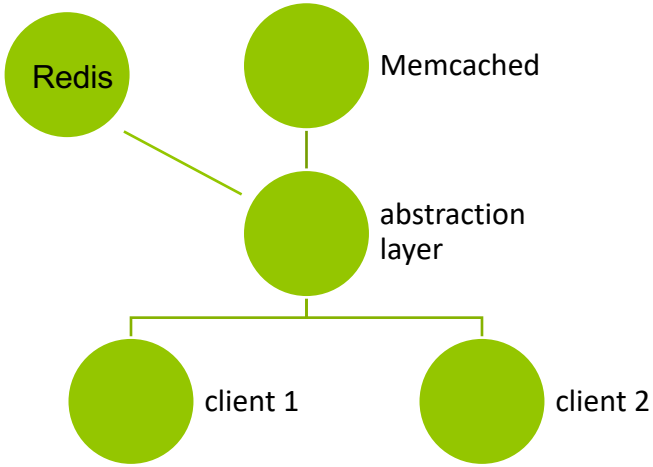
Trunk Development Techniques

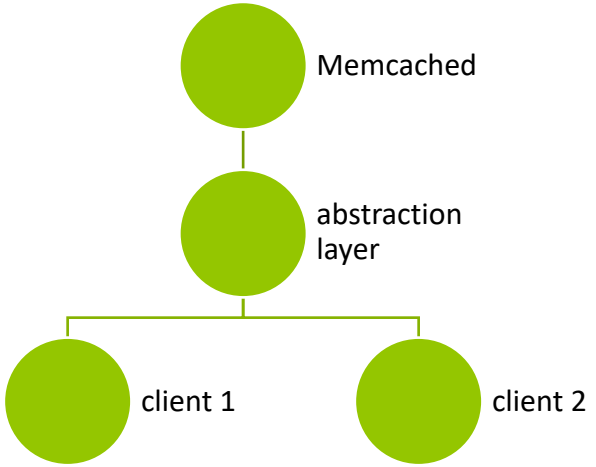
- Versioning and Tagging
- Re-architecture through Abstraction
- Feature Flags

Re-Architecture Through Abstraction Technique









Feature Flag Technique

- Kill Switch
- Beta Feedback
- Feature Targeting
- Don’t roll back – just turn off the feature
- Hypothesis-Driven Development
- Subscription Plans

❖ *Responsibilities of Team Leaders*

- **Transforming Development processes**
  - The big Challenge
  - Coping with attitude adjustments
  - Understanding the objectives
- **Investing in Process Changes**
  - Effective Communication
  - Fixing the issue by re-committing code
  - Holding Developers accountable for bad code

- **Using tools to force the action**
  - Utilizing Technology to force the Issue
  - Optimizing available tools
  - Delivery of new Capabilities

❖ *The New Mindset: No more Branching*

- Resistance to the Trunk
- Real Time Feedback
- Converting the non-believers
- Arguments for Branching
- Shifting Mindsets

Exercise: Identify Deployment Improvements

- Update your Value Stream Map based

  to reflect Using **Automated Testing** and **Trunk**

  Development

  - Estimate new Lead Times and %CAs

Telemetry, Monitoring & Logging

Module 8

Telemetry: Metrics, Monitoring, Alerting

Telemetry = “an automated communications process by which measurements and other data are collected at remote points and are subsequently transmitted to receiving equipment for monitoring.”

Event = a change in the state of something that is relevant

Metric = data that are captured because they are useful for detecting Events and understanding current state and history

Monitoring = watching metrics to identify Events that are of interest

Alerting = Notifying people when an Event requires action

Five Telemetry “must haves”

- Instrumentation of applications and infrastructure for data collection
- Storage and easy retrieval
- Data dashboards which are aggregated, formatted and presented in a way suitable for the business and IT staff
- Alerts and notifications for event priorities
- Remediation triggers and/or automation

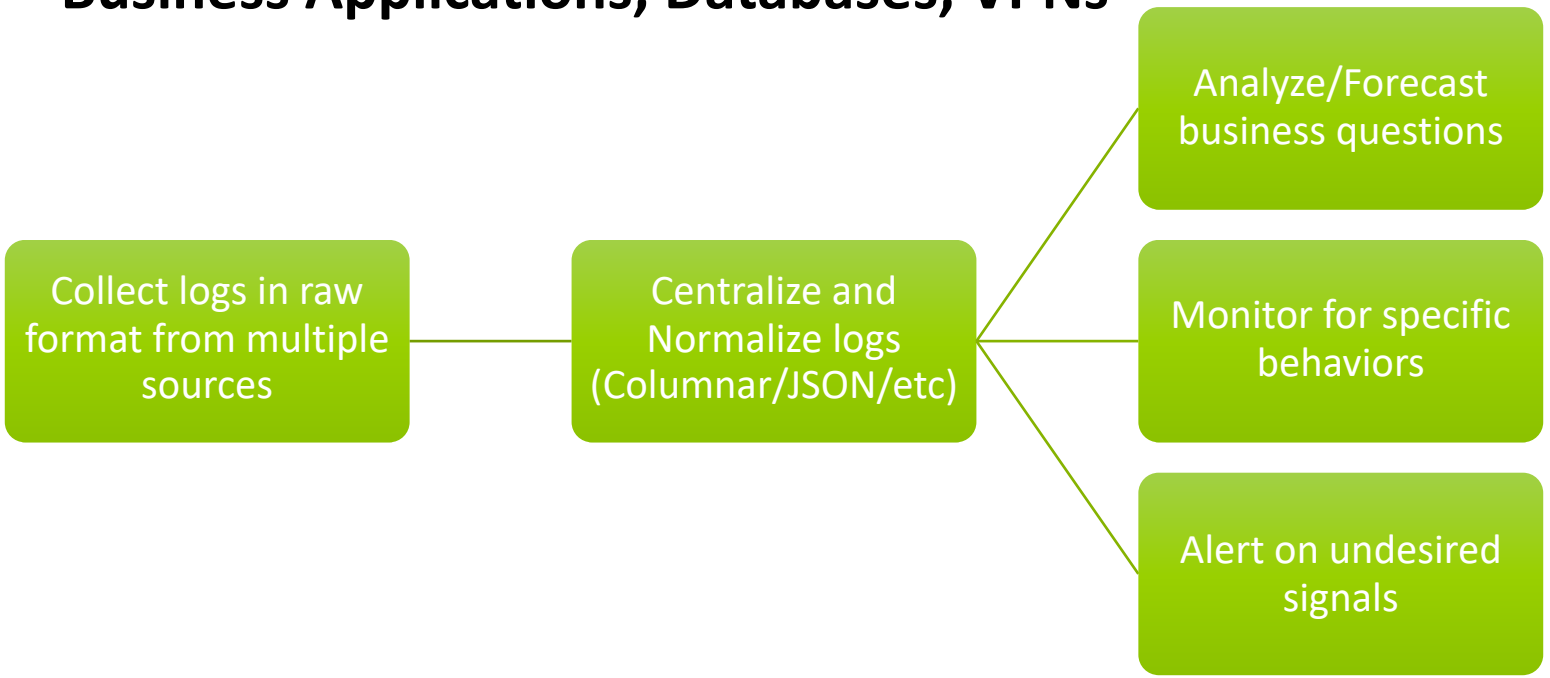
Telemetry Principles

- Automate Telemetry
- Build Monitoring and Measurement in:
  - Production Environment
  - Development & Test environments
  - Deployment Pipeline
- Use alerts to boost efficiency
- Make information visible to all
  - Within IT and to Customers

Log Aggregation

All logs – Audit records, Transaction logs, Intrusion alerts, Connection logs, User activity, Various alerts/Messages

From all sources – Firewalls, Routers/Switches, Web Servers, Business Applications, Databases, VPNs



Common DevOps Metrics

- Number and frequency of software releases
- Volume of defects
- Time/cost per release
- MTTR (Mean Time to Recover)
- Number and frequency of outages / performance issues
- Revenue/profit impact of outages / performance issues
- Number and cost of resources

Telemetry Tool Set

No one tool is sufficient for Telemetry

- Data Collection
- Data Storage
- Data Aggregation
- Data Retrieval
- Data Visualization
- Dashboards

