DevOps and Code Management: Code management and DevOps automation

Min-Yuh Day
Associate Professor
Institute of Information Management, National Taipei University

https://web.ntpu.edu.tw/~myday
2020-12-29
<table>
<thead>
<tr>
<th>週次 (Week)</th>
<th>日期 (Date)</th>
<th>內容 (Subject/Topics)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2020/09/15</td>
<td>軟體工程概論 (Introduction to Software Engineering)</td>
</tr>
<tr>
<td>2</td>
<td>2020/09/22</td>
<td>軟體產品與專案管理：軟體產品管理，原型設計 (Software Products and Project Management: Software product management and prototyping)</td>
</tr>
<tr>
<td>3</td>
<td>2020/09/29</td>
<td>敏捷軟體工程：敏捷方法、Scrum、極限程式設計 (Agile Software Engineering: Agile methods, Scrum, and Extreme Programming)</td>
</tr>
<tr>
<td>4</td>
<td>2020/10/06</td>
<td>功能、場景和故事 (Features, Scenarios, and Stories)</td>
</tr>
<tr>
<td>5</td>
<td>2020/10/13</td>
<td>軟體架構：架構設計、系統分解、分散式架構 (Software Architecture: Architectural design, System decomposition, and Distribution architecture)</td>
</tr>
<tr>
<td>6</td>
<td>2020/10/20</td>
<td>軟體工程個案研究 I (Case Study on Software Engineering I)</td>
</tr>
</tbody>
</table>
課程大綱 (Syllabus)

週次 (Week)  日期 (Date)  內容 (Subject/Topics)
7  2020/10/27  基於雲的軟體：虛擬化和容器，軟體即服務
(Cloud-Based Software: Virtualization and containers,
Everything as a service, Software as a service)
8  2020/11/03  雲端運算與雲軟體架構
(Cloud Computing and Cloud Software Architecture)
9  2020/11/10  期中報告 (Midterm Project Report)
10 2020/11/17  微服務架構：RESTful服務、服務部署
(Microservices Architecture: RESTful services,
Service deployment)
11 2020/11/24  軟體工程產業實務
(Industry Practices of Software Engineering)
12 2020/12/01  安全和隱私 (Security and Privacy)
<table>
<thead>
<tr>
<th>Week</th>
<th>Date</th>
<th>Subject/Topics</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>2020/12/08</td>
<td>軟體工程個案研究 II (Case Study on Software Engineering II)</td>
</tr>
<tr>
<td>14</td>
<td>2020/12/15</td>
<td>可靠的程式設計 (Reliable Programming)</td>
</tr>
<tr>
<td>15</td>
<td>2020/12/22</td>
<td>測試：功能測試、測試自動化化、測試驅動的開發、程式碼審查 (Testing: Functional testing, Test automation, Test-driven development, and Code reviews)</td>
</tr>
<tr>
<td>16</td>
<td>2020/12/29</td>
<td>DevOps和程式碼管理：程式碼管理和DevOps自動化 (DevOps and Code Management: Code management and DevOps automation)</td>
</tr>
<tr>
<td>17</td>
<td>2021/01/05</td>
<td>期末報告 I (Final Project Report I)</td>
</tr>
<tr>
<td>18</td>
<td>2021/01/12</td>
<td>期末報告 II (Final Project Report I)</td>
</tr>
</tbody>
</table>
Software Engineering and Project Management

**Analyze**
- Requirements definition

**Design**
- System and Software design

**Build**
- Implementation and unit testing

**Test**
- Integration and system testing

**Deliver**
- Operation and maintenance

---

**Project Management**
Product management concerns

- Business needs
- Technology constraints
- Customer experience

Technical interactions of product managers

- Product vision management
- Product backlog management
- User stories and scenarios
- Acceptance testing
- Customer testing
- User interface design

Software Development Life Cycle (SDLC)
The waterfall model

1. Requirements definition
2. System and Software design
3. Implementation and unit testing
4. Integration and system testing
5. Operation and maintenance

