What is Continuous Integration

And how do I get there
Related Workshops

• Introduction to DevOps
  – Transform your Organization with DevOps Concepts

• DevOps Implementation Boot Camp
  – Comprehensive literacy on transitioning to Agile-Driven DevOps Workshop
  – Teach your teams to manage infrastructure, operations, and production work the Agile way.

• Test-Driven Development Workshop
  – Learn to implement Test-Driven Development methods by incorporating unit testing, design, refactoring and frameworks into your workflow.
Related Workshops

• Continuous Integration Boot Camp
  – A fast-paced, practical approach to CI in the real world - skills, tools, processes, and people.

• Continuous Delivery Workshop
  – Learn to implement Continuous Delivery strategies into your current application workflow.

• Hands-On Agile Engineering
  – TDD, Continuous Integration, Refactoring, and Emergent Design for Agile technical excellence.
Continuous Integration Bootcamp

In this course you will learn how to:

• Set up and navigate a CI environment that enables automated testing & automated builds
• Execute releases in test environments prior to product delivery
• Identify the most efficient CI tools for quick release & reliable maintenance of solutions
• Integrate Continuous Integration methods & techniques into current workflow for a SCM environment
• Deliver value from continuous integration in your own organization
• Examine real world examples of continuous integration benefits

• http://www.techtowntraining.com/courses/continuous-integration-bootcamp/
Agenda

• The Problem with Delivering Software
• Becoming Agile
• Automated Testing
• Continuous Integration
• Configuration Management
• Infrastructure as Code
• Nonfunctional Requirements
• Continuous Delivery
The problem with delivering software

- Changing Priorities
- Project Visibility
- Time to market
Changing Priorities

• Allow changes throughout the project
• Freeze changes in sprint
• Prioritize changes
• Incremental planning
Project Visibility

- Kanban Board
- Test Reporting
- Sprint Burndown
- Epic and Release Burndown
- Key Agile Metrics
Time to market

- Mean Time to Change
- Mean Time to Recovery
Becoming Agile

- Agile Manifesto
- Agile vs Waterfall
- The Agile Team
- Agile Practices
We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

- **Individuals and interactions** over processes and tools
- **Working software** over comprehensive documentation
- **Customer collaboration** over contract negotiation
- **Responding to change** over following a plan

That is, while there is value in the items on the right, we value the items on the left more.
DEFINING AGILE

Why does Agile work?
The Agile Team

Servant Leaders
- Product Owner
- Scrum Master
- Agile PM

Development Team
- QA
- BA
- Developer
- UX
- Developer

ASPE TRAINING
A division of Fortis College
Agile Practices

• Agile Planning
• Development Sprints
• Test Driven Development
• Continuous Inspection
Continuous Inspection

- Core concepts
- Developer push back
- 10 key points
Core Concepts

• Automated code scans
• Standard Naming Conventions
• Variable Declarations
• Scan on commit
Developer Push back

• Developers tend to push back on action plans generated from punctual audits, because they:
  • Are generated outside the team, and are seen as a new constraint in daily work
  • Are subjective; findings rely on the judgement of the auditors rather than on objective measures
  • Miss contextual and historical information, and are therefore seen as irrelevant
Developer push back

• Are invalidated by on-going changes and quickly become out-dated
• Do not involve developers and other stakeholders in the review & audit process
• Intervene too late in the process; by the time a feature is audited, developers need to “relearn” the code to address a finding
10 Key Concepts

• All stakeholders in the development process – not just developers or managers - must have ready access to meaningful data about software quality. Managing software quality must be everyone’s concern from the beginning of development, but is the development team's ultimate responsibility.

• Software Quality must be part of the development process, meaning that meeting quality standards is one of the hard requirements to be able to declare development complete.

