# ELEN 7044 – Software Technologies and Techniques

# Assignment 1: Continuous delivery

**April 2015**

| **Name** | **Student Nr** |
|---|---|
| Adrian Freemantle | 1307968 |

## Abstract

Continuous delivery is an approach that automates the build, test and deployment elements of the software development lifecycle. Rapid feedback on failures allow issues to be corrected early in the development life cycle. This leads to increased code quality and reduces the overall development costs. While potentially costly to implement, the benefits realised can result in a significant return on the initial investment. Continuous delivery is not a silver bullet and cannot change organisational dysfunctions, it requires a qualitative as opposed to a quantitative development culture to be adopted.

**Table of Contents**

# 1 Introduction

Software development methodologies generally focus on how requirements are obtained, ensuring the validity of the requirements and the development and testing processes [1, p. 3]. The software development lifecycle is frequently seen as a set of sequential stages; development, testing, integration and deployment [2, p. 2]. Release processes are frequently complex, slow, require manual intervention and manual preparation of configuration files [3, p. 1].

Traditional manual release methods are slow and therefore tend to deploy large feature sets at infrequent intervals. This practice results in non-repeatable deployment processes, increases the risk for human error, requires extensive human testing and has unpredictable outcomes [1, p. 5] [3, p. 2]

This report will discuss what continuous delivery is and how it can be used to mitigate the risks and problems associated with traditional manual deployment techniques by automating the build, deploy, test and release processes of the software development lifecycle.

A brief case study of the author's experience with implementing continuous integration at a life insurance company will be presented along with benefits realised and challenges faced in adopting the practice in a real word scenario.

# 2 Continuous delivery

Continuous delivery is a process that aims to reduce the time between the conception of a requirement to the delivery of working software by addressing:

> *"...how all the moving parts fit together: configuration management, automated testing, continuous integration and deployment, data management, environment management, and release management."* [4, p. 2]

This goal of Continuous delivery is achieved through the usage of a deployment pipeline that automates the build, test and deployment elements of the software development lifecycle and provides centralised configuration management. This allows for continuous integration of software, fast detection of failures and provides a consistent and repeatable deployment process as it is constantly being tested with each deployment [5, p. 3].

Each source-code commit made by a developer is treated as potentially releasable code. This release candidate is processed through a series of stages or gates where it is tested and verified until there is sufficient confidence to deploy it to the next stage [3, p. 2]. This allows testers and operations teams to safely and efficiently deploy any version of the code to test and production environments at the click of a button [2, p. 2].

Should the release candidate fail at any of the stages, it is rejected and will not be able to progress to the next stage. This increases the visibility of failures and encourages rapid fixing of the problem as no further progression can happen until the issue is resolved [2, p. 3].

## 2.1 Continuous Integration

Continuous Integration (CI) is *the practice of building and testing your application on every check-in* [1, p. 13]. This practice does not focus on the upstream portion of the development process such as deployment, host-environment configuration and application configuration, it does however form the foundation on which continuous delivery operates [1, p. 4]. It is the first stage within the deployment pipeline.

## 2.2 Configuration Management

Configuration Management is *the process by which all artefacts relevant to your project, and the relationships between them, are stored, retrieved, uniquely identified, and modified* [1, p. 31].

Storing this information in a central repository is critical element of continuous delivery as all configuration settings for different environments, applications and automated tests must be kept consistent. It is this central repository which eliminates the possibility of incorrect configuration settings being applied in a manual deployment process.

## 2.3 Branching and Merging

Branching and merging is a release management strategy in which in which developers make a copy of the main development source code before making changes. This copy is referred to as a branch and changes can be made to the source code without affecting any other branches. [6]

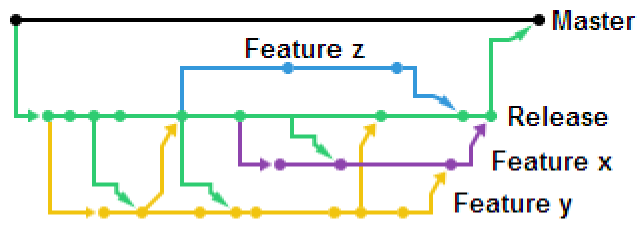


*Figure 2-1 Github network graph of a branching and merging strategy*

The isolation of changes means that a feature can be developed independently and then merged into a release candidate's branch. The merging process is the integration of the feature into the main body of code. This allows features to be developed in isolation until they are considered sufficiently stable for integration with the release candidate.