Plan-based and Agile development

Plan-based development

1. Requirements engineering
2. Requirements specification
3. Design and implementation

Requirements change requests

Agile development

1. Requirements engineering
2. Design and implementation

The Continuum of Life Cycles

Degree of Change

Low

High

Frequency of Delivery

Low

High

Incremental

Predictive

Agile

Iterative

Predictive Life Cycle

Analyze → Design → Build → Test → Deliver

Iterative Life Cycle

Analyze → Analyze Design → Build Test → Deliver

Prototype → Refine
A Life Cycle of Varying-Sized Increments

Analyze
Design
Build
Test
Deliver

Analyze
Design
Build
Test
Deliver

Analyze
Design
Build
Test
Deliver

Iteration-Based and Flow-Based Agile Life Cycles

**Iteration-Based Agile**

- Requirements Analysis
- Design
- Build
- Test
  
- Repeat as needed
  ...

**Flow-Based Agile**

- Requirements Analysis
- Design
- Build
- Test
  - the number of features in the WIP limit
  
- Repeat as needed
  ...

From personas to features

1. Personas
   A way of representing users

2. Scenarios
   Natural language descriptions of a user interacting with a software product

3. Stories
   Natural language descriptions of something that is needed or wanted by users

4. Features
   Fragments of product functionality

Multi-tier client-server architecture

Service-oriented Architecture

VM

Virtual web server

Server software

Guest OS

Hypervisor

Host OS

Server Hardware

Virtual mail server

Server software

Guest OS

Container

User 1

Container 1

Application software

Server software

User 2

Container 2

Application software

Server software

Container manager

Host OS

Server Hardware
Everything as a service

- **Software as a service (SaaS)**
- **Platform as a service (PaaS)**
- **Infrastructure as a service (IaaS)**

- Photo editing
- Cloud management
- Monitoring
- Storage Network
- Logistics management
- Database development
- Computing Virtualization

Software as a service

Software customers

Software provider

Cloud provider

Software services

Cloud Infrastructure

Microservices architecture – key design questions

- What are the microservices that make up the system?
- How should service failure be detected, reported and managed?
- How should data be distributed and shared?
- How should the microservices in the system be coordinated?
- How should microservices communicate with each other?
Types of security threat

**Availability threats**
An attacker attempts to deny access to the system for legitimate users
- Distributed denial of service (DDoS) attack

**Confidentiality threats**
An attacker tries to gain access to private information held by the system
- Data theft

**Integrity threats**
An attacker attempts to damage the system or its data
- Virus
- Ransomware

**SOFTWARE PRODUCT**
- PROGRAM
- DATA
Software product quality attributes

Reliability
Availability
Security
Resilience
Usability
Maintainability
Responsiveness

A refactoring process

1. Identify code ‘smell’
2. Identify refactoring strategy
3. Make small improvement until strategy completed
4. Run automated code tests

Functional testing

Start

1. Unit Testing
2. Feature Testing
3. System Testing
4. Release Testing
Test-driven development (TDD)

1. Identify new functionality
2. Identify partial implementation of functionality
3. Write code stub that will fail test
4. Run all automated test
5. Implement code that should cause failing test to pass
6. Run all automated test
7. Refactor code if required

Functionality complete
Functionality incomplete
All tests pass

DevOps and Code Management: Code management and DevOps automation
Outline

• Source code management
• DevOps automation
• DevOps measurement
Software support

• Traditionally, separate teams were responsible for software development, software release and software support.

• The development team passed over a ‘final’ version of the software to a release team.
  – Built a release version, tested this and prepared release documentation before releasing the software to customers.

• A third team was responsible for providing customer support.
  – The original development team were sometimes also responsible for implementing software changes.
  – Alternatively, the software may have been maintained by a separate ‘maintenance team’.

Software Development, release and support

Problem and bug reports

Software Development
- Tested software ready for release

Software Release
- Deployed software ready for use

Software Support

DevOps

- There are inevitable delays and overheads in the traditional support model.
- To speed up the release and support processes, an alternative approach called **DevOps (Development + Operations)** has been developed.
DevOps

• Three factors led to the development and widespread adoption of DevOps:
  – Agile software engineering reduced the development time for software, but the traditional release process introduced a bottleneck between development and deployment.
  – Amazon re-engineered their software around services and introduced an approach in which a service was developed and supported by the same team. Amazon’s claim that this led to significant improvements in reliability was widely publicized.
  – It became possible to release software as a service, running on a public or private cloud. Software products did not have to be released to users on physical media or downloads.