System Monitoring Tools

- Librato
- Monit
- Nagios
- PagerDuty

Log Aggregation Tools

- Splunk
- ELK

Visualization Tools

- Kibana
- Grafana

Using Telemetry to Measure and Improve

- Enterprise Objectives
- Using metrics
- Feedback
- Aligning Goals

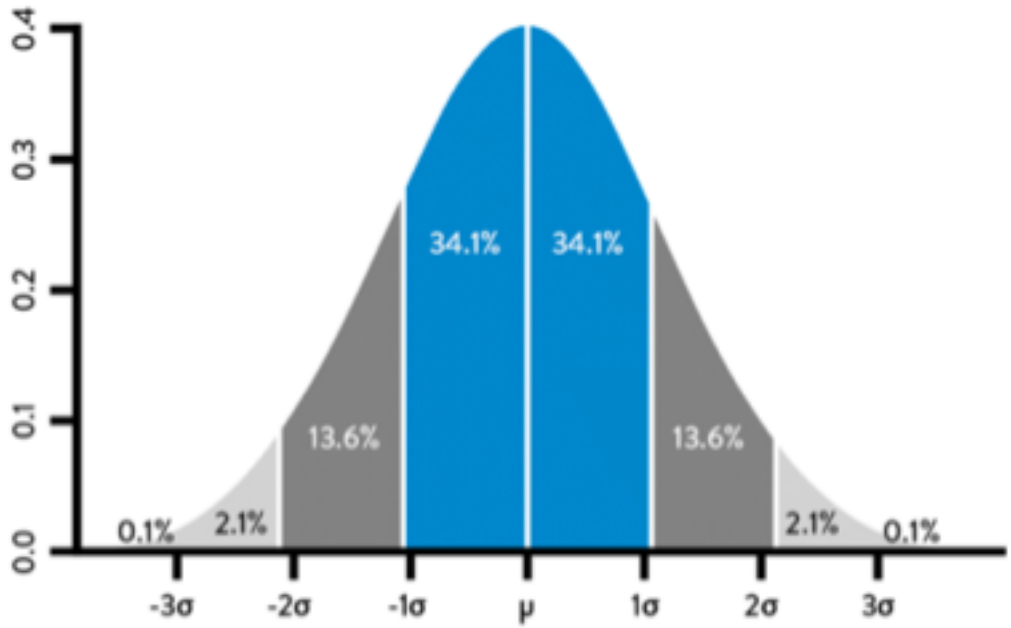
Using Data to Identify Challenges

- Using metrics to identify goals
- Utilizing Objectives to guide the work
- Where are developers struggling, and why?

Use Telemetry To Anticipate Problems

Use statistical techniques to understand “normal”, then address the “abnormal”

- e.g. Mean and Standard Deviation



Example: NetFlix

Use Telemetry To Anticipate Problems

Monitor for (and alert on) trends.. e.g.:

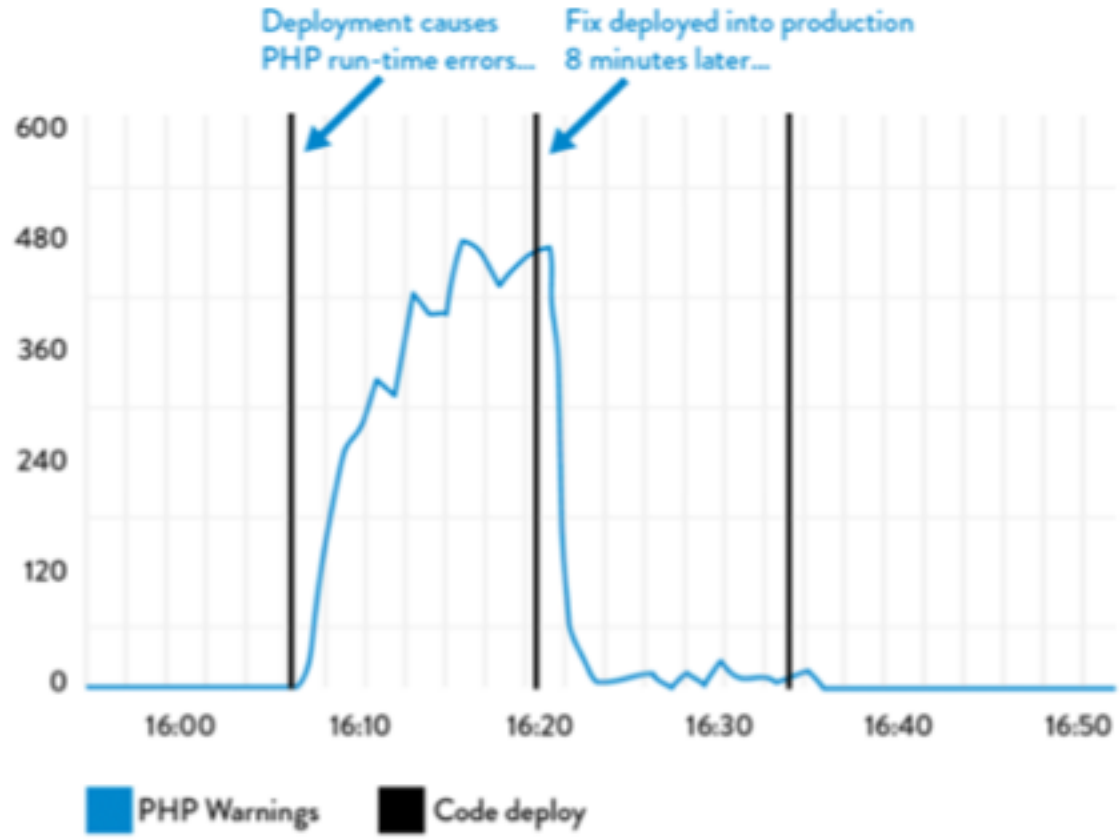
- Increasing web page load times
- Decreasing server free memory
- Decreasing available disk space
- Growing database transaction times
- Declining number of functioning servers behind the load balancer

Use Telemetry for Safe Deployment of Code

Actively Monitor key metrics immediately before and after each deploy

- Take action on unexpected changes

Example: Etsy.com



Use Telemetry to understand Issues at the Executive Level

- Communicating Challenges to Executives
- Executives are there to help
- Executive participation in the cultural shift

Using Telemetry Metrics to Increase Progress

- A new role for executives
- Understanding which Processes work
- Learning what needs improvement

Using Telemetry Metrics to Identify Challenges

- Sharing ideas for improvement
- Engineers, Developers, and Executives
- Trust enables a greater ability to fix issues

Using Telemetry and Metrics to Improve Stability

- What are the teams achieving?
- Integrating success into future iterations
- Adjusting processes to prevent errors

Customer-Oriented Telemetry

Hear the “Voice of the Customer”

- Applications capture:
  - Customer activities
  - User actions and mistakes
  - Business activity

Discuss: Customer-Oriented telemetry

- What data would provide insights?

Using Telemetry to Understand the Voice of the Customer

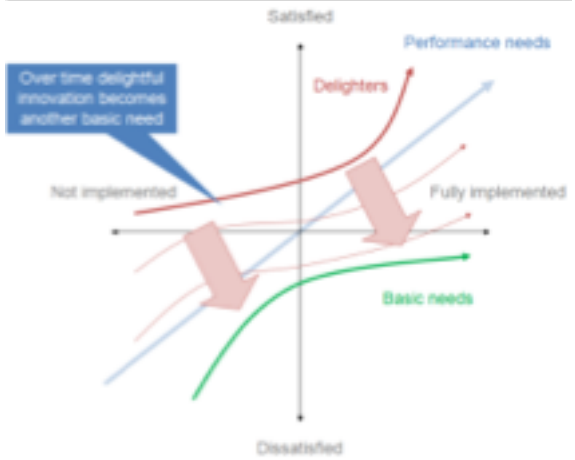
Critical to Quality (CTQ) Chart

Customer Journey Map

POV MadLib

Empathy Map

KANO Model



Exercise: Identify Deployment Logging

- Identify what data should be logged in your Deployment process

Discussion
Voice of the Customer

- What telemetry can be useful to understand the Voice of the Customer?
- How could you use it?

Exercise: Identify Telemetry to be Reported, Displayed, or Alerted

- Identify what data should be logged in your Deployment process
- What telemetry can be useful to understand the Voice of the Customer?
- How could you use it?
- Identify what telemetry should be reported, displayed or alerted on in your Deployment process
- Determine if any sensitive information or PII is being (or should be) logged. If so determine what to do about it. (e.g. stop logging it, or protect it)

Architectures for Low Risk Deployment

Module 9

Architecture for Reduced-Risk Deployment

- **Feature Flags (Feature Toggles)**

  Discussed Earlier
- **Microservices**
- **Blue-Green Deployments**
- **Containerization**

What Are Microservices?

- Object-Oriented Design – OOD (1980's)
  - Design the application as a collection of objects
  - (Encapsulation, Information hiding, Tight cohesion, Loose coupling, SOLID principles – in notes)
- Service Oriented Architecture – SOA (2000's)
  - Take OOD the next step
  - Implement the application as a collection of Services
  - Each service is a separate executable object
- Microservices (2010's)
  - Take SOA to the extreme
  - Each service is small and simple

Why Use Microservices?

