Best Practices, Benefits and Obstacles When Conducting Continuous Delivery in Software-Intensive Projects

By
Björn Hansson

Supervisor:
Helena Holmström Olsson

Examiner:
Johan Holmgren
Contact Information

Author:
Björn Hansson
E-mail: bjorn.hansson@live.se

Supervisor:
Helena Holmström Olsson
E-mail: helena.holmstrom.olsson@mah.se
Malmö University, Department of Computer Science.

Examiner:
Johan Holmgren
E-mail: johan.holmgren@mah.se
Malmö University, Department of Computer Science.
"... Our highest priority is to satisfy the customer through early and continuous delivery of valuable software ..." – Agile Manifesto
Abstract

The goals with continuous delivery are to reduce the risk, cost, and time of releasing software to the stakeholders and the users. It is a process which can result in reliable releases and reducing errors in the software. Furthermore, there are some best practices to follow when conducting the continuous delivery process, involving version control, and build tools. There are however some obstacles and challenges for organizations when moving to continuous delivery. For example, complex environments, organizational problems, and lack of automated test cases. This master thesis investigates continuous delivery through a literature review, a multiple-case study, and fieldwork. The result can either be used by software engineers and organizations who are new to the continuous delivery concept. Or the result can be used by more experienced software engineers to gain more knowledge about existing obstacles and for further research.

Keywords — continuous delivery; continuous integration; continuous deployment; software engineering;
Acknowledgements

I would like to thank my supervisor Helena Holmström Olsson for all the advice and guidance throughout the process of writing the master thesis. I would also like to thank Andreas Rudén, Jesper Lähdevirta and the other participants from the industry for their time and valuable knowledge.
## Contents

1 Introduction 8

2 Background 10
   2.1 Continuous Activities .................................. 10
   2.2 DevOps .............................................. 11
   2.3 Technology and Components .............................. 11
      2.3.1 Operating System ................................. 11
      2.3.2 Version Control System ........................... 11
      2.3.3 Staging Area and Code Collaboration Tool ........ 12
      2.3.4 Build Automation and Dependency Manager Tool ... 12
      2.3.5 Automation Server ................................ 12
      2.3.6 Off-the-shelf Solutions ............................ 13
   2.4 Continuous Delivery System ............................. 13

3 Research Goals and Method 15
   3.1 Research Goals ....................................... 15
   3.2 Research Questions .................................... 15
   3.3 Expected Results ...................................... 16
   3.4 Literature Review ..................................... 16
      3.4.1 Literature Search Method .......................... 16
   3.5 Multiple-case Study .................................... 16
      3.5.1 Interviews ........................................ 17
   3.6 Development and Fieldwork .............................. 17
   3.7 Evaluation ........................................... 17
   3.8 Validity ................................................ 17
   3.9 Limitations ............................................ 18
   3.10 Alternatives .......................................... 18

4 Literature Review Result 19
   4.1 Benefits (RQ1) ....................................... 19
      4.1.1 Reliable Releases .................................. 19
      4.1.2 Reducing Defects ................................... 19
      4.1.3 Improved Efficiency and Productivity ............ 19
      4.1.4 Shorter Time to Market ............................ 20
      4.1.5 Empowering Teams ................................ 20
      4.1.6 Lowering Stress ................................... 20
   4.2 Obstacles (RQ2) ....................................... 20
      4.2.1 Lack of Test Automation ............................ 21
      4.2.2 Complex Environments and Technical Challenges ... 21
      4.2.3 Human, Organizational and Resource Problems .... 21
   4.3 Best Practices (RQ3) .................................. 22
      4.3.1 Create a Reliable and Repeatable Process ....... 22
5 Multiple-case Study Result
  5.1 Company A ........................................... 24
    5.1.1 Background .................................... 24
    5.1.2 Delivery Setup .................................. 24
    5.1.3 Benefits ....................................... 24
    5.1.4 Obstacles .................................... 25
    5.1.5 Solutions and Best Practices ................. 25
  5.2 Company B ........................................... 26
    5.2.1 Background .................................... 26
    5.2.2 Delivery Setup .................................. 26
    5.2.3 Benefits ....................................... 26
    5.2.4 Obstacles .................................... 27
    5.2.5 Solutions and Best Practices ................. 27
  5.3 Company C ........................................... 27
    5.3.1 Background .................................... 27
    5.3.2 Delivery Setup .................................. 28
    5.3.3 Benefits ....................................... 28
    5.3.4 Obstacles .................................... 28
    5.3.5 Solutions and Best Practices ................. 29
  5.4 Company D ........................................... 29
    5.4.1 Background .................................... 29
    5.4.2 Delivery Setup .................................. 29
    5.4.3 Benefits ....................................... 30
    5.4.4 Obstacles .................................... 30
    5.4.5 Solutions and Best Practices ................. 30