Branching and merging is crucial for the successful implantation of continuous deployment. Isolation of incomplete code from the release branch means that the release branch should always be in a potentially releasable state.

## 2.4 Deployment Pipeline

The deployment pipeline is *an automated implementation of your application's build, deploy, test, and release process.* [1, p. 3]

> *The aim of the deployment pipeline is threefold. First, it makes every part of the process of building, deploying, testing, and releasing software visible to everybody involved, aiding collaboration. Second, it improves feedback so that problems are identified, and so resolved, as early in the process as possible. Finally, it enables teams to deploy and release any version of their software to any environment at will through a fully automated process.* [1, p. 4]

The deployment pipeline is created from a collection of scripts, repositories and tools that may include; source control systems, change management systems, configuration management systems, virtualisation, test frameworks, build and integration systems [2, p. 3].

The number of stages are not prescriptive and there is no single correct manner in which to chain these tools and scripts together as the implementation will be different per company and even per project [3, p. 10]. As an example, certain systems may require performance testing, this can be addressed by including a performance or capacity testing stage.

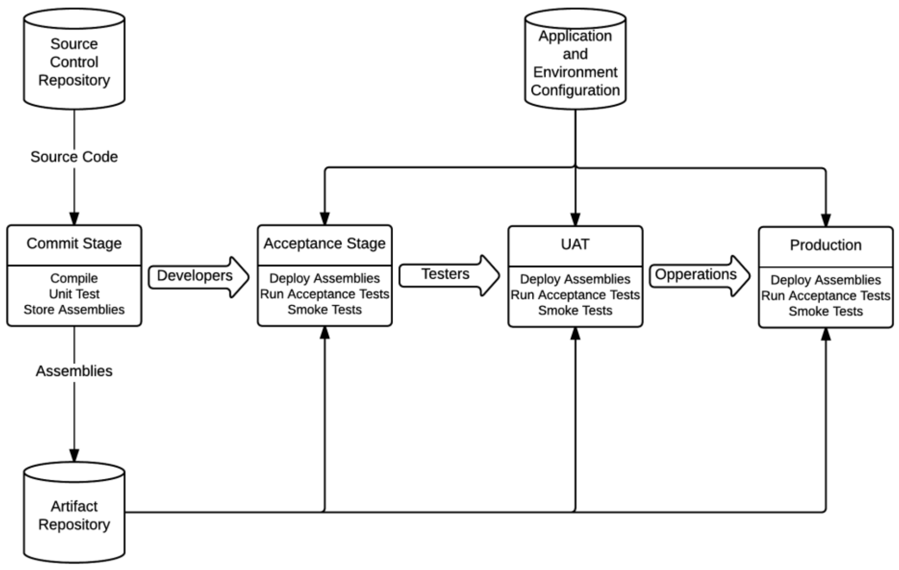


*Figure 2-2 Basic Deployment Pipeline [4, p. 1]*

### 2.4.1 Commit Stage

The first stage in the deployment pipeline is the commit stage. Building on the concept of continuous integration, every source-code commit triggers the build of a new release candidate. Code is compiled, unit tests are run and the resulting artefacts, assemblies and installers are stored in an artefact repository [1, p. 4].

### 2.4.2 Acceptance Stage

This is the second stage in the deployment pipeline. The system is deployed to a test environment and configured with the settings for this specific environment. Test databases may be created and seeded with predetermined test data. This ensures the tests performed here are done in an environment that exists in a known state [3, p. 5].

Automated acceptance tests are executed once the deployment and configuration have been completed. These are end-to-end functional tests which verify that the software operates according to the criteria captured in the requirements [3, p. 5].

### 2.4.3 UAT

The preceding stages are entirely automated and provide developers with immediate feedback on any failures. This allows the development team to rapidly respond to and fix any issues. The benefit of this is that faster detection of failures at the commit and acceptance stages is significantly less expensive to fix than had the error been found later in the development life cycle [3, p. 4].

Not all testing can be done though automatic means. Examples of testing that are best done manually include; usability testing, exploratory testing and non-functional acceptance tests [1, p. 85].

The User Acceptance Testing (UAT) environment is a testing environment that should be similar to the production environment in every possible manner. This includes database versions, operating system versions, configuration settings of the environment and application and security roles and permissions.