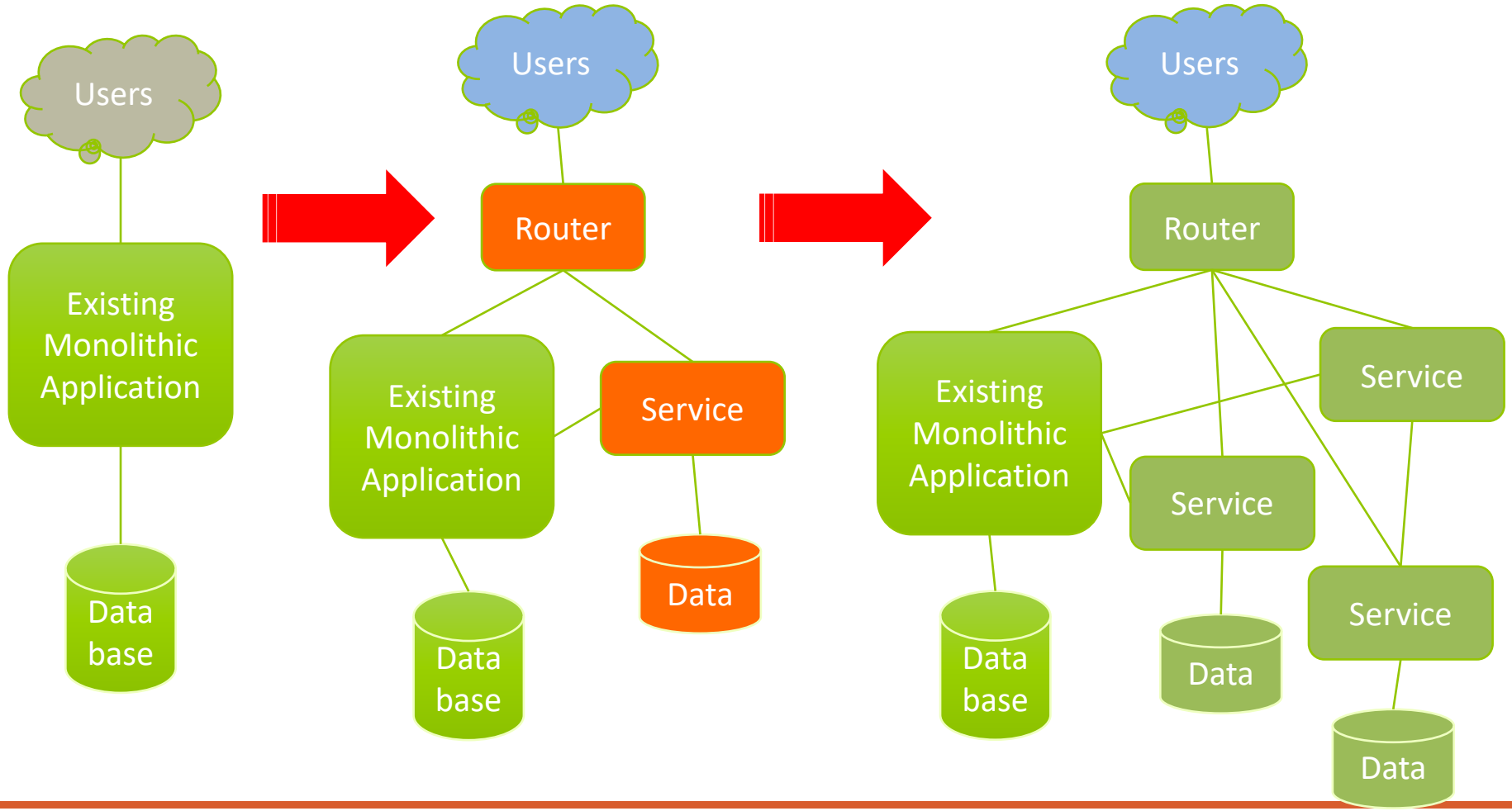
- Overall system resilience
  - Service failure does not kill the application
  - Service failure may be invisible to users
- Scalability
  - Replicate a Microservice to accommodate demand
- Observe Conway's Law
  - Small Agile Teams & Microservices

Re-architecting Applications Gradually
The Strangler Pattern

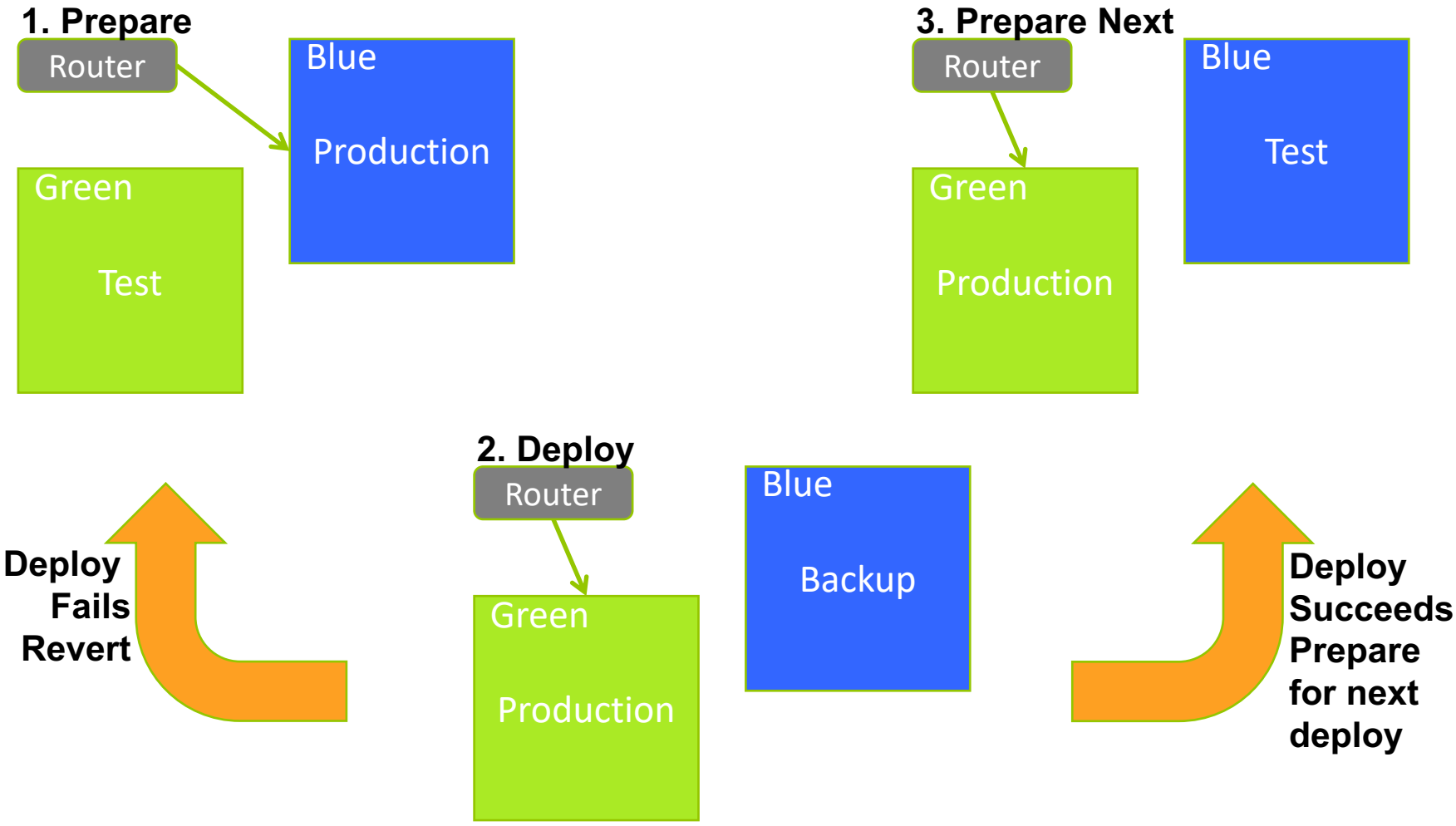
Why The Strangler Pattern?

- Avoid risky application rewrite projects
- Start seeing the benefits of SOA quickly
- Begin moving toward smaller market-oriented teams

How the Strangler Pattern Works



Blue-Green Deployment Pattern



What is Docker?