6 Development and Fieldwork Result 32

7 Discussion and Future Work 35
  7.1 The Stairway ....................................... 35
  7.2 RQ1 .................................................. 36
  7.3 RQ2 and RQ3 ......................................... 36

8 Conclusion 39

9 References 40

10 Appendix 42
   10.1 Interview Questions ............................... 42
1 Introduction

Delivering new software to the stakeholders and the users can be a time-consuming process. It can be a process that contains certain risks and problems. Being able to release and deliver software painlessly on time, whenever needed, is an ever more emerging demand. Furthermore, with the demand to ensure the same or even better software quality from previous releases. Continuous delivery (CD) is a software engineering process or an approach to tackle these problems. It aims at developing software in short cycles that gets released safely, more frequently, tested and at any given time. Continuous delivery is about always being releasable and ready to deliver. This is partly achieved by focusing on automation, short cycles, and fast feedback [1]. Continuous delivery can be compared to the traditional software development process, e.g. Waterfall. In Waterfall, the software integration and release activities are in its own lifecycle as an independent step in a later phase of the project [2]. In continuous delivery, the integration and release activities are instead iterative used throughout the entire project.

Figure 1: "The stairway to heaven" per Olsson et al. [3].

An organization’s maturity regarding their evolution path for their way to work with software projects can be viewed as a stairway, illustrated in Figure 1, where more benefits, such as closer contact with the customers and reduced waste of unused functionality, are possible higher up on the stairway. The stairway goes from traditional development (e.g. Waterfall), up to more and more agile practices. Organizations are on different steps in this stairway and there are some identified obstacles between these steps. This master thesis investigates continuous delivery, which in this case is considered to be a part of step D in Figure 1. Continuous delivery is investigated by looking at the benefits, obstacles, and best practices.

The motivations for the research in this master thesis are that the barriers
are still to be explored in the ”stairway to heaven” [3] and it has been shown that a lot of companies has no automatic delivery process all the way from a change in the product to deployment in a production environment [4]. Furthermore, Laukkanen et al. state that obstacles and solutions regarding continuous delivery should be investigated further [5]. Hence, there seems to still be a need for more research and it may provide some value. Research questions, goals and methods used to investigate this continuous delivery topic are described in section 3.
2 Background

2.1 Continuous Activities

Continuous delivery or just "continuous" can sometimes be used as an umbrella term for other continuous activities. The term "continuous software engineering" can also be used to describe the different continuous activities [6]. Continuous delivery builds upon for instance continuous integration (CI), which is the first step in the continuous delivery process. Even here, there is not one homogeneous practice of continuous integration [7]. After continuous delivery comes continuous deployment in the typical evolution path for a software [3]. Continuous delivery is about delivering, while continuous deployment is focused on the automated software deployment to production. When looking at the goal of continuous deployment, there are no manual steps between a developer’s new change and a deployment to production. But for continuous delivery, there can still be manual steps before a change goes to production. While continuous integration, on the other hand, is only concerning the software integration and it is not about delivering [5].

The continuous integration process is to assemble the software code, components and produce a single unified software product. It aims at minimizing the risks of software integration and to improve the quality. In practice, it means that the software developers code commits, frequently gets tested and automatically compiled before being integrated into the final software. Furthermore, continuous integration should provide immediate feedback to the developers about their newly integrated code [1]. Continuous integration is about integrating code frequently, usually at least daily [8]. The concep-
The dual difference between continuous integration and continuous delivery is illustrated in Figure 2.

2.2 DevOps

DevOps is a similar, but different, concept compared to continuous delivery. It centers more around organizational changes, e.g. to improve the collaboration between the people working with the delivery process. Bringing operations and development engineers together and make them collaborate using agile techniques thru the whole lifecycle of a software. DevOps can be called for a movement, rather than a process [1].

2.3 Technology and Components

The purpose of this section is to provide a background of the technologies and components that may be needed to build a continuous delivery system. This section also clarifies some definitions which are later used in this master thesis and increases the understanding of how continuous delivery could be implemented in practice. Of course, each project has its own requirements and there exist many alternatives. Most of the discussed examples are either free or open source, chosen consciously partly for the possibility for the reader to try the technologies. The other reason for some of the examples is because they are used by the companies involved in the multiple-case study of this master thesis.

2.3.1 Operating System

As for all software, it will need to be running in a place where the computer hardware and software resources are managed, as well as common services are provided. Ubuntu is one Linux open source operating system to choose from and it offers, for instance, Ubuntu Desktop and Ubuntu Server versions [9]. There exist several other Linux distributions to choose from, e.g. Red Hat Linux [10]. Most of the software for continuous delivery is originally developed specifically for Linux. However, a non-open source operating system could be used as well, such as Windows or Mac OS.

2.3.2 Version Control System

VCS (Version Control System) handles the changes of the different files of a project, normally source code in the context of this master thesis. The changes to the VCS are identified by something unique, e.g. a number, hash code or timestamp. This makes it possible to for instance go back in history and compare changes between different versions. More typical information stored together with the changes are who did the change, e.g. the author, and why. Git [11] is a popular VCS solution, has a strong foothold in the
industry [12] and is often used in the open source world. An alternative could be Subversion [13].