• Software Quality requirements must be objective and not require a subjective pass / fail decision. As much as possible, software quality requirements must be common to all software products, regardless of their specifics.
10 Key Concepts

- Software Quality data must be up to date, i.e. measured on the very latest version of the code.
- Software products must be continuously inspected, so that errors are found quickly, when they are easy to correct.
- Developers must be able to spot new quality flaws as soon as they are introduced, i.e. within the IDE as they write code, similar to how spell checkers highlight misspellings. Whether through push or pull, stakeholders must be alerted when new quality flaws are injected, whether that's by sending email, breaking the build or by other methods. Injection of new issues must be tracked, enabling teams to make quick, informed decisions about quality.
- Software quality data must be available both as absolute (on all code) and differential (new code only) values so that the development team can be in full control of the incoming flow of issues. All new issues and existing critical issues must be assigned a clear path and timeline for resolution.
10 Key Concepts

• Software Quality data must be up to date, i.e. measured on the very latest version of the code.

• Software products must be continuously inspected, so that errors are found quickly, when they are easy to correct.

• Developers must be able to spot new quality flaws as soon as they are introduced, i.e. within the IDE as they write code, similar to how spell checkers highlight misspellings. Whether through push or pull, stakeholders must be alerted when new quality flaws are injected, whether that's by sending email, breaking the build or by other methods. Injection of new issues must be tracked, enabling teams to make quick, informed decisions about quality.

• Software quality data must be available both as absolute (on all code) and differential (new code only) values so that the development team can be in full control of the incoming flow of issues. All new issues and existing critical issues must be assigned a clear path and timeline for resolution.
Configuration Management

• Core concepts
• Version Control
• Build Automation
Core concepts

- Configuration details in version control
- Lab environments that mirrors production
- Test / Staging environment that mirrors production
- Automated configuration
- Never make manual changes in production
Version Control

Version Control is a system that records changes to files, so that one can recall specific versions later.

**Benefits**

- Backup/Restore
- Team Collaboration
- Accountability
- Audit trail
- Sync across multiple work stations

**Terms**

- Repository: Stores documents
- Server: Main file store
- Client: user work station with local copy of files
- Trunk: Master location of files
- Head: Latest version of files
Version Control

Questions to consider:

• Can new developers setup their environment from one source?
• Can the operations or testing team duplicate the product from the repo?
• Can a business user compile the product to demonstrate to a user?

Failure to place everything in version control can lead to reduced quality, agility and efficiency.

• New (and current) developers wasting time setting up environments
• Reduced testing coverage due to challenges creating the artifacts
• Loss of opportunities due to lack of easily demonstrable value
Version Control

Small, Frequent Commits

- Less risk
- Easier to rollback
- Easier to review
- Less merge conflicts
- More frequent test coverage
- Less disruptive to other outstanding branches in development
Version Control

Is your team doing Continuous Integration?

• Do developers check into trunk every day?
• Does every check-in trigger testing (unit and regression) and an automated build?
• If the build is broken or tests fail, is the problem resolved in a few minutes?
Build Tool Automation

1. It should be possible for a developer to easily reproduce a build on his or her local machine.

2. It is especially beneficial if anyone can trigger a build from a common machine. i.e. Product Owner can trigger builds independently without developer help.
Automated 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
Configuration management tools:

(puppet labs, ANSIBLE, SALTSTACK, CHEF)

(we will discuss these more later)
Continuous Integration

- Commit to Source Control
- Automated Build
- Automated Tests
- Gated Check-In
- Automatic Bug Item Creation and Assignment
The deployment pipeline

- Commit Stage:
  - Compile
  - Unit Test
  - Analysis
  - Build Installers

- Automated Acceptance Test

- Automated Capacity Testing

- Manual Testing:
  - Showcases
  - Exploratory testing

- Release

Source: Continuous Delivery, Humble & Farley
The deployment pipeline is enabled by the alignment of testing and automation with the product life cycle

- **Testing** – The deployment pipeline relies on testing being an integral part of the product.
- **Automation** – Among the many other areas of IT workflow in which automation serves a transformative role, one of the most important is testing.
The deployment pipeline concept

Source: Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation
Automated Testing

- Test Driven Development
- Acceptance Test Driven Development
- Automated Unit and Acceptance Tests
- Gated Check-in
Test Driven Development

• Write a failing test
• Write just enough code to pass the test
• Refactor as needed
Why Do We Unit Test