The Docker Project

Open Source Project

- 2B+ Docker Image Downloads
- 2000+ contributors
- 40K+ GitHub stars
- 200K+ Dockerized apps
- 240 Meetups in 70 countries
- 95K Meetup members

Docker Inc

Containers as a Service provider

- Integrated platform for dev and IT
- Commercial technical support

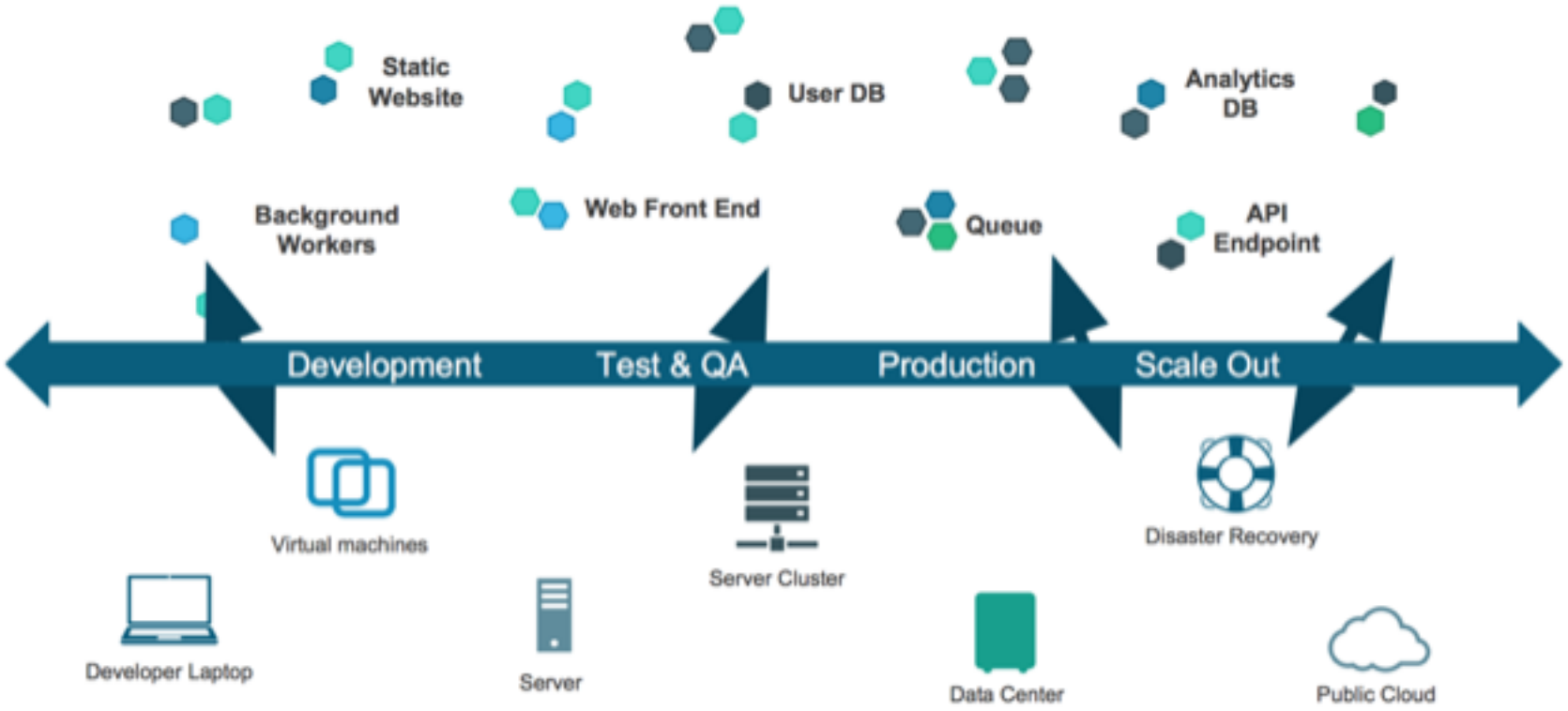
Docker project sponsor

- Primary sponsor of Docker project
- Supports project maintainers

Application Evolution

Challenge: The Dependency Matrix

"It works on MY machine."

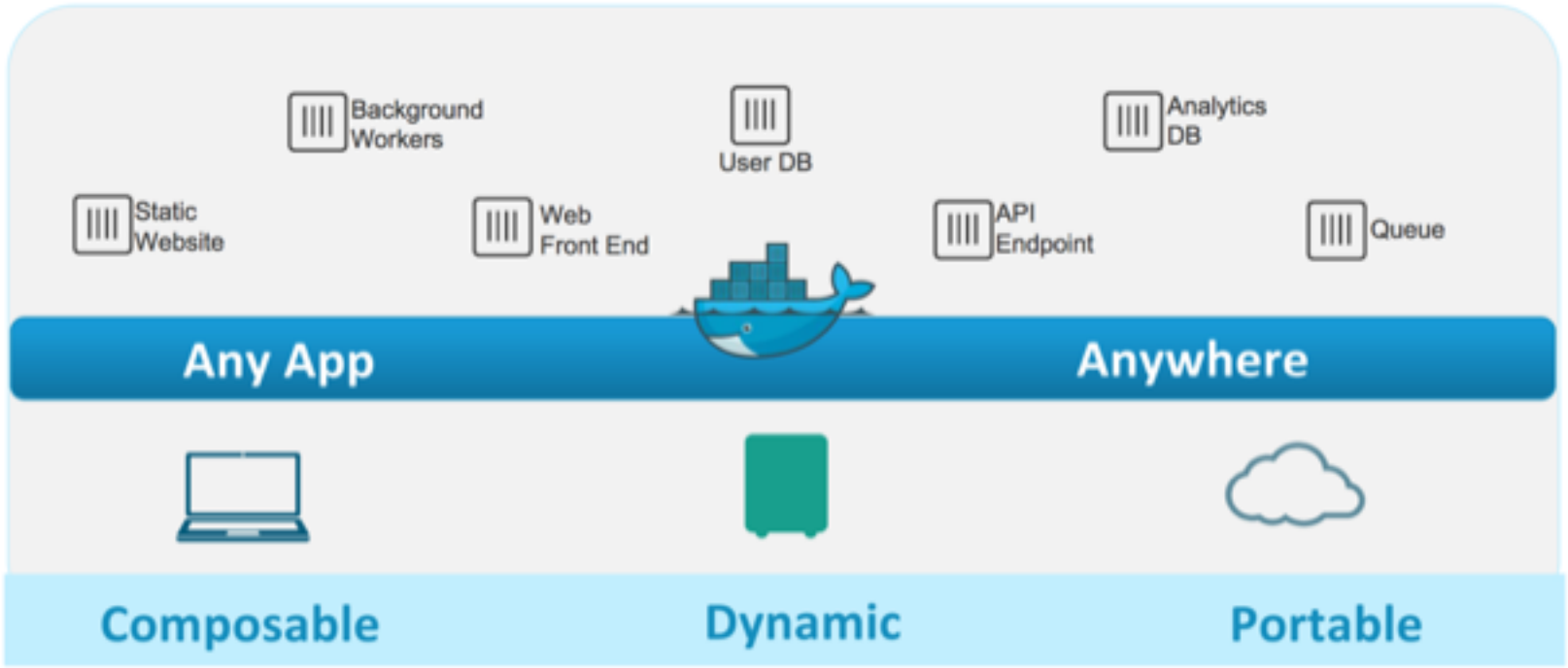


Containers as a Solution

Container

- Packages up software binaries and dependencies
- Isolates software from each other
- Container is a standard format
- Easily portable across environment
- Allows ecosystem to develop around its standard