There are some common terms used when writing about VCS. One term is Repository, that is where the data for a project is stored, typically on a server. "Commit" is the name for a change locally or in the server repository. "Push" means uploading local changes. "Merged" or "Checked in" means that a change is saved and stored in the repository. "Pull" or "Checkout" are terms for fetching changes stored in the repository.

2.3.3 Staging Area and Code Collaboration Tool

There might be a need for a staging area before code gets merged into the version control system. A stage where automated builds can run the code and where other developers can review the code. Reviewing changes is important for ensuring the quality of the code [1] and to share knowledge between developers. A code collaboration tool could be used for this and would as well share the responsibility for the code. One such tool is Gerrit [14], which is free and web-based. Developers can push the code to the collaboration tool and the automation server will take the changes and run a build. After approval from other developers and successful builds, can the code automatically get merged into the version control system.

2.3.4 Build Automation and Dependency Manager Tool

To be able to automate tasks, such as test cases, is a build automation tool needed. One tool especially suited for Java projects is Maven [15], but it could be used for other projects with other languages as well. In Maven, the instructions how the project should be built and what dependencies (such as external Java libraries) there are, is described in an XML file. The instructions are normally a part of the software under development and are checked into the VCS. An alternative to Maven is Gradle [16], which is for instance used as the official choose when developing Android applications.

2.3.5 Automation Server

Every project using continuous integration or continuous delivery will need some type of automation server software. A server that can build the software that is being developed and run certain tasks, such as test cases. Testing is mostly the crucial part of the automation server. Different test phases to be run could be unit, integration, system, and acceptance testing. Sometimes the phases are separated in different builds. Furthermore, the automation server presents the results and reports of the builds. Jenkins [17], the fork from Hudson, is one option to use and it is open source. Its functionality can heavily be extended by using plugins, e.g. Maven Integration [18].
Usage of automation or continuous integration servers, mainly Jenkins, are frequently used in the industry [12, 19].

2.3.6 Off-the-shelf Solutions

Off-the-shelf is a term used to describe a product that does not need to be specially developed to suit a particular purpose or organization, e.g. it is ready and available to be used immediately. There are some claims that there exists no off-the-shelf or out-of-the-box solution for continuous delivery [5, 20]. With some reservation on how to interpret the claims of course. However, there are some services claiming to offer a full solution for continuous delivery. One such continuous delivery solution is offered by AWS (Amazon Web Services) [21]. AWS offers many on-demand cloud computing platforms and services. Travis CI [22] is another possible off-the-shelf solution choice, often used together with GitHub [23].

2.4 Continuous Delivery System

To understand how continuous delivery could work in practice and the technology behind it has a general test application example been developed.

![Figure 3: Mockup to illustrate an application to be tested.](image)

The test application is consciously generic and an image to illustrate it is viewed in Figure 3. It is a time tracker, which can be used to record working time and later fetch the time history. This purpose of the application is not important but it is used as an example. The application has a frontend
developed with HTML and JavaScript. It communicates via a WEB API in the backend which is developed with Java. The recorded data is stored in a MySQL database. The application is using Maven as build automation and dependency manager.

Secondly, has a continuous delivery system example been developed for the test application. The continuous delivery system makes it possible to update the code for the test application and then automatically verify the full test application, before automatically being deployed to production. This can be called the process of continuous delivery, or in the literature often referred to as the deployment pipeline [1]. The continuous delivery system involves subcomponents mentioned in the above chapter, such as automation server.

![Diagram for a use case realization of the continuous delivery system.](image)

The continuous delivery system is setup as in the diagram in Figure 4. A developer can push a commit to the code collaboration tool (Gerrit). The automation server (Jenkins) pick up the commit and then run several builds with it, partly using Maven. If the builds for some reason fails, it will simply not be possible to merge code to the VCS (Git). The automation server sends this feedback directly to the developer or via the code collaboration tool. The developer can fix the problem and amend the changes to the same commit. The developer pushes the commit again and this time the builds succeed. The commit can now be submitted and automatically merged, for instance after approval from a second developer. After the merge, will a deploy build start and the newly built artifact is released and deployed automatically to production, e.g. to a server. This is one example how a continuous delivery (and deployment) system can be implemented.
3 Research Goals and Method

3.1 Research Goals

The goal of this master thesis is to make a qualitative and cross-sectional study consisting of two main methods: literature review and case study. In addition to this are also development and fieldwork performed. The reason for this combination of methods is that the information gathered hopefully complement each other and is more in-depth than using just one method. The research process is illustrated in Figure 5. The study is examining the impact on a software project when conducting continuous delivery, looking at benefits, obstacles, and best practices. This is defined in three research questions.

![Research process flowchart](image)

Figure 5: Research process flowchart.