While automated acceptance and smoke tests should be executed here to catch immediate failures, the primary focus is on human based interaction and testing. A tester will exercise the functionality of the system and may demonstrate it to stakeholders in order to get feedback on the suitability of the system for their need. [1, p. 86]

At the end of this stage, confidence in the release candidate will be sufficient to promote it to the production environment.

#### 2.4.4 Production

The final stage is the deployment of the release candidate to the production environment. Test will be executed to ensure that the deployment was successful and that all configuration settings were applied correctly. Should any errors be detected at this stage, the deployment can be rolled back to the previous version.

## 2.5 Advantages

The main advantages of continuous deployment can be summarised as:

1. Developers spend more time adding value and delivering working code.
2. Detecting failures early results in lower total development costs.
3. Automated testing provides faster, wider and consistent levels of test coverage.
4. Testers are free to perform exploratory testing.
5. Automated deployment pipelines keep detailed audit logs of deployments.
6. Continuously deploying the system to a production like environment means the deployment process itself is constantly being tested.
7. Release cycles are drastically reduced.

[1, p. 8]

## 2.6 Disadvantages

The primary disadvantage is the initial time and cost of setting up a deployment pipeline. The number of tools that need to be configured and the number of scripts to integrate the various components of the deployment pipeline pose a significant barrier of entry for most small to medium enterprises.

# 3 Continuous Delivery at a Life Insurance Company

The following section will provide a brief case study of the author’s experience with implementing continuous integration at a life insurance company.

## 3.1 Deployment Pipeline

The deployment pipeline uses a variety of tools and platforms to minimize the amount of custom scripting and lower the barrier of entry for future projects to be deployed in this manner.