Containers as a Solution



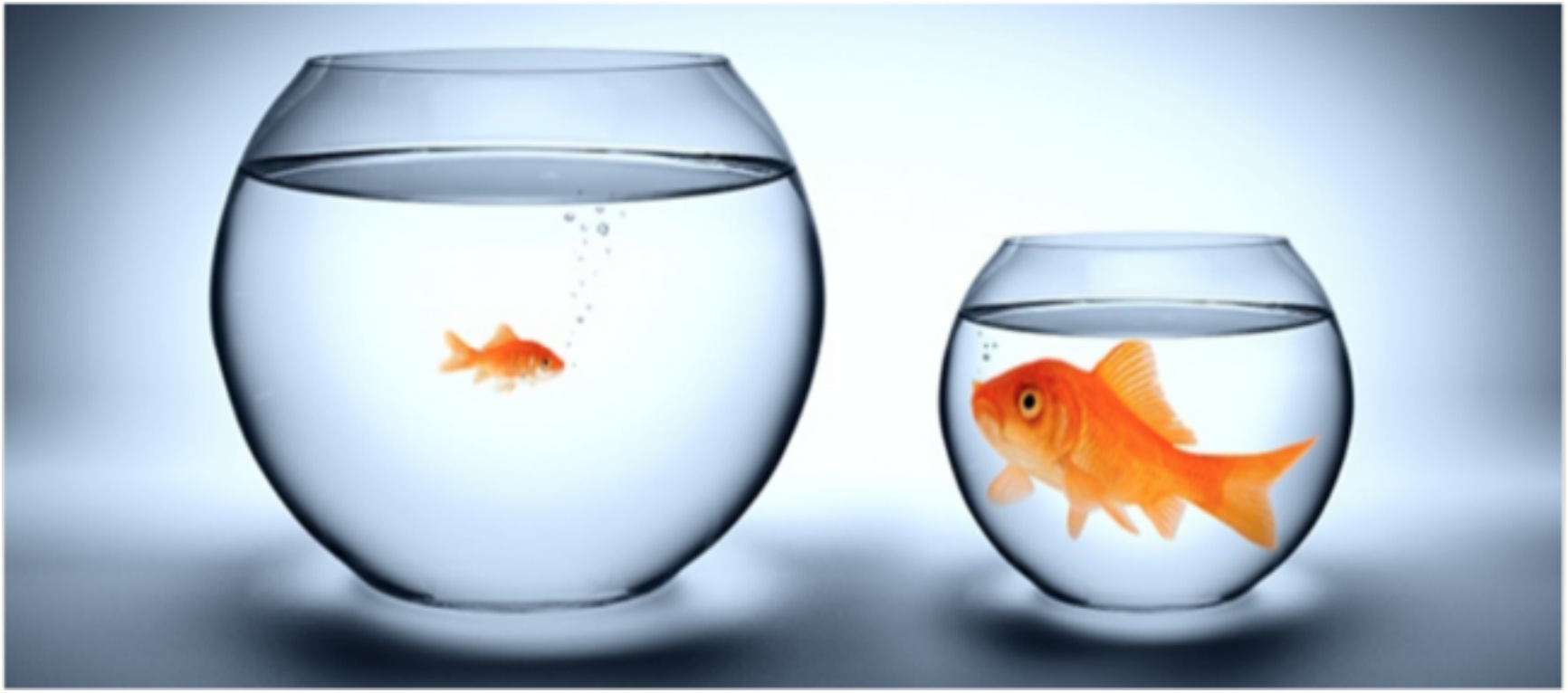
Developer Benefits

- Build once...(finally) run anywhere
  - A clean, safe, hygienic and portable runtime environment for your app.
  - No worries about missing dependencies, packages and other pain points during subsequent deployments.
  - Run each app in its own isolated container, so you can run various versions of libraries and other dependencies for each app without worrying
  - Automate testing, integration, packaging...anything you can script
  - Reduce/eliminate concerns about compatibility on different platforms, either your own or your customers.
  - Cheap, zero-penalty containers to deploy services? A VM without the overhead of a VM? Instant replay and reset of image snapshots?

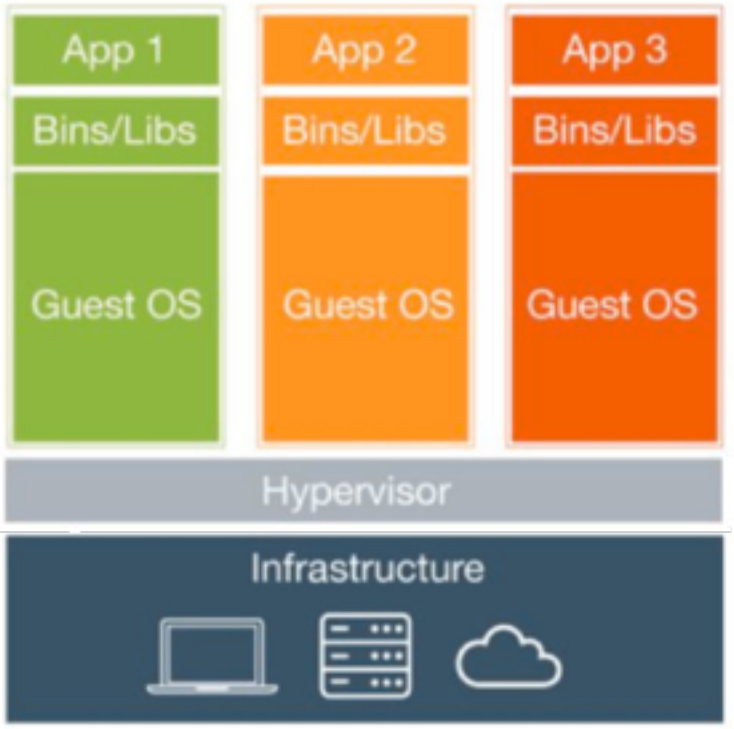
Operational Benefits

- Configure once...run anything
  - Make the entire lifecycle more efficient, consistent, and repeatable
  - Increase the quality of code produced by developers.
  - Eliminate inconsistencies between development, test, production, and customer environments
  - Support segregation of duties
  - Significantly improves the speed and reliability of continuous deployment and continuous integration systems
  - Because the containers are so lightweight, address significant performance, costs, deployment, and portability issues normally associated with VM

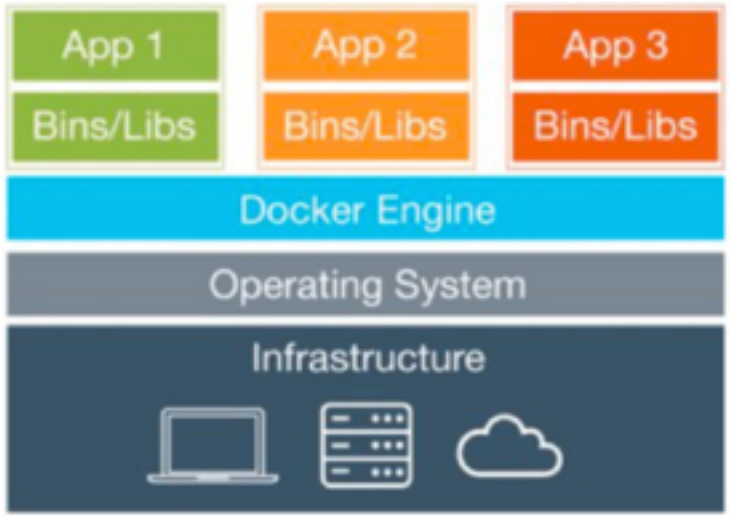
VMs vs. Containers



VMs vs. Containers



Virtual Machines

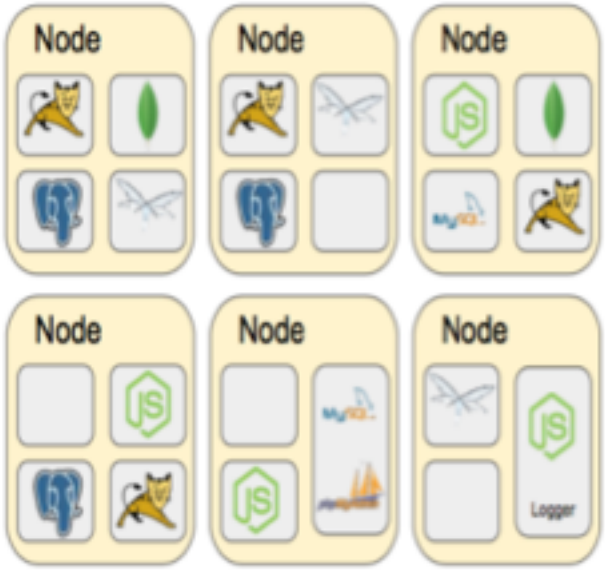


Containers

Container Advantages

- Portable
- Isolated
- Lighter footprint & overhead (vs VMs)
- Simplify DevOps practices
- Speed up Continuous Integration
- Empower Microservice architectures and adoption

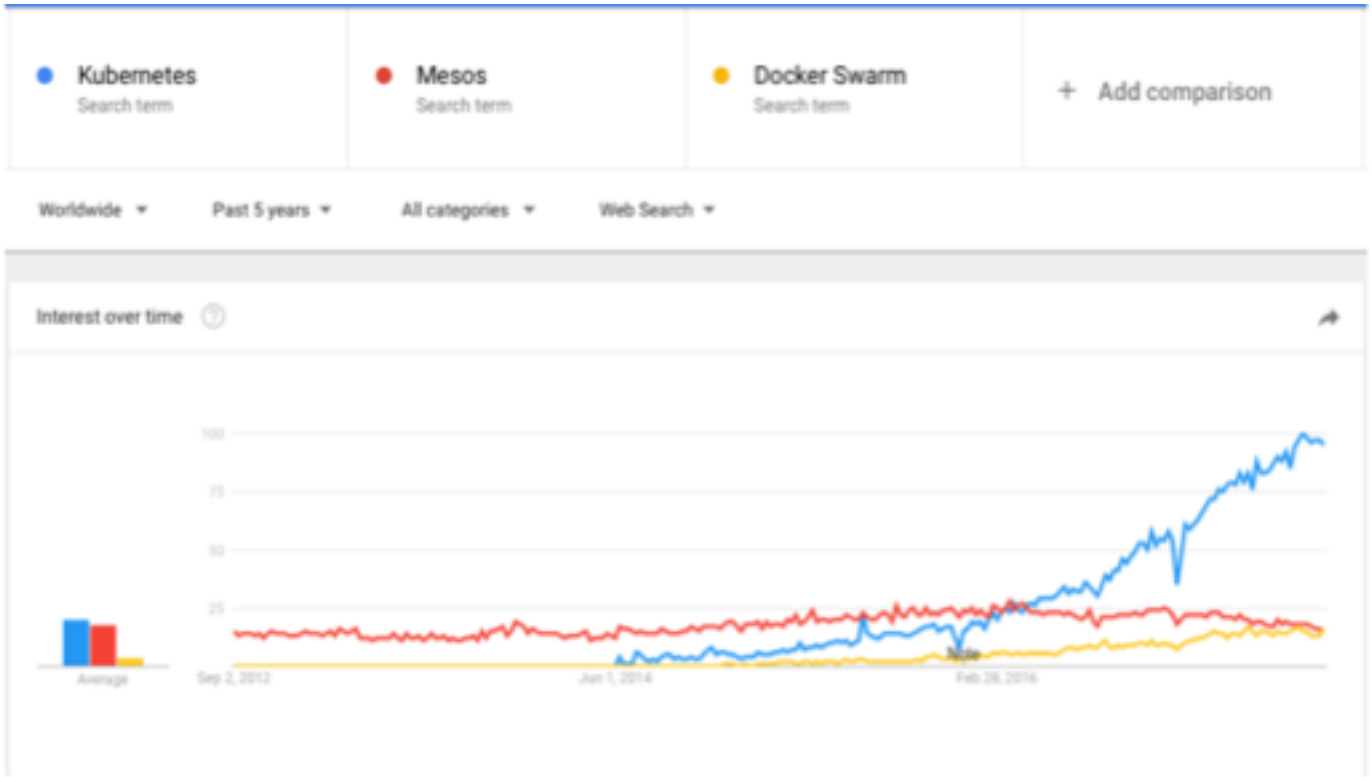
Challenges with multiple containers



- How to scale?
- Once I scale, where are they?
- How do my containers find each other?
- How should I manage port conflicts?
- What if a host fails?
- How to update them? Health checks?
- How will I track their logs?