3.2 Research Questions

**RQ1** What are the benefits of conducting continuous delivery in a software project?

**RQ2** What are the obstacles or challenges of conducting continuous delivery in a software project, and how might they be solved?

**RQ3** What are the best practices regarding continuous delivery in a software project?
3.3 Expected Results

The contribution of this master thesis is to form a deep foundation for understanding the benefits, obstacles and best practices when conducting continuous delivery. The expected results are to connect literature knowledge to the industries knowledge and to give possible solutions for obstacles when conducting continuous delivery. Also to give motivations for future work within the continuous delivery topic. E.g. a foundation with commonly identified obstacles and with a clearly visible research gap to investigate. Further on, hopefully contributing with new knowledge about how an organization can reach the highest step in the ”stairway to heaven” [3] evolution path for a software.

3.4 Literature Review

The process of gathering relevant knowledge and information is conducted by doing a literature review. It is important to learn about previous academic research first and to get the knowledge within the given topic to find a worthwhile research gap for the thesis. It will also build the foundation and increase the understanding of the topic for the researcher [24].

3.4.1 Literature Search Method

The research questions are answered in this paper by doing a literature review. Google Scholar and Malmö University Libsearch were used as online search tools. Searches were done using the following keywords: ”continuous delivery”, ”continuous deployment”, ”continuous activities”, in combination with the keyword ”software” to exclude results not part of computer science or software engineering. The highest-ranking search results were then chosen based on reading the abstract or introduction and by reviewing the origins.

3.5 Multiple-case Study

The multiple-case study process was conducted at four companies in the software industry, where the literature review result was used as input. A literature search often precedes the work of a case study. Case studies are well suited for software engineering research questions and can provide a better understanding of the case or phenomena under study. Further on, case studies tend to provide rich and deep descriptions of the phenomenon [24, 25]. Since there are four companies that are very different, involved in this master thesis, it is considered as four case studies, hence the term multiple-case study.
3.5.1 Interviews

Interviews were used as the data collection method. One definition of interviews is that it is a particular kind of conversation between people, with a set of predefined assumptions. Usually, an interview is planned and has an agenda. It can be a suitable data generation method when there are complex or open-ended questions. Normally resulting in obtaining detailed information [24].

The interview questions are to some extent abstract to cover the entire master thesis and to not affect the participants in a particular orientation. The interviews are semi-structured, which is common in case studies, meaning the interviews are mostly open and the order of the questions is not fixed. The interview questions were followed up with sub-questions which are based on the results of the literature review. The participants received a list in advance with questions to be discussed and answered. The participants are able to develop his or her ideas and speak more freely than in a structured interview [25].

3.6 Development and Fieldwork

Development and fieldwork (that could also be considered like action research [24], or field research) at one software company was conducted for further data collection. Fieldwork is the collection of information outside a laboratory, e.g. in the field, and is generally characterized as a qualitative method. The fieldwork aimed to retrieve knowledge not provided by the interviews and to understand a continuous delivery system in practice in the industry. The fieldwork investigated a solution for the company to achieve or improve their continuous delivery process and system. Field documentation was carried out during the fieldwork, as well as iteratively consulting the literature for knowledge.

3.7 Evaluation

The drafted formulated results of the conducted case studies were sent back to the participants of the case studies for evaluation, comments, and feedback. To evaluate the fieldwork were discussions or unstructured interviews conducted iteratively via email with a stakeholder at the involved company.

3.8 Validity

For the literature review were only books, papers, journals, and magazines articles included. Any potential web-only sources and blogs were excluded since they can disappear or move without warning [24]. The literature review could miss some facts depending on the used keywords and online search tools.
Some interviews were performed only over the phone and that could affect the results. However, to increase the validity of the result of the multiple-case study, were results of the interviews sent back to the participants to enable corrections [25]. During interviews was audio recording sometimes used if allowed. Participants are to be considered experts in the software engineering field, however not all necessarily experts in continuous delivery. To make the participants feel comfortable were they allowed to be anonymous, which also allowed for more transparency in their answers.

There could have been a company confidentiality issue when conducting the case studies, resulting in that some information might not be allowed to be shared. Which means that the data might need to be anonymized. In addition, while performing the interviews there is always a risk that the answers from the participants will be affected by the person conducting the interview [24]. In these cases, might the quality of the data suffer. Development and fieldwork might not be considered an established research method.

3.9 Limitations

The master thesis is focusing on identifying the obstacles and best practices regarding continuous delivery from a software engineering perspective, in contrast to a manager perspective.

The literature review was focused on literature issued after 2000 and the literature was limited on how they responded to the research questions.

The multiple-case study is focusing on RQ2 since it is considered to add the most value for the industry. The interviews were limited to the available time from the participants.