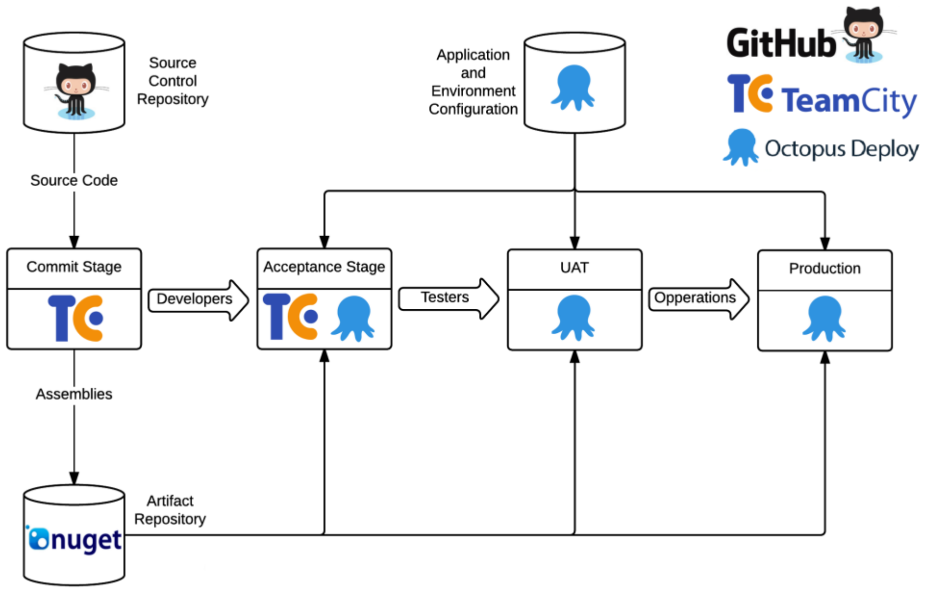


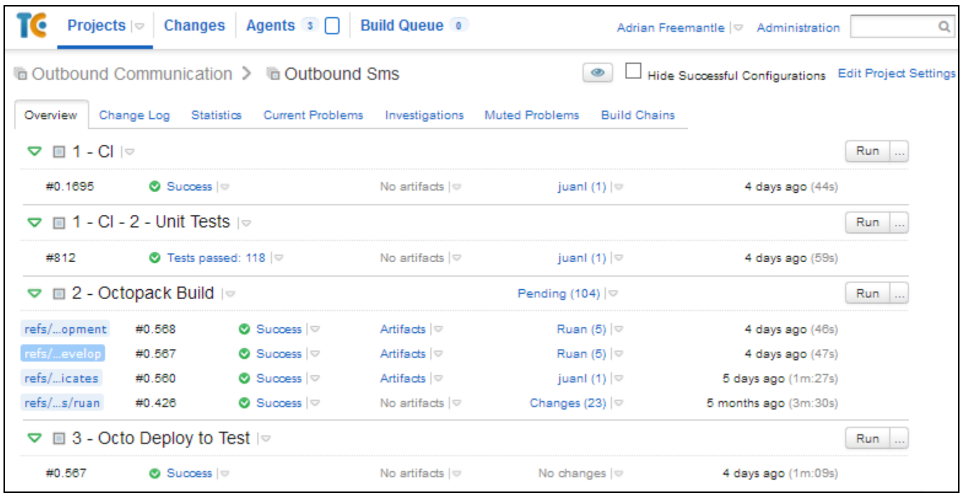
*Figure 3-1 Deployment pipeline and technologies [4, p. 1]*

Tools used include:

- Github for source control
- TeamCity for continuous integration and the building of deployment packages.
- Nuget as an artefact repository for deployment packages
- Octopus Deploy for deployment as well as application and environment configuration-management.

## 3.2 Continuous Integration

TeamCity provides detailed reports on the continuous integration state of each system or component. This level of visibility makes it easy to determine if a particular build has failed and what the reason was.



1 - CI

| | | | | | |
|---|---|---|---|---|---|
| | #0.1695 | Success | No artifacts | juanl (1) | 4 days ago (44s) |

1 - CI - 2 - Unit Tests

| | | | | | |
|---|---|---|---|---|---|
| | #812 | Tests passed: 118 | No artifacts | juanl (1) | 4 days ago (59s) |

2 - Octopack Build — Pending (104)

| | | | | | |
|---|---|---|---|---|---|
| refs/...opment | #0.568 | Success | Artifacts | Ruan (5) | 4 days ago (46s) |
| refs/...evelop | #0.567 | Success | Artifacts | Ruan (5) | 4 days ago (47s) |
| refs/...icates | #0.560 | Success | Artifacts | juanl (1) | 5 days ago (1m:27s) |
| refs/...s/ruan | #0.426 | Success | No artifacts | Changes (23) | 5 months ago (3m:30s) |

3 - Octo Deploy to Test

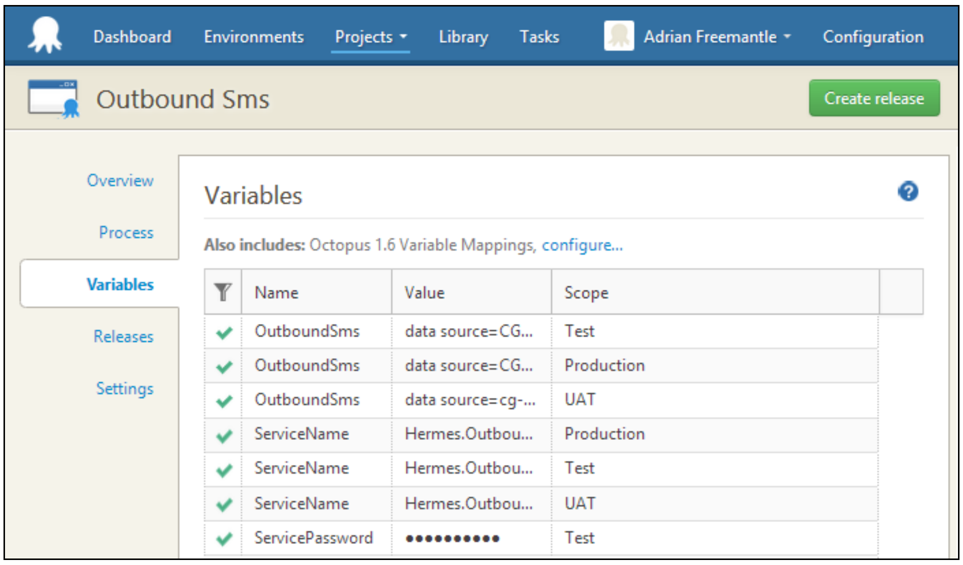
| | | | | | |
|---|---|---|---|---|---|
| | #0.567 | Success | No artifacts | No changes | 4 days ago (1m:09s) |

*Figure 3-2 Continuous Integration with TeamCity*

TeamCity stores the compiled assemblies as a NuGet package which Octopus Deploy can use to deploy the system to any environment.