DevOps

Multi-skilled DevOps team

DevOps principles

• **Everyone is responsible for everything**
  All team members have joint responsibility for developing, delivering and supporting the software.

• **Everything that can be automated should be automated**
  All activities involved in testing, deployment and support should be automated if it is possible to do so. There should be minimal manual involvement in deploying software.

• **Measure first, change later**
  DevOps should be driven by a measurement program where you collect data about the system and its operation. You then use the collected data to inform decisions about changing DevOps processes and tools.

## Benefits of DevOps

<table>
<thead>
<tr>
<th>Benefit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Faster deployment</strong></td>
<td>Software can be deployed to production more quickly because communication delays between the people involved in the process are dramatically reduced.</td>
</tr>
<tr>
<td><strong>Reduced risk</strong></td>
<td>The increment of functionality in each release is small so there is less chance of feature interactions and other changes causing system failures and outages.</td>
</tr>
<tr>
<td><strong>Faster repair</strong></td>
<td>DevOps teams work together to get the software up and running again as soon as possible.</td>
</tr>
<tr>
<td><strong>More productive teams</strong></td>
<td>DevOps teams are happier and more productive than the teams involved in the separate activities.</td>
</tr>
</tbody>
</table>

Code management

• Code management is a set of software-supported practices that is used to manage an evolving codebase.

• During the development of a software product, the development team will probably create tens of thousands of lines of code and automated tests.

• These will be organized into hundreds of files. Dozens of libraries may be used, and several, different programs may be involved in creating and running the code.

Code management

• You need **code management** to ensure that changes made by different developers do not interfere with each other, and to **create different product versions**.

• **Code management tools** make it easy to create an executable product from its **source code files** and to **run automated tests** on that product.

Source code management, combined with automated system building, is essential for professional software engineering.

In companies that use DevOps, a modern code management system is a fundamental requirement for ‘automating everything’.

Not only does it store the project code that is ultimately deployed, it also stores all other information that is used in DevOps processes.

DevOps automation and measurement tools all interact with the code management system.

Code management and DevOps

DevOps automation

- Continuous integration
- Continuous deployment
- Continuous delivery
- Infrastructure as code

Code management system

- Branching and merging
- Recover version information
- Save and retrieve versions

Code repository

Transfer code to/from developer’s filestore

DevOps measurement

- Data collection
- Data analysis
- Report generation

Code management fundamentals

• **Code management systems** provide a set of features that support four general areas:

• **Code transfer**
  – Developers take code into their personal file store to work on it then return it to the shared code management system.

• **Version storage and retrieval**
  – Files may be stored in several different versions and specific versions of these files can be retrieved.

• **Merging and branching**
  – Parallel development branches may be created for concurrent working. Changes made by developers in different branches may be merged.

• **Version information**
  – Information about the different versions maintained in the system may be stored and retrieved.

Code repository

• All source code management systems have the general form with a shared repository and a set of features to manage the files in that repository:
  – All source code files and file versions are stored in the repository, as are other artefacts such as configuration files, build scripts, shared libraries and versions of tools used.
  – The repository includes a database of information about the stored files such as version information, information about who has changed the files, what changes were made at what times, and so on.

Code repository

• Files can be transferred to and from the repository and information about the different versions of files and their relationships may be updated.

  – Specific versions of files and information about these versions can always be retrieved from the repository.

Features of code management systems

- Version and release identification
- Change history recording
- Independent development
- Project support
- Storage management

In 2005, Linus Torvalds, the developer of Linux, revolutionized source code management by developing a distributed version control system (DVCS) called Git to manage the code of the Linux kernel.

This was geared to supporting large-scale open source development. It took advantage of the fact that storage costs had fallen to such an extent that most users did not have to be concerned with local storage management.

Instead of only keeping the copies of the files that users are working on, Git maintains a clone of the repository on every user’s computer.

Benefits of distributed code management

- Resilience
- Speed
- Flexibility
Branching and merging

- Branching and merging are fundamental ideas that are supported by all code management systems.
- A branch is an independent, stand-alone version that is created when a developer wishes to change a file.
- The changes made by developers in their own branches may be merged to create a new shared branch.
- The repository ensures that branch files that have been changed cannot overwrite repository files without a merge operation.