3.10 Alternatives

An alternative research method is to conduct a quantitative survey, i.e. obtaining the same kinds of data from a large group of people, within the software industry. A survey will, however, require a high response rate to be considered valid, e.g. at least 30 for small-scale research studies. Another alternative is conducting an experiment. Both survey and experiment simplify the complexities of the real world [24], therefore it may not be suitable for the real-world topic of this master thesis.
4 Literature Review Result

The literature review has resulted in several relevant results. They are divided into three sections in this paper, Benefits, Obstacles, and Best Practices, to reflect the research questions. Continuous delivery and continuous integration are sometimes used loosely for explaining the same thing in the literature. The two approaches are somewhat intertwined. Continuous integration is part of this section and seen as a pre-requirement for doing continuous delivery.

4.1 Benefits (RQ1)

The purpose of this section is to define benefits regarding continuous delivery. Needed partly to justify the need for organizations to adapt to continuous delivery and to see the values of it.

4.1.1 Reliable Releases

One benefit which stands out above the other benefits is the result to ensure reliable and stable releases. Continuous delivery creates a release process that is reliable, repeatable, and predictable [1, 4, 20]. The build process, scripts, and other automation tasks are tested repeatedly before deployment to production. Most errors and problems will be found early, before the actual release, securing the proper release [20].

4.1.2 Reducing Defects

Defects (e.g. bugs and errors) found by stakeholders can be reduced since the software will be tested iteratively before release. The final deployment into production is being rehearsed every single time new code is merged to the project [1]. The number of code changes in each release decreases with more frequent releases. This results in defects being found faster and much easier for each separate release. Continuous delivery enables automatically roll back to an older release if a new one fails. The number of open defects has been shown to decrease and improve the product quality [20]. The number of reported defects by customers are likely to level out with the workload to fix them for each release [6].

4.1.3 Improved Efficiency and Productivity

With continuous delivery comes the ability to automatically set up the environments and test the code of the product. There is no need for each developer to set this up themselves. This means that the developers can have more focus on the code. The total effort to release a software to production will decrease [4, 20]. Deployment of a software normally takes several days
doing it manually [26]. There will be a flexibility in the deployment since it will be easy to automate the process to install and prepare for a new environment [1].

4.1.4 Shorter Time to Market
Features and bug fixes can be delivered to users much faster [1]. If the release frequency increases, it gets easier to build the “Right product”. This because the business value to users is delivered more quickly. This capability helps the company to stay one step ahead of the competition and in the end improving the customer satisfaction [20].

Proceeding from continuous delivery to the next step, to continuous deployment, would enable continuous customer feedback and the ability to learn more from customer usage data. This also eliminates any work that does not produce any value [3]. The ability to deliver frequently has long been recognized and used in the open source software communities as a good practice. In the end, releasing often has a lot to do with the ”lean” way of thinking [6].

4.1.5 Empowering Teams
Continuous delivery can change the way developer teams works, to the better, e.g. they feel empowered and produce better results. Testers, operation, and support staff can easily deploy a released version of the software to start their work, e.g. for conducting acceptance testing. Functioning releases can be installed at the push of a button. This enables teams to collaborate and work more effectively [1]. It has been shown for instance that closer connection between the development and operations people can result in better discussions and improved productivity [4].

4.1.6 Lowering Stress
Releases into production are traditionally big events. Continuous delivery will, in theory, enable release of an already configured software, just with one command [26]. The stress in all parties associated with a release of a software will then be reduced [1]. Interviews with developers have shown a lower level of stress when using continuous delivery [20].

4.2 Obstacles (RQ2)
The purpose of this section is to define obstacles for organizations when they introduce continuous delivery. It is also needed in the comparison against the result of the multiple-case study.
4.2.1 Lack of Test Automation

Certain things, such as user interfaces, hardware, and multiplatform systems can be difficult to automate and test [5]. The lack of automated user interface testing is, in fact, one of the biggest challenges [12]. Furthermore, there are problems with ambiguous test result, flaky test cases, and time-consuming testing [5]. Then there is exploratory testing that requires human intervention and this type of testing cannot be automated [1].

Feedback loops from the automated regression tests cannot be too long. If builds take too long, it can, for instance, lead to larger commits from the software developers. Larger commits which contain multiple changes and conflicts with changes made by others [5].

4.2.2 Complex Environments and Technical Challenges

Dealing with software not amenable to continuous delivery (e.g. large, monolithic software) is challenging. An off-the-shelf, highly customizable, and comprehensive solution for continuous delivery might not yet exist. This means that the companies develop their own solutions, which in turn can take time to maintain [5, 20].

Build systems and scripts are often complicated. They cannot be modified easily and might not be flexible [5]. When moving further to continuous deployment, can different network configurations at customer sites be a problem. There may be configurations that cannot simply be fully counted for [3].

4.2.3 Human, Organizational and Resource Problems

An obstacle for continuous delivery can be human and organizational problems like lack of discipline, lack of motivation and lack of experience [5]. Simply a resistance from humans to change [4]. An organizational challenge might be a feature ready for release, that must go through a change advisory board [20]. Some companies have mentioned a barrier and lack of transparency when moving closer towards continuous deployment of a software. Furthermore, the internal verification loop needs to be shortened to develop functionality faster [3].