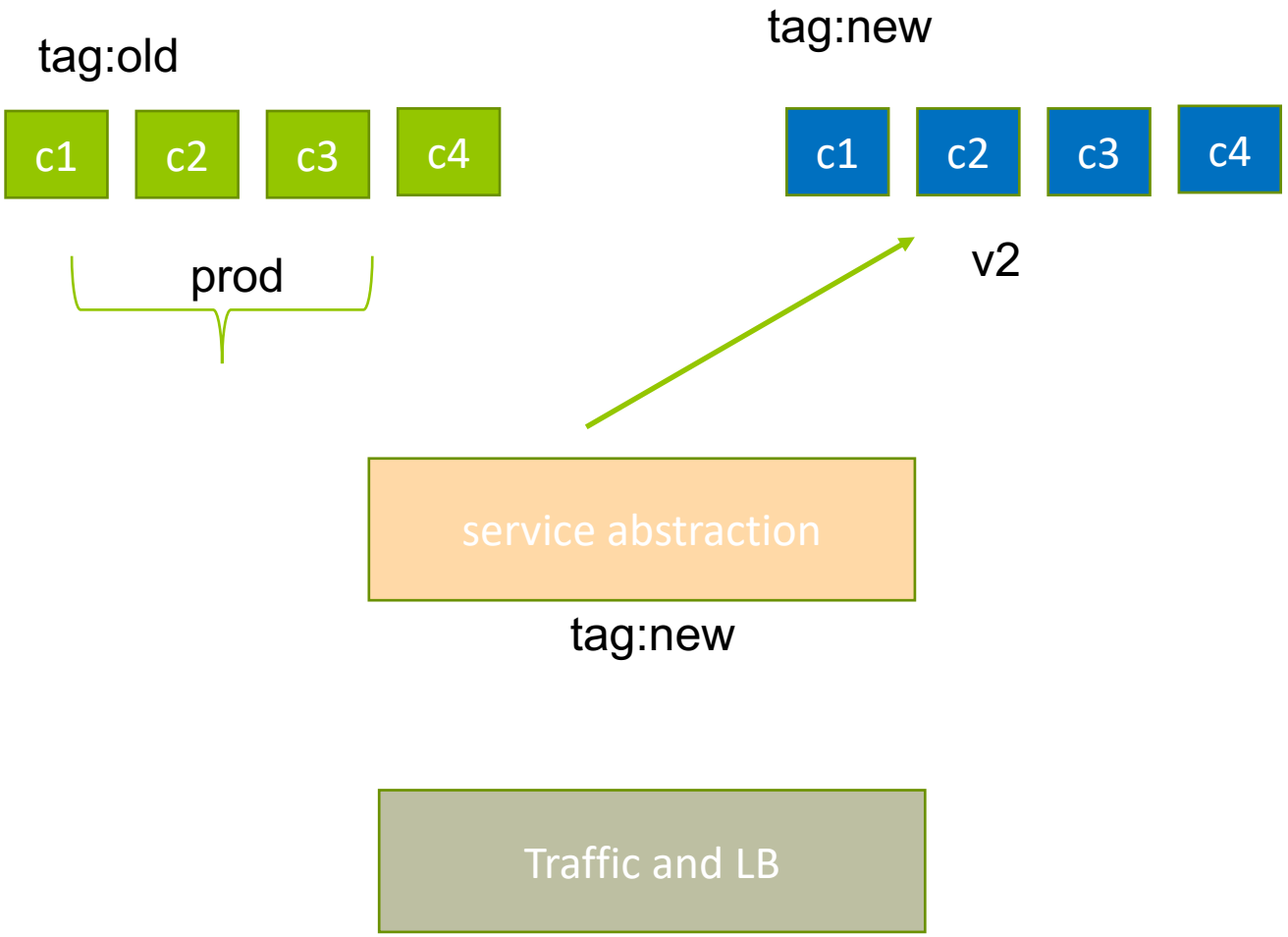
Container Orchestration

- The three most popular are: Kubernetes, Docker Swam & Mesos
- Kubernetes has become the unofficial standard



Orchestration Advantages

- Declarative approach to deploying applications
- Highly scalable, Highly available
- Self Healing
- Tonnes of APIs for container deployment, Jobs, ABAC, RBAC
- Cluster Management and Monitoring
- Automates application configuration through service discovery
- Maintains and tracks the global view of the cluster
- APIs for deployment workflows
  - Rolling updates, canary deploys, and blue-green deployments, scaling, auto-scaling


tag:old
c1
c2
c3
c4
prod
tag:new
c1
c2
c3
c4
v2
service abstraction
tag:new
Traffic and LB

Security in CI/CD

Module 10

DevOps Security Principles

DevOps is

incompatible with:

- InfoSec as a standalone silo of activity
- Security Checks at the end of the process
- “Security is someone else’s responsibility”

DevOps is

compatible with:

- Collaboration with Security specialists
- Security Checks built into the process
- “Security is everyone’s responsibility”

- Integrate Security Into Everything We Do
- Ensure Security of Everything We Use

Integrate Security Into Everything we do

- **Development Iteration Demo**
  - Include InfoSec as a team member OR invite InfoSec to every Sprint demo
- **Defect Tracking & Postmortems**
  - Treat security incidents like any other
  - Make Security a priority and use same tracking tools and processes
- **Shared Repositories**
  - Encrypted libraries, code authentication, secret management..

    Tool: KeyWhiz CredStash
- **Shared Services**
  - Authentication, logging security events
- **Deployment Pipeline**
  - Security scan on check-in (for both Infrastructure and application code)
  - Automated security test stages – Tool: Gauntlt
- **Production Telemetry**
  - Application security telemetry – logging password resets, login events
  - Operational logging – security group changes, config changes, web server errors