Branching and merging

Feature experiment branch

Alice

Bug fix branch

Bob

Merge

Master branch

DevOps automation

• By using **DevOps with automated support**, you can dramatically reduce the time and costs for integration, deployment and delivery.

• Everything that can be, should be **automated** is a fundamental principle of DevOps.

• As well as reducing the costs and time required for integration, deployment and delivery, **process automation** also makes these processes more **reliable and reproducible**.

• **Automation information** is **encoded in scripts and system models** that can be checked, reviewed, versioned and stored in the project repository.

# Aspects of DevOps automation

<table>
<thead>
<tr>
<th>Aspect</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Continuous integration</strong></td>
<td>Each time a developer commits a change to the project’s master branch, an executable version of the system is built and tested.</td>
</tr>
<tr>
<td><strong>Continuous delivery</strong></td>
<td>A simulation of the product’s operating environment is created and the executable software version is tested.</td>
</tr>
<tr>
<td><strong>Continuous deployment</strong></td>
<td>A new release of the system is made available to users every time a change is made to the master branch of the software.</td>
</tr>
<tr>
<td><strong>Infrastructure as code</strong></td>
<td>Machine-readable models of the infrastructure (network, servers, routers, etc.) on which the product executes are used by configuration management tools to build the software’s execution platform.</td>
</tr>
</tbody>
</table>
Characteristics of infrastructure as code

- Visibility
- Reproducibility
- Reliability
- Recovery

DevOps measurement

• After you have adopted DevOps, you should try to continuously improve your DevOps process to achieve faster deployment of better-quality software.

• There are four types of software development measurement:
  – Process measurement
  – Service measurement
  – Usage measurement
  – Business success measurement

Automating measurement

• As far as possible, the DevOps principle of automating everything should be applied to software measurement.
• You should instrument your software to collect data about itself and you should use a monitoring system to collect data about your software’s performance and availability.
• Some process measurements can also be automated.
  – However, there are problems in process measurement because people are involved. They work in different ways, may record information differently and are affected by outside influences that affect the way they work.

Metrics used in the DevOps scorecard

Process metrics

- Deployment frequency
- Change volume
- Percentage of failed deployment
- Lead time from development to deployment
- Mean time to recovery
- Percentage increase in customer numbers
- Performance
- Number of customer complaints
- Availability

Service metrics

DevOps is the integration of software development and the management of that software once it has been deployed for use. The same team is responsible for development, deployment and software support.

The benefits of DevOps are faster deployment, reduced risk, faster repair of buggy code and more productive teams.

Source code management is essential to avoid changes made by different developers interfering with each other.

Summary

• All code management systems are based around a shared code repository with a set of features that support code transfer, version storage and retrieval, branching and merging and maintaining version information.

• Git is a distributed code management system that is the most widely used system for software product development. Each developer works with their own copy of the repository which may be merged with the shared project repository.

Summary

• DevOps is the integration of software development and the management of that software once it has been deployed for use. The same team is responsible for development, deployment and software support.

• The benefits of DevOps are faster deployment, reduced risk, faster repair of buggy code and more productive teams.

• Source code management is essential to avoid changes made by different developers interfering with each other.

Continuous integration means that as soon as a change is committed to a project repository, it is integrated with existing code and a new version of the system is created for testing.

Automated system building tools reduce the time needed to compile and integrate the system by only recompiling those components and their dependents that have changed.

Continuous deployment means that as soon as a change is made, the deployed version of the system is automatically updated. This is only possible when the software product is delivered as a cloud-based service.

Summary

• **Infrastructure as code** means that the infrastructure (network, installed software, etc.) on which software executes is defined as a machine-readable model. Automated tools, such as Chef and Puppet, can provision servers based on the infrastructure model.

• **Measurement** is a fundamental principle of DevOps. You may make both process and product measurements. **Important process metrics** are deployment frequency, percentage of failed deployments, and mean time to recovery from failure.

References

• Titus Winters, Tom Manshreck, and Hyrum Wright (2020), Software Engineering at Google: Lessons Learned from Programming Over Time, O'Reilly Media.