Initially setting up continuous delivery in an organization requires effort and resources. Organizations can have limited hardware resources and bad network latencies, leading to larger commits from the developers [5]. It can take months of work to get everything automated and the deployment process streamlined [27].

Organizations will need to start with a "lean" mind-set before moving to continuous delivery. It can be difficult to work with many small releases and
prevent half-done features being delivered. Changing the responsibilities needed, can be an obstacle and increase pressure on some roles within the organization. The teams must most likely work extremely efficiently to achieve continuous delivery [28].

4.3 Best Practices (RQ3)

The purpose of this section is to define best practices and principles, from the literature, to follow regarding continuous delivery. Organizations could use the best practices when introducing continuous delivery and it could help them address common problems. Best practices are important to understand before investigating the obstacles in a case study since the answer could already be in the literature.

4.3.1 Create a Reliable and Repeatable Process

One of the main goals of continuous delivery is to make releases easy to perform. The repeatability of the process can be easy if every single part of the process has been tested many times before. For the process to be reliable, means that changes should not be checked in on a broken build [1]. For software developers, the need to commit code frequently is important. Broken and half done code should not be merged. Broken (red) builds should be immediately fixed and kept working (green). Getting it back to green should be the highest priority [19, 29]. Test cases should be written in an automated way [29]. In fact, automate as much as possibly [1]. All tests and inspections must pass before the code can be merged into the code base [29]. If releasing the software is painful, the aim should be to release it as often as possible [1].

4.3.2 Use Automation

Human error is more likely when judgements must be made within the delivery steps. In fact, the whole delivery process might be at risk if being dependent on certain persons. It is common for incidents to occur by manual configuration mistakes [1, 20]. As much as possible should be automated to save time, increase the repeatability and reliability. If constantly going through the release process, will manual steps be daunting tasks, but not if automated. To enable release via the push of a button, automation is the only way to go. Automate test cases will also let developers spend time on other tasks while their commits get tested automatically [1].

The automated builds should not only be compiling the software, but also verify the functionality of the code, e.g. include a thorough test suite, unit, and integration tests. A good practice is to measure and increase the test coverage [19]. So, the build might involve generating a code coverage report,
run the integration tests and various static code analyses. The build tools typically upload the built artifacts in the end to a repository that manages them for deployment or distribution [20].

4.3.3 Use Version Control

As much of the parts of the project should be kept in some sort of VCS [1, 19, 26]. It makes it possible to go back in history if something is not working. Password or other sensitive data should however not be stored in the VCS, instead rather in some other location on the file system in an encrypted form. Most of the systems have the option to add a description and other meaningful information to the commit. These commit messages are important since it could save hours of debugging where the message can be used to find out who wrote the code and why. The version should be identifiable in the VCS for any given build release. This makes it possible for stakeholders, such as testers or customers to use specific versions of the software [1].

4.3.4 Share Responsible and Build in Quality

Everyone in the organization is responsible for the delivery process and the quality. Defects are cheaper to fix, the earlier they are found. They are even cheaper to fix if they are not checked in to the VCS in the first place at all. To achieve the built-in quality is test driven development a choose. In fact, a comprehensive test with high code coverage is necessary to fulfil continuous delivery. This will also mean that failing test cases should never be disabled [1]. Another way to build quality in is having other developers reviewing the code in addition to static program analysis tools [12].

4.3.5 Release Frequently

Integration and release processes are documented as painful or stressful. By doing it often will bring the pain forward, dealing with the problems early and where they are fewer in number. In opposite to dealing with everything at the end of a project. Every time the release process is repeated will make it better. A good thing to follow is to make the definition of done equals released. E.g. A feature is only done when it is delivering value to users or stakeholders [1]. Build process might need to be shortened so that tests can be run faster and development needs to be modularized into small units [3].
5 Multiple-case Study Result

This section contains the evaluated results of the multiple-case study. Evaluated means in this case that the sections have been sent to the participants for comments, and if so, then corrected thereafter.

5.1 Company A

5.1.1 Background

The company referred to as Company A is part of the telecom industry. The participant from this company is working with a new product within the BSS (Business Support Systems) domain. Continuous delivery has been used within the products organization for some years.

The interview was performed with a test and quality architect that is responsible for securing an efficient, stable, and reliable framework for continuous integration and delivery. The participant has a high experience and a deep knowledge of continuous delivery. The interview took around 40 minutes and was performed over the phone. As a complement were the interview questions also answered by the participant beforehand via email.