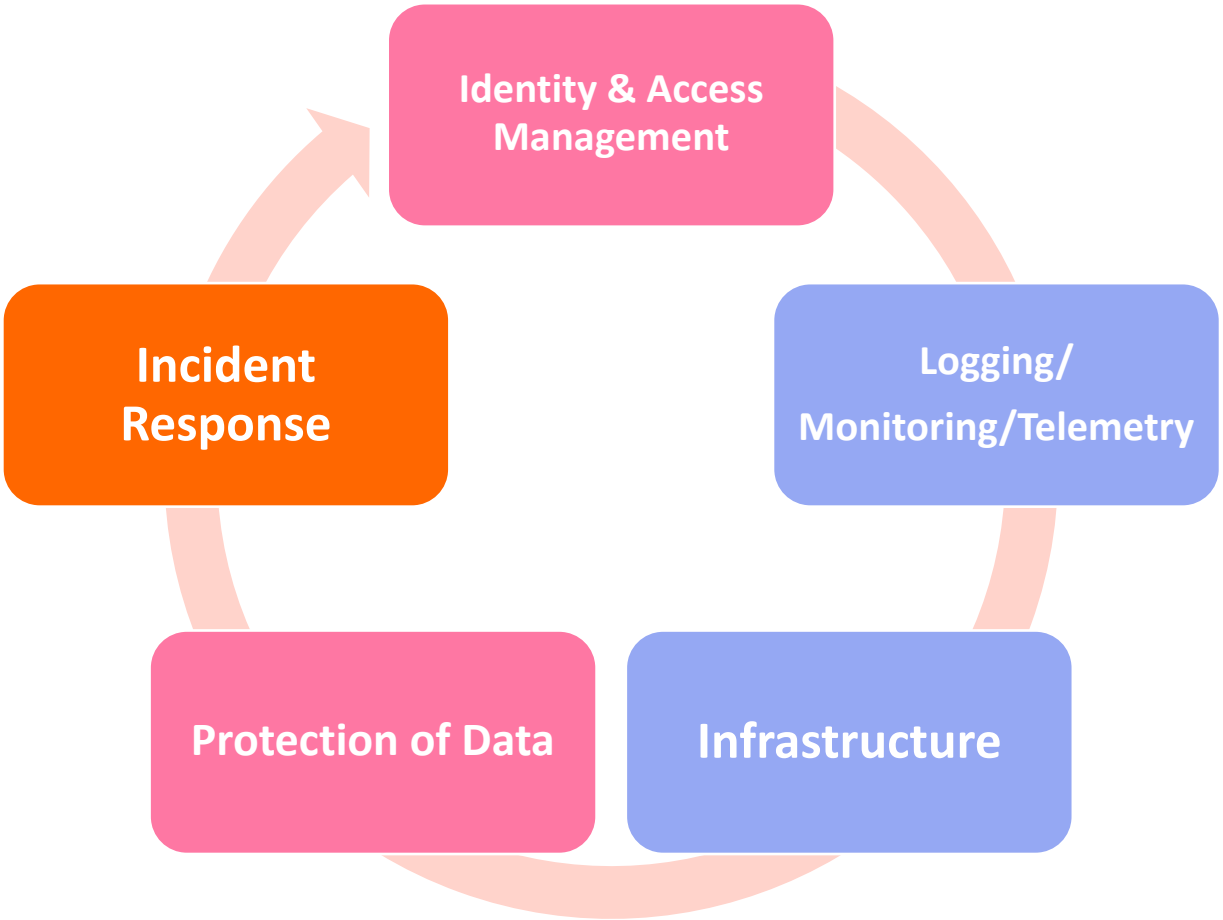
Ensure Security Of Everything we use

- **Each Application**
  - Security in CICD
  - Include misuse/abuse cases, sad paths in test
  - Securities of dependencies
  - Penetration tools: Metasploit, Nmap
- **Software Supply Chain**
  - Security check all open source components

** 2015 sonatype report – 605,000 open source projects servicing 17 billion downloads originating from 106000 orgs – 7.5% has security concerns, 66% are open for > 2 years

- **Environment**
  - Deploy only security-checked environments
  - Automate monitoring of all environments for deviation from deployed state
- **Deployment Pipeline**
  - Run each CI process in isolated containers/VMs
  - Protect CI credentials

CI/CD Security – Primary Priorities



After the priorities are implemented, sprints can be added to support and build further on security priorities.

Implementing Security in a CD Pipeline

- Define the **security strategy**
- **Automate security checks**, and remediation when possible – always providing feedback to developers
- **Promote the culture** of awareness, ownership, and continuous improvement of security outcomes
- **Focus on priorities first** – formulate overarching security policies into epics which can be executed incrementally
- Task **dedicated small teams** with establishing and increasing security velocity

Questions needed to be answered

- Can developers view sensitive environment variables for other projects?
- Are credentials checked in and stored in plaintext?
- Are some permissions too freely given so that anonymous developers could run bash scripts on all projects?
- Is the build orchestrator susceptible to easy attacks?
- Are developers able to delete a job from another project with ease?
- Are attackers checking in their code?
- What other services does CICD use and trust?
- Can I exploit the bugs of any build software CICD is deploying?

https://www.csoonline.com/article/2130877/the-biggest-data-breaches-of-the-21st-century.html

Use Telemetry to help identify problems

- Identify which **types of logging** and logging levels are required for each phase of the pipeline.
- **Articulate the value** of logging and its diagnostic value
- **Evaluate logging** management tools
- Interpret and **utilize log data**
- **Share** findings to access control and security

Expectations for Code Stability

Module 11

Code Stability

How robust the code is?

Benefits of Improved Code Stability

- Helps dev team to integrate new code easily
- Deliver prod features more frequently
- Helps to keep trunk in deployable state

  ALWAYS

Code Stability – How to improve?

- **Code should be stable in between Release and Production**
- **Manage test results with statistics**
  - Use proper metrics to compare test results and that in-turn provides a signal to measuring stability
- **Acceptance tests increase trunk stability**
  - Manage the Build Acceptance Test Suite
  - Impart functional stability
- **Project Manager's Responsibilities**
  - Provide proper release criteria, acceptance criteria and maintain balance between stability and features

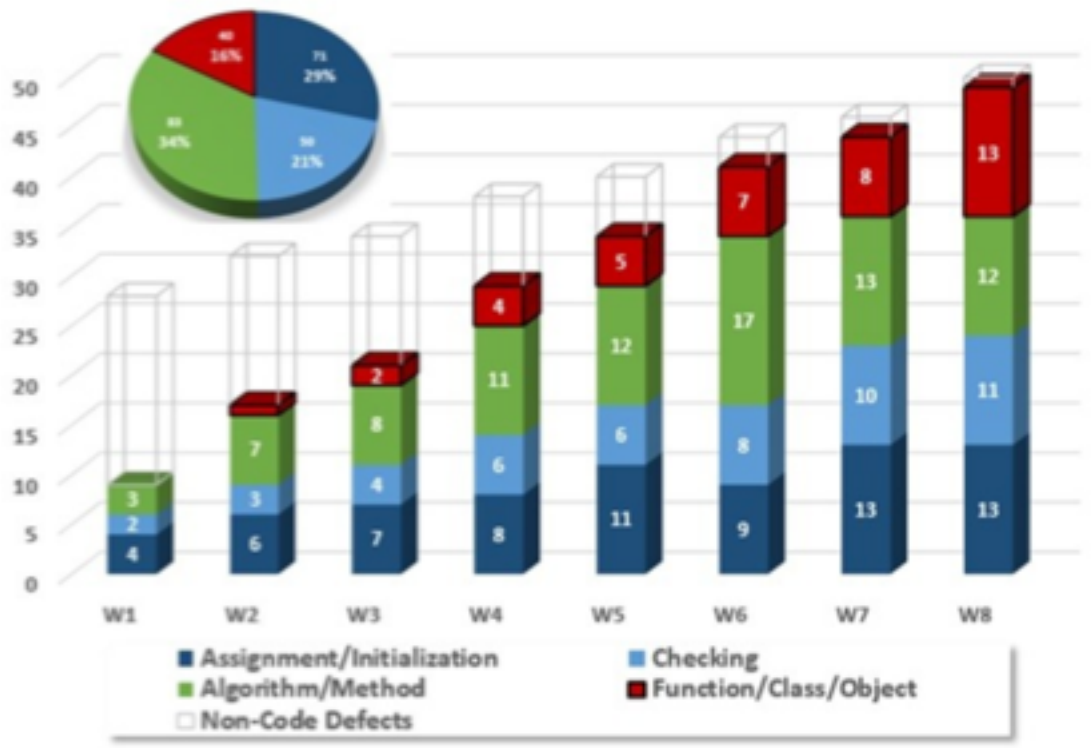
Stability Index

- The weighted defect raise and fix rates
- The proportion of the code-base that is changing
- The proportion of the code-base that is changing repeatedly

an aggregate measure of the rate at which your code is changing, combined with a measure of how well it is coping with that rate of change… some indication as to the level of risk associated with the project

no way does the Stability Index measure the quality of the product or code.

Measuring Stability



Unstable and Undesirable System

Assignment/Initialization: values in the code are assigned incorrectly

Checking:
The error is caused by missing ot incorrect validations of parameters or data in conditional statements

Algorithm/Method:
Calculation/algorithm is flawed or missing altogether

Function/Class/Object:
Function is flawed or missing and must be added

Lab: Configure Cloud Services

- **Exercise 01: Create App Service**
- **Exercise 02: Create Azure Database**

Exercise 01: Create App Service

- **Create App Service**

Exercise 02: Create Azure Database

- **Create Azure Database**

A Management & Planning Paradigm for Continuous Delivery

Module 12

12: Planning Process Paradigm Cultural Shift

- Plan for flexibility
- Optimize based on Process Intent

The Laser Jet **Planning** Example -

https://itrevolution.com/the-amazing-devops-transformation-of-the-hp-laserjet-firmware-team-gary-gruver/

Flexibility of Software Development

- Waterfall development limitations
- Embracing the flexibility of software
- Ability to make changes on demand
- Smaller timeframes increase speed to delivery of business value

❖ *Process Intent*

- The Cost of Commitment
- Development Capacity
- Planning Requirements

The Cost of Commitment

- Clarifying process intent
- Goal – reduce the cost of building and deploying software
- Estimate cost of adding features
- Balance between short and long term planning
- Long term commitments
- Long range = lower accuracy

Development Capacity

- Capacity is a Finite Resource
- Determining required Capacity
- Planning for built in Capacity

Determining Development Capacity

- Using Metrics to calculate Capacity
- Avoid assumptions, rely on data
- Business Value over Accuracy

Planning Requirements

- Viewing Priorities across teams
- Avoiding team sub-optimization
- Business Priorities always come first
- Prioritizing business initiatives
- Prepare for inevitable errors
- Use data to support Business Decisions

Accuracy of Long-Range Planning

- High range accuracy is rare
- Lot of assumptions
- Discoveries on the way
- Plan for short term & build for long term