• Reduce bugs in new futures
• Reduce bugs in existing features
• Provides documentation as to intent
• Reduces Fear, Uncertainty, and Doubt (FUD)
  – Cheaper to make future changes
  – Allows for refactoring
• Helps refine design by creating “thinking” time
  – Did we think of everything?
  – Does the code respond properly in all scenarios
• Insulates code from “the next guy”
Why We Test Early

What Should We Test

- Increase in:
  - Cost to develop and maintain
  - Execution time
  - Possibility of false negatives

But also in:
- Coverage
- End-user (business) relevance

Number of tests

User interface tests

Service / API layer tests

Unit tests

What Too Often Happens

- User interface tests
- Service / API layer tests
- Unit tests

What Makes A Good Unit Test

• Automated
• Test the required behavior not the implementation
• Isolates the “system under test”
  – Small units
  – Avoid making these Integration Tests
• Executed frequently
• Follows all same code conventions as code under test (maintainable & Readable)
• Fast!
Infrastructure as Code

- Reproducible Infrastructure
- Consistent configuration
- Reduced deployment time
- Reduced Mean Time to Recovery
- All solution artifacts in version control
IaaS and Infrastructure as code

The ideal (future) state:

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

– Adam Jacob, CTO at Chef
Infrastructure as code

Accepted best practice ‘Solution’
• Migrate infrastructure to reproducible, reliable methods.
• Iterate until its achieved
Infrastructure as code allows us to...

- Apply many of the processes and tools that work well in development to operations
- Use common version control repositories and processes for both software and infrastructure
- Reduce the need for completely different departments and job descriptions between Dev and Ops
- Give greater access and visibility to developers and operations regarding infrastructure dependencies for application products
- Allow development projects to accommodate much earlier testing, staging, simulation of final production environments, etc.
Nonfunctional Requirements

- Be sure to track beyond developer tasks
  - Infrastructure
  - Security
  - Auditing
  - Compliance
  - Testing
Continuous Delivery

• Be ready and able to deploy
  – Any Version of your solution
  – To any platform
  – At any time
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.
Moving to DevOps
Open Discussion, Q and A
Class Feedback Survey

Please take a few moments to fill out the class evaluation survey:

www.metricsthatmatter.com/aspe

You will receive your certificate of completion after filling this out.
Thank you!
Automation

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Second</td>
<td>1 Day</td>
<td>2 Hours</td>
<td>30 Minutes</td>
<td>4 Minutes</td>
<td>1 Minute</td>
<td>5 Seconds</td>
</tr>
<tr>
<td>5 Seconds</td>
<td>5 Days</td>
<td>12 Hours</td>
<td>2 Hours</td>
<td>21 Minutes</td>
<td>5 Minutes</td>
<td>25 Seconds</td>
</tr>
<tr>
<td>30 Seconds</td>
<td>4 Weeks</td>
<td>3 Days</td>
<td>12 Hours</td>
<td>2 Hours</td>
<td>30 Minutes</td>
<td>2 Minutes</td>
</tr>
<tr>
<td>1 Minute</td>
<td>6 Days</td>
<td>1 Day</td>
<td>4 Hours</td>
<td>1 Hour</td>
<td>5 Minutes</td>
<td></td>
</tr>
<tr>
<td>5 Minutes</td>
<td>9 Months</td>
<td>6 Days</td>
<td>21 Hours</td>
<td>5 Hours</td>
<td>25 Minutes</td>
<td></td>
</tr>
<tr>
<td>30 Minutes</td>
<td>6 Months</td>
<td>5 Weeks</td>
<td>5 Days</td>
<td>1 Day</td>
<td>2 Hours</td>
<td></td>
</tr>
<tr>
<td>1 Hour</td>
<td>10 Months</td>
<td>2 Months</td>
<td>10 Days</td>
<td>2 Days</td>
<td>5 Hours</td>
<td></td>
</tr>
<tr>
<td>6 Hours</td>
<td>2 Months</td>
<td>2 Weeks</td>
<td>1 Day</td>
<td>8 Weeks</td>
<td>5 Days</td>
<td></td>
</tr>
<tr>
<td>1 Day</td>
<td>2 Months</td>
<td>2 Weeks</td>
<td>1 Day</td>
<td>8 Weeks</td>
<td>5 Days</td>
<td></td>
</tr>
</tbody>
</table>