DEVOPS System Analysis & Design Course
Sharif University of Technology
MohammadAmin Fazli
“DevOps is development and operations collaboration”

“DevOps is using automation”

“DevOps is small deployments”

“DevOps is feature switches”

“DevOps is treating your infrastructure as code”

“Kanban for Ops?”
Our Definition of DevOps

"DevOps is the union of people, processes and products to enable continuous delivery of value to end users."

- Donovan Brown, Microsoft DevOps PM

DevOps encompasses

• Culture
• Measurement
• Automation
• Collaboration
DevOps – The (semi) holistic view

- Development
  - Requirements, version control, test case management, bug tracking, etc

- Testing
  - Unit, integration, exploratory, load, automated UI, performance, etc

- Deployment
  - Environment definition, provisioning and configuration
  - Application configuration and deployment
  - Approval workflows and automation

- Monitoring
  - Application Performance Monitoring
  - Alerts and notifications
What DevOps is NOT

• It is not a product
• It is not a specification
• It is not centralized
• It is not trademarked

“You cannot buy DevOps and install it. DevOps is not just automation or infrastructure as code. DevOps is people following a process enabled by products to deliver value to our end users.”

- Donovan Brown
What’s driving DevOps?

The agile Methodologies are accelerating the construction process

Current ITLM/ITSM “best practices” made the release and operate processes reliable, but not agile

Disconnects between Development and Operations increase mistakes and MTTR when issues occur

Determine next set of investments based on learnings

The agile Methodologies are accelerating the construction process

BACKLOG

Development

Production

Collaboration
<table>
<thead>
<tr>
<th>DevOps Demand Drivers</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Improve quality and performance of the applications</td>
<td>42%</td>
</tr>
<tr>
<td>Improve the end customer experience</td>
<td>34%</td>
</tr>
<tr>
<td>A greater need for simultaneous deployment across different platforms</td>
<td>29%</td>
</tr>
<tr>
<td>Increasing use of mobile devices</td>
<td>28%</td>
</tr>
<tr>
<td>Pressures to release applications more quickly</td>
<td>27%</td>
</tr>
<tr>
<td>Need for greater collaboration between development and operations teams</td>
<td>25%</td>
</tr>
<tr>
<td>Need to develop and deploy cloud-based applications</td>
<td>19%</td>
</tr>
<tr>
<td>Increasingly complex IT infrastructure</td>
<td>14%</td>
</tr>
<tr>
<td>Need to reduce IT costs</td>
<td>5%</td>
</tr>
</tbody>
</table>
Value of DevOps

• DevOps bridges the traditional divide allowing teams to produce high quality releases at increasing cadence

• DevOps goals span the entire delivery pipeline
Shorter Cycles & Higher Quality

- Faster time to market
- Lower failure rates
- Shortened lead time
- Faster MTTR\(^4\)
  - Mean Time To Realize, Recover, Repair, Remediate
DevOps Report 2014

- Companies with high-performing IT organizations are twice as likely to exceed their profitability, market share and productivity goals.
- IT performance improves with DevOps maturity, and strongly correlates with well-known DevOps practices.
- **Culture matters. The cultural practices of DevOps are predictive of organizational performance.**
- Job satisfaction is the No. 1 predictor of performance.

DevOps Report 2015

- High-performing IT organizations experience 60X fewer failures and recover from failure 168X faster than their lower-performing peers. They also deploy 30X more frequently with 200X shorter lead times.
- Lean management and continuous delivery practices create the conditions for delivering value faster, sustainably.
- High performance is achievable no matter if your apps are greenfield, brownfield or legacy.
- DevOps initiatives launched solely by C-level executives or from the grassroots are less likely to succeed.
- IT managers play a critical role in promoting diversity and limiting burnout.

DevOps Benefits

- Software/services made available across more platforms: 31% (Already Seen) - 60% (Expect to See)
- Increased frequency of deployments: 46% - 44%
- Increased collaboration between departments: 39% - 50%
- A reduction in time spent fixing and maintaining applications: 34% - 54%
- Increased numbers of customers using our software/services: 39% - 49%
- Improved quality and performance of our applications: 36% - 51%
- New software/services that would otherwise not be possible: 32% - 53%
- Reduced time-to-market: 34% - 50%
- An increase in revenue: 34% - 50%
- A reduction in spend on development, testing or operations: 31% - 51%
- Fewer employees working on developing and deploying our software/services: 30% - 50%

Figure 1.
What benefits have you seen or do you anticipate seeing from implementing DevOps in your organization? Total: 1,256 respondents who already have or plan to implement DevOps.

History of DevOps

**Agile Conference 2008**
Patrick Debois and Andrew Shafer discuss “Agile Infrastructure”

**October 2009**
Patrick Debois starts “DevOpsDays” in Ghent, Belgium

**Velocity 2009**
John Allspaw and Paul Hammond present “10 Deploys per Day: Dev and Ops Cooperation at Flickr”

**2010-**
DevOpsDays spread globally OSS Tools like Chef, Puppet, Vagrant, LogStash, Jenkins etc. gain popularity

**March 2011**
Cameron Haight of Gartner predicts explosion of DevOps in Global 2000 companies
John Allspaw’s visual – Slow delivery cycles
John Allspaw’s visual – Fast delivery cycles
7 DevOps Practices

Configuration Management
Release Management
Continuous Integration
Continuous Deployment
Infrastructure as Code
Application Performance Monitoring
Test Automation

Sam Guckenheimer
(http://devops.com/2015/12/03/11626/)
7 DevOps Habits

- Team Autonomy and Enterprise Alignment
- Rigorous Management of Technical Debt
- Focus on Flow of Customer Value
- Hypothesis Driven Development
- Evidence Gathered in Production
- Manage Infrastructure as a Flexible Resource
- Live Site Culture

Sam Guckenheimer
(http://devops.com/2015/12/03/11626/)
DevOps metrics

<table>
<thead>
<tr>
<th>Deployment frequency</th>
<th>Change lead time</th>
<th>Change fail rate</th>
<th>Mean time to detect &amp; repair</th>
</tr>
</thead>
</table>

**Agility** performance indicators

**Reliability** performance indicators
Revisiting the Agile Manifesto

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.
Principles behind the Agile Manifesto

We follow these principles:
Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.

Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.

Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

Business people and developers must work together daily throughout the project.

Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.

The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

Working software is the primary measure of progress.

Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

Continuous attention to technical excellence and good design enhances agility.

Simplicity--the art of maximizing the amount of work not done--is essential.

The best architectures, requirements, and designs emerge from self-organizing teams.

At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.
We follow these principles:

Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.

Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.

Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

Business people and developers must work together daily throughout the project.

Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.

The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

Working software is the primary measure of progress.

Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

Continuous attention to technical excellence and good design enhances agility.

Simplicity—the art of maximizing the amount of work not done—is essential.

The best architectures, requirements, and designs emerge from self-organizing teams.

At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.
Principles behind the Agile Manifesto

Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.

Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

Business people and developers must work together daily throughout the project.

Working software is the primary measure of progress.
Agile Operations

• Source Control
• Small, frequent releases
• Automated testing
• Continuous Integration
• Continuous Deployment
• Peer Review
• Immutable Infrastructure
Benefits of Small Releases

- Lower risk
- Faster feedback
- More Confidence

John Allspaw’s visualization of slow and fast delivery cycles
What is Version/Source Control?

- The management of changes to documents, computer programs, large web sites, and other collections of information.
  - Supported by a tool
  - Provides ways to see differences between versions
  - Allows parallel development through merges and branches

- Foundational in software development, but occasionally new to operations teams
Benefits of Version/Source Control

• One of the highest predictors of performance (State of DevOps Report 2014)

• Gene Kim hypothesizes as to why:
  – High numbers of failures are due to misconfiguration, not bad code
    • Countermeasure: source control all configuration to a single source of truth
  – Allows anyone to recreate environments solely from what is in source control, as opposed to requiring individual or tribal knowledge
What to Version Control

• Source Code
• Environment definition
• Infrastructure configuration
• Deployment scripts
• Documentation

• EVERYTHING!
Automation enables continuous value delivery
Benefits of Automation in DevOps

- Removes manual errors
- Enables anyone to perform tasks
- Enables speed, reliability and consistency
- Empowers frequent releases and self-service
What to Automate

• Build and Deployment
• Environment creation
• Infrastructure configuration
• Unit, Integration, UI and Performance Testing
• Documentation generation
• Monitoring and notifications

• EVERYTHING!
What Devs and Ops can learn from each other

- Value collaboration on all aspects of the system
  - Code and infrastructure/configuration
  - Solve issues early and quickly
- Have a production-first mindset
- Version control everything
- Automate everything (esp. manually intensive tasks)
- Create small, frequent deployments (of code and configuration)
- Monitor, log and validate performance obsessively
What is Continuous Integration (CI)?

• The practice of merging all developer working copies to a shared code line several times a day, and validating each integration with an automated build.
  – Unit tests are generally executed during the build

• In practice, CI is often defined as having a build with unit tests that executes at every commit / check-in to version control
  – This provides confidence in individual branches, but not on the integration of all the code changes
Continuous integration
Benefits of Continuous Integration

• Rapid feedback for code quality
• Trigger for automated testing for every code change
• Code analysis and technical debt management
• Reduces long, difficult and bug-inducing merges
• Increases confidence in code long before production
What is Continuous Delivery?

- A software engineering approach in which teams produce software in short cycles, ensuring that software can be reliably released at any time.
  - Aims to build, test and release software faster and more frequently
  - Reduce the cost, time and risk of delivering changes by allowing for more incremental updates to production

- In practice, continuous delivery focuses on an automated deployment pipeline
  - This may have one or more manual approval gates prior to reaching production
Continuous Delivery vs Continuous Deployment

- Continuous Deployment is generally defined as a Continuous Delivery pipeline with no manual gates between initial code commit / check-in and production.
- Feature flags are commonly used in both patterns, however, they are often necessary for Continuous Deployment.
  - Feature flags ensure that code deployed to a production environment is not necessarily released to all end users (Deployment Release).
  - This allows for more mature features to be enabled in production (generally via configuration), while newer features can be switched off for most users.
Continuous delivery
Benefits of Continuous Delivery

• Encourages Infrastructure as Code
• Encourages Configuration as Code
• Enables automated testing throughout the pipeline
• Provides visibility
• Provides fast feedback cycles
• Makes going to production a low stress activity
• Increases confidence in code long before production
What is a Build Pipeline?

• Automated system responsible for Continuous Integration
  – Builds code, runs unit tests, creates packages, etc.
  – Generally triggered by a code commit / check-in, or on a schedule

• Note: The Build Pipeline and the Deployment Pipeline can be considered two different concepts, but in many systems the same tool orchestrates both.
Defining a Build Pipeline

• Trigger
  – Typically a commit / check-in to version control
  – Can include gated check-ins

• Tasks
  – Compilation, minification, tokenization, etc.

• Unit Testing

• Code Analysis

• Versioning and Packaging