❖ *Pointers for Planning*

- It’s a “MUST”, or it will never happen…
- Commitments and deadlines
- Planning around marketing initiatives
- Balancing capacity with long-range planning

Commitments and Deadlines

- Commit for reducing technical debt along with feature release
- Separate commitment requirements for different process|features|technical debt reduction
- Develop a light-weight planning process

Balancing Capacity with Long-Range Planning

- Higher capacity for developing of new features
- Minimized the Requirements inventory
- Broke the initiatives into detailed user stories
- Avoid locking in Capacity
- Keep long-range commitments to 50%
- Delivered a Prioritized Backlog

Lab: Continuous Delivery

- **Exercise 01: Create a Release Pipeline**
- **Exercise 02: Create a new Release**
- **Exercise 03: Browse Service Reservations on Azure**

Lab guide: Page 20 to 25

Exercise 01: Create Release Pipeline

- **Create a Release Definition Based on Build output**

Exercise 01: Create Release Pipeline (Continued)

Exercise 02: Create a new Release

- Add Azure SQL Connection to Web.Release.Config
- Release solution to QA

Exercise 02: Create a new Release (Continued)

Exercise 03: Browse Service Reservations on Azure

- **Open Service Reservations in the Browser**

Setting Goals and Getting Started

Module 14

14 Setting Goals and Getting Started

- Where do we start the transformation?
- Activity Based vs Cycle-Time
- Scaling Agile & DevOps Principles

❖ *Where do We Start the Transformation?*

- Don’t “Do” Agile as a task, be and remain Agile
- Clear Objectives
  - Shared and agreed upon between executive, development and operation level
- Collaboration
  - Org wise collaboration

“Being” Agile

- Never set out to “do” Agile
  - Just setting up some process, stand ups and user stories just make Agile
  - It’s a cultural transformation
- Agile development is always evolving
  - Continuous evolving and improvement
  - No one recipe works for everyone

Clear Objectives Guide the Journey

- Developing business objectives specific to your organization
- A continuous improvement process

Collaboration is Essential

- Clearly defining organization-wide cooperation
- Making sure everybody is on board

❖ Activity-Based vs Cycle-Time

- Defining cost drivers
  - What's driving activity-based cost
  - Identify and avoid unnecessary task and bottlenecks
- Define Cycle-time
  - These drivers identify waste in the development process
  - Use analysis to expand capacity

❖ Activity-Based vs Cycle-Time

- Prioritize objectives by business value
  - Engage business stakeholders
- Plan for continuous improvement
  - Track metrics
  - Get Feedback

❖ Scaling DevOps & Agile Principles

- Planning, Integrating, and qualifying code
  - Maintaining your code for consistent release
  - Eliminate Unnecessary Planning
  - Modify processes for flexible response to customer feedback
- Large organizations vs. small organizations
  - Writing code is similar but planning and integration may be different.

The Key to Getting Started

One size doesn't fit all

- Strive for improvement, not perfection
- Setting achievable goals
- Quantifiable advancements

Key Pointers to Getting Started

Cultural Aspect

- Use the value stream mapping technique to identify current process and scope of improvement
- Start small, focus on value
- Prioritize actions that show improvement
- Begin with clear business objectives – Choose the right objectives
- Choose the right team to lead the process of change management
- Have proper communication - the key to Transformation
- Get leadership to support the priorities
- Track Improvement, keep it visible
- Ensure goals are quantifiable

Key Pointers to Getting Started

Technology Aspect (if possible and relevant)

- Get the right team for technology changes
- Train people on advanced technologies
- Get better test coverage (all types of test)
- Resolve environmental issues
  - Use Containers
  - Use Configuration Management
- Use automated CICD tools

Questions?

Feedback Email

If you haven't already, you will receive an email similar to the one here asking for your feedback on the course.

This Evaluation will ask your feedback on many aspects of your training, including the course materials, instructor facilitation and labs where applicable.

We appreciate you completing this survey to help us continue to better our content and deliveries in order to serve you better.

Thank You for Choosing ASPE as Your Learning Partner

ASPE
Learn. Transform. Succeed.

Thank you for choosing ASPE as your learning partner!
Please tell us about your experience.

<Your Name>

Thank you for attending the with ASPE. Your feedback is very important to us. Please complete our online course evaluation by clicking on the 'Complete Evaluation' button below.

If you are unable to see the button below, please copy and paste this link into your browser:
http://aspetraining.com/student-feedback

Complete Evaluation

Working toward certification with PMI, Scrum Alliance, or the IIBA?
Need to report your Continuing Education Units? You can access your own Certificate of Completion for download as a PDF after completing the survey.

For More Information Contact:
customerservice@aspeinc.com
919.816.1750

ASPE | 2000 Regency Parkway, Suite 335, Cary, NC, 27518 | 919-816-1750

Appendix

DevOps & Agile Principles for CD at Scale

06: DevOps & Agile Principles at Scale

- ***Improving the Developer Feedback Loop***
- ***Source and Version Control***
- ***Reducing Release Requirements***
- ***Increase Capacity***
- ***Automate Deployment***
- ***Prevent Duplicated Work in Branches***

Improve the Developer Feedback Loop

- Don’t beat the developer up over mistakes
- Developers improve from consistent feedback
- Cultural Shift for Development & Operations

Source and Version Control

- Allows Multiple Developers to work simultaneously
- Provides Branching Capabilities
- Allows work to begin on SP1 while stabilizing Release Candidate 1
- Central Point of Authority for current version
- Source for Automated Builds

Benefits of Version Control

- Use of common repositories and processes
- Reconstruction of the business from source code repository
- Enable easy rollbacks to minimize risk of changes
- Back Up/Restore

"Enable the reconstruction of the business from nothing but a source code repository, an application data backup, and bare metal resources."

- Adam Jacob

Benefits of Version Control (continued)

Disaster recovery benefits from Devops practices

- Reuse deployment automation to rebuild application in disaster scenario
- Rebuilding the application from a single source in version control
- Automating and limiting dependencies
- Storing data and configuration in source control
- Practicing and anticipating failure

Test Driven Development

- AKA Test “First” Development
- Create Unit Test before code is written
- Ensures all code has an associated test
- If there is a defect in the solution there is a missing or incomplete test
- Create a test for the defect before fixing the defect
- Build quality in early – don’t wait until release

Reducing Release Requirements

- **Finding & fixing Code Base Defects**
- **Automate Daily Regression Testing**
- **Add Code without breaking existing functionality**

Finding & Fixing Code Base Defects

- Bringing the Code Base to Release Quality
- Continuous Deployment Techniques
- Multiple daily Check-ins

Automate Daily Regression Testing

- Automating daily testing
- Decreasing production time
- Working towards Frequent Releases

Add Code Without Breaking Existing Functionality

- Localize broken Code
- Isolate Deployment Issues
- Decrease Manual testing

Increase Capacity

- Repeatable test, build, deploy process
- Detecting Inefficiencies
- Designing a deployment Pipeline

Repeatable Test, Build, Deploy Process

- Development and Operations working together
- Aligning common objectives
- Applying DevOps principles at Scale

Detecting Inefficiencies

- **Providing Immediate Feedback**
- **Feedback weeks later has limited value**
- **Delivering Value to the Customer**

Designing a Deployment Pipeline

- Designing a pipeline specified to your organization
- If it's labor intensive, it's probably wrong
- Structured, repeatable processes

Automated Deployment

- Deploying to multiple servers
- Effective Deployment Process
- Errors: code or deployment related?

Deploying to Multiple Servers

- **Organization Size Matters**
- **Removing duplication of Work**

Debugging the Deployment Process

- Isolating deployment issues
- Finding code defects in a complex system

Errors: Code or Deployment Related?

- System Test Failures
- Testing deployment before testing code

Lab: Be Agile With Team Services

- Exercise 1: Create a User Story
- Exercise 2: Define Acceptance Criteria
- Exercise 3: Define TDD Test Tasks
- Exercise 4: Define TDD Implementation Tasks

Lab guide: Page 40 to 51

Exercise 1: Create a User Story

- Create a New User Story in Team Project
- Add Story Point Estimation
- Assign User Story

Exercise 1: Create a User Story (Continued)

Exercise 2A: Define Acceptance Criteria

- **Add a basic description**
- **Add Acceptance Criteria in Gherkin format**
  - **Add Customer Story Acceptance Criteria**
  - **Add Service Provider Story Acceptance Criteria**

Exercise 2A: Define Acceptance Criteria (Continued)

Exercise 2B: Define Acceptance Criteria

- **Add Acceptance Criteria in Gherkin format**
  - **Add Service Provider Story Acceptance Criteria**

Exercise 2B: Define Acceptance Criteria (Continued)

Exercise 3: Define TDD Test Tasks

- Create a new Developer Task (Unit Test)
- Estimate task complexity
- Assign the Task

Exercise 4: Define TDD Implementation Tasks

- Create a new Developer Task (Method)
- Estimate task complexity
- Assign the Task