### 3.2.1 Automated Deployment and Configuration Management

Octopus Deploy is used for environment configuration-management and deployment. It provides a deployment mechanism that allows anyone to deploy software to environments that they have access to while protecting sensitive information such as service account passwords.

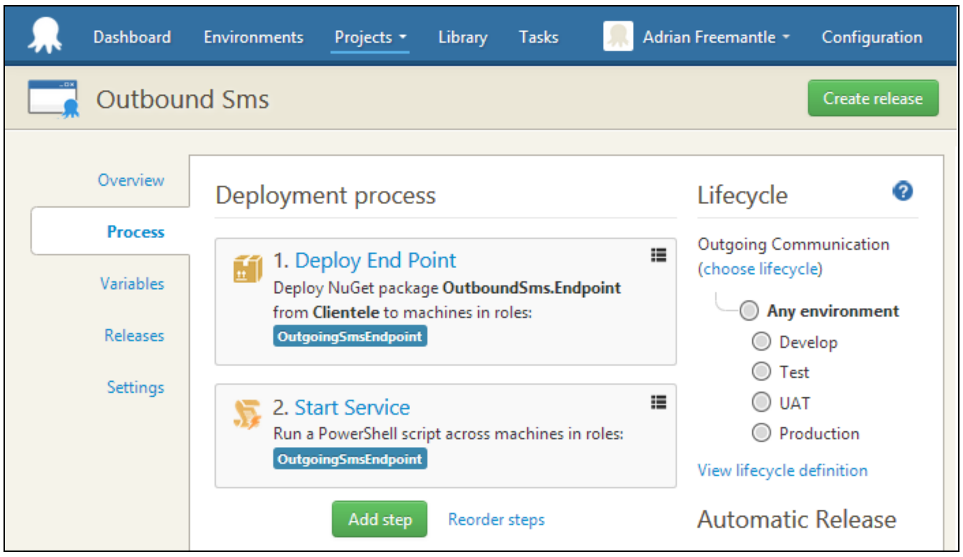


Variables

Also includes: Octopus 1.6 Variable Mappings, configure...

| | Name | Value | Scope |
|---|---|---|---|
| ✓ | OutboundSms | data source=CG... | Test |
| ✓ | OutboundSms | data source=CG... | Production |
| ✓ | OutboundSms | data source=cg-... | UAT |
| ✓ | ServiceName | Hermes.Outbou... | Production |
| ✓ | ServiceName | Hermes.Outbou... | Test |
| ✓ | ServiceName | Hermes.Outbou... | UAT |
| ✓ | ServicePassword | •••••••••• | Test |

*Figure 3-3 Environment Configuration with Octopus Deploy*



Deployment process

1. Deploy End Point
Deploy NuGet package **OutboundSms.Endpoint** from **Clientele** to machines in roles: OutgoingSmsEndpoint

2. Start Service
Run a PowerShell script across machines in roles: OutgoingSmsEndpoint

Add step · Reorder steps

Lifecycle

Outgoing Communication (choose lifecycle)

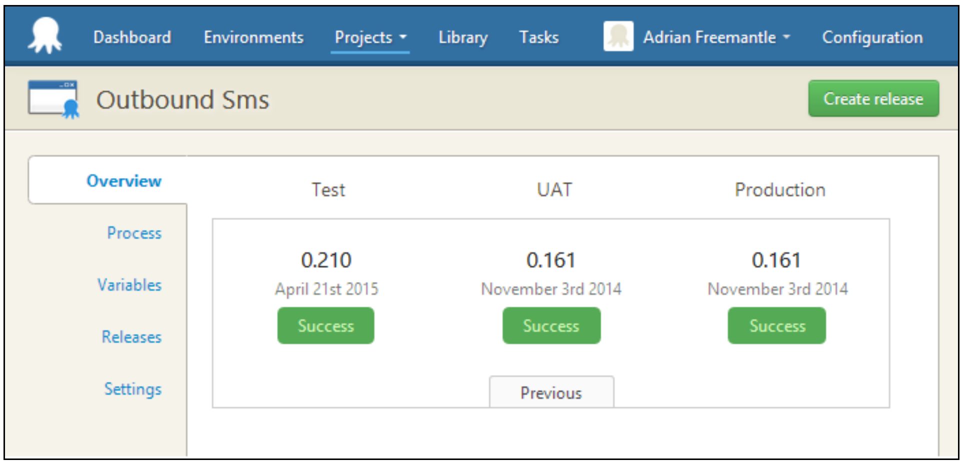
- **Any environment**
  - Develop
  - Test
  - UAT
  - Production