5.1.2 Delivery Setup

Six people in total are fully dedicated to working with the products continuous integration and the delivery process. They consist of one release technician, two build masters, two continuous integration "guardians", and the participant himself. Many more people, e.g. developers, are involved but not fully dedicated. On average, releases/deliveries (e.g. release of software) are performed two times per week to internal stakeholders. This includes new packages, repack of releases or emergency packages. The number of deliveries depends on how many active branches there exist. The time to deliver takes approximately one hour, from the start of the build to that the release mail has been sent, assuming regression tests are working. The time can be extended if some last changes need to be merged before delivery. As tools and systems are the company, for example, using VMware, Git, Gerrit, Maven, Jenkins and JFrog Artifactory. Besides this are they using in-house/self-developed tools and frameworks for deployment, configuration and testing of the products.

5.1.3 Benefits

The participant perceives continuous delivery as the ability to deliver the products with known quality and content at any point in time. To achieve this, is a repeatable process for building, testing, releasing and deploying a requirement. In most successful cases, is this requirement met through
automation. So, a major focus is put on removing the human factor. The main reason to use continuous delivery is a very high demand on functional growth while keeping the quality and delivery precision. Continuous delivery allows the ability to always know the current state and content of the product.

5.1.4 Obstacles

Obstacles before delivery can be to keep the feedback loops short and reliable for the developers. Short loops will let the developers act fast on issues arising, as well as keeping deployment and regression tests "green". Developers must respect the feedback they receive from the builds. If not, it can go so far that the building is blocked. This places a lot of responsibility on the developers. In other words, a lot of technical and way of working challenges. During the delivery can environment instability sometimes block or interrupt the actual delivery procedure, forcing re-runs which result in wasted time. After delivery is the system not yet mature enough to support a full one-branch with continuous delivery, so release branches (e.g. multiple branches) for corrections and repacks need to be maintained. This puts additional resource requirements on both hardware for automation and developers.

5.1.5 Solutions and Best Practices

To ease obstacles, it is important that frameworks should be stable and perform without suspect behaviors. Furthermore, should automation, feedback, reliability, and execution time be invested in the project from the start. The time and money it saves are most often easily justified over almost any projects period. Some of the obstacles can be a result of that the developers not always understand the importance of testing from the beginning. Especially when it comes to graphical user interfaces where UX (user experience) will need to be considered. So, one best practice is to not neglect the testability of the product and keep it in the scope for all functionality that is being developed. If the functionality cannot be tested completely and reliably, there will be issues with stability or quality in the context of continuous delivery. There is a high-level strategy in the company to adhere to and certainly have many studies been done according to the participant. Mostly is it the fact of testability that appears from these studies. TDD could be one solution. But when there is a time pressure, testing is often down prioritized. Functionality comes in front of quality. Trying to buy quality afterwards can be both expensive and difficult.
5.2 Company B

5.2.1 Background

The company referred to as Company B is smaller software provider for the construction industry. The company has a minor experience of continuous delivery. They just recently, during the work of this master thesis, started an attempt to introduce continuous delivery in their projects.

The participant is a team leader of a development team. This means that he is, among other things, a type of technical manager and he determines which tools and frameworks to be used. The team is, for example, developing a web-based product that calculates the energy performance of buildings. The interview took around 60 minutes and was performed at the company’s office.

5.2.2 Delivery Setup

Nobody is fully dedicated to working with the products continuous integration and delivery process. Instead are every developer responsible for handling their own delivery and deployment. Some are doing it semi-automated, but it is not a company requirement. Others build the software locally and deploy it manually. However, there is a desire to automate more. An external delivery (e.g. release of software) is on average performed each month to customers, but it varies. Daily deliveries are performed internal, especially when an external delivery is expected. The actual delivery time, e.g. starting the build and deploy, is approximately 15 minutes. The company is using AWS to maintain and conduct continuous delivery. The goal is to use as much as possible of AWS. Some other open source and commercial systems are used for testing. In the case of some computational components in the products, have they built their own test systems.

5.2.3 Benefits

The participant perceives continuous delivery as the ability to get functionality out quickly, not just to customers, but also for internal testing. Furthermore, it gives a smooth flow (pipeline) and automation. There should be no need to perform manual steps before each delivery. The expected benefits are that it should be quick to deliver new features and provide a good method for quality assurance. Furthermore, remove the human factor. Developers should be able to make a change in the software somewhere without breaking something in another part, without recurring errors and destroying the old functionality. Developers should not have to deploy manually. This because they can forget something or make mistakes. Developers should focus on developing features.
5.2.4 Obstacles

Before a delivery, it can take time to set up the system used for the delivery. Everybody does it differently. The delivery process itself is not clearly defined. During the delivery can something suddenly go wrong, this often has something to do with dependencies. To get started with a process that works and is comprehensive can be difficult. It takes time. Furthermore, what to do if something goes wrong and to know what caused the error. AWS does a rollback, but that might not be the best solution. Maybe it is not a good idea that every developer is involved in the delivery process. After delivery is it mostly testing obstacles emerging. For example, much graphics (e.g. graphical user interfaces) are difficult to automate tests for and to run beforehand. The last bit of testing, end-to-end, is difficult without mocked data and real users.