View lifecycle definition

Automatic Release

*Figure 3-4 Deployment Process Configuration with Octopus Deploy*

Configuring a deployment script with Octopus deploy is simple as it has predefined steps that can be chained together to create a deployment process. The flexibility and ease of use has had caused developers to dislike the creation manual deployment scripts.



| Test | UAT | Production |
|---|---|---|
| 0.210 | 0.161 | 0.161 |
| April 21st 2015 | November 3rd 2014 | November 3rd 2014 |
| Success | Success | Success |

*Figure 3-5 Version Control with Octopus Deploy*

Release candidates can be promoted to environments or rolled back easily and quickly. Version control is simplified as it is always know which release candidate is deployed to a specific environment and audit logs are also kept of every deployment, rollback and configuration change.

## 3.3 Benefits Realised

Without continuous deployment the new Service Oriented Architecture being introduced would have been difficult to implement. A lot of time would have been wasted on manual deployments and version control would have been difficult. With the current process it is possible to create an end-to-end deployment process for a new service in a few minutes.

The reliability, repeatability, security, auditability and ease of rolling back a deployment has made it possible to deploy services multiple times per day and to break monolithic applications into smaller services.

Junior developers feel more empowered to make changes with the confidence that serious mistakes will be detected. The rapid feedback cycle allows developers to make mistakes and learn from them.

The development focus is shifting from a production focus to a quality focus. This is largely due to the fact that errors are detected early and that this information is publically visible to developers and stakeholders. The unexpected result of this is that by focusing on quality our productivity has increased as well.

### 3.4 Barriers to Adoption

Legacy systems were difficult and in some cases impossible to include in this process, their architectures or platforms being the primary barrier.

It was difficult to get time allocated for research, funding for tools, and time for the implementation. Departmental or team managers felt that the concept was theoretical and would not work within their environment.

Changing from a production focused culture to a quality focused culture continues to present challenges. The deployment pipeline is only as good as the tests created, it has not been uncommon to find systems being deployed using the pipeline with no tests in place.

In some instances, the mind-set that developers work with code and testers only test against the user interface prevails. This results in developers that refuse to create unit tests and testers that refuse learn how to write automated acceptance tests.

## 4 Conclusion

Continuous delivery is an approach that automates the build, test and deployment elements of the software development lifecycle through the use of a deployment pipeline. The focus on automated testing provides rapid feedback on failures which allow such issues to be corrected early in the development life cycle, helping reduce the total development cost.

When properly implemented, it has the potential to increase code quality, reduce the overall development costs and reduce the time taken from for a concept to translate into usable functionality. Test automation is central to achieving these goals.

Implementing a deployment pipeline requires a significant investment in terms of acquiring skills, training staff, acquiring tools and creating deployment scripts. The long term benefit is significant and offsets any initial costs incurred.

The development culture, existing infrastructure and code base can be significant obstacles to overcome. Adopting continuous delivery requires that the organisation adopt a quality focused culture, without this the deployment pipeline simply becomes a more efficient means of deploying defective systems into production.

## 5 References

[1] J. Humble and D. Farley, Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation, Addison Wesley Professional , 2010.

[2] Forrester Consulting, “Continuous Delivery: A Maturity Assessment Model,” Forrester Research, Inc, 2012.

[3] D. Farley, “The Deployment Pipeline,” 2007.

[4] P. Duvall , “Continuous Delivery: Patterns and Antipatterns in the Software Lifecycle,” DZone, Cary, 2011.

[5] CloudBees, Inc., “The Business Value of Continuous Delivery,” CloudBees, Inc., Los Altos, 2015.

[6] red-bean.com, “What's a Branch?,” red-bean.com, [Online]. Available: http://svnbook.red-bean.com/en/1.7/svn.branchmerge.whatis.html. [Accessed 28 05 2015].

[7] Praqma, “Praqmatic Software Development,” Praqma A/S, Allerød - Denmark.