5.2.5 Solutions and Best Practices

Good and simple tools are considered to ease the obstacles of delivering according to the participant. There exist many tools, but they often solve specific problems. One system, where everything needed can be accomplished is to prefer. Other desires are systems with a clear overview, visualized flow, and transparency. A system that helps you to get started with continuous delivery are desired. It may also be good to not have to deal with physical servers because it could create problems with too little hardware resources. AWS takes care of physical servers in the case for this company. In general, is it useful to keep it simple and not to develop large testing systems that need to be maintained. The testing itself is, however, one key to success. Developers should focus on creating functionality and submitting the code to the repositories. Then leave it to the automated process. Otherwise, there may be a need to have people who are working full time with continuous delivery.

5.3 Company C

5.3.1 Background

The company referred to as Company C is part of the defence and security industry but is also involved in other fields. In addition to software is the company also working with for example custom hardware.

The interview was performed with an integration and test manager working with a product in the aeronautic area. The interview took around 30 minutes and was performed over the phone.
5.3.2 Delivery Setup

Ten people in total are in some way involved and dedicated to work with the products continuous integration and delivery process. A new delivery (e.g. release of software) to internal stakeholders is performed whenever someone changes the software, in other words, almost daily. However, delivery to external stakeholders, e.g. end customers, are not performed as often. It is not considered necessary to deliver to end customers constantly, partly since the project is lasting over several years. The time to perform a delivery depends on what is changed in the product compared to the previous delivery. E.g. there are different types of deliveries. Because some parts of the system might need to be rebuild, but other parts with no changes have no need to be rebuild and can instead be reused. Depending on this can the time for a delivery take between minutes up to several hours. The company is using many different systems to maintain and conduct continuous delivery, such as Jenkins and Git. It is, in general, a combination of common commercial or open source systems (the same as many other companies are using), simulation systems, and self-developed systems.

5.3.3 Benefits

The participant interprets continuous delivery as the ability to create a release candidate whenever the company wants it, but not necessarily to the end customer. The release candidate should be functional and runnable. Furthermore, continuous delivery is interpreted to provide a pipeline which can produce a working software every day. The benefits for using continuous delivery are that it is a prerequisite for getting a large and very complex system to work during development. Since complex systems are difficult to specify beforehand and hard to put together. There is a need to take the development in steps, e.g. small iterations. Continuous delivery is a support for iterative development and this is an established fact since way back in time, according to the participant.

5.3.4 Obstacles

Obstacles mentioned in the process before delivery were technically complex connections in the architecture and in different parts of the system. A change in one part may affect another part because they depend on each other. Developers need to synchronize between each other before changes can be performed. Somebody needs to change something first, before someone else can change their parts. This creates dependence obstacles between different developers. Access to special test resources and test facilities can be an obstacle. As well as special hardware that must be tested before delivering. During the delivery can tools that are not simple or fast be an obstacle. How to set up the pipeline correct is another. After delivery is the stability in the
test environment an obstacle. Tests are being run thru the whole process, but the majority of the tests are run after delivery because it is such a large and complex system.

5.3.5 Solutions and Best Practices

To ease some of the obstacles regarding continuous delivery, according to the participant, is one option to find and use easier tools. Another is to make the test systems more stable. Furthermore, decouple the systems, work long-term and identify the problems early. Move restrictions in the system further away and work on them continuously. It is an advantage to have a high level of knowledge about large complex systems in general and to understand the concepts about continuous delivery from the literature. In other words, a combination of domain knowledge (of the system being built) and continuous delivery knowledge. Custom made hardware that is used will also need to be carefully considered in the process.

5.4 Company D

5.4.1 Background

The company referred to as Company D are developing and supporting software products that allow organizations to plan, manage and train their staff, e.g. HR and salary systems. The company has a relatively small organization but they have large customers. The products are partly developed using Java and JavaScript.

The interview was performed with a software developer that is part of a team that deals with the integration and migration between their HR systems and the customers own systems. The interview took around 60 minutes and was performed over phone and online chat.

5.4.2 Delivery Setup

Nobody at the company is fully dedicated to working with the products continuous integration and delivery process. But part-time is there mainly three people working with the process. One person focuses on the administrative part, e.g. what should be included in the delivery. Then there are two people who work with the building and automation environments. One delivery is performed at the end of each sprint, that means every four weeks in this case. The deliveries are always to external customers. The deliveries can be retrieved via a website and then the customers can choose the time to deploy the deliveries by themselves. The time to deliver takes approximately eight hours (one working day). As tools and systems for continuous delivery are the company using Jenkins and Maven to a great degree. Other
than this is the company, for example, using Grunt, Evry (for hosting the final deliveries), Jira, Git, and Bitbucket.

5.4.3 Benefits

The participant perceives continuous delivery as the use of sprint based deliveries, e.g. the result of using Scrum or Kanban. In other words, small and many releases while the product is being iteratively developed. Furthermore, he sees continuous delivery as the process of developing code, test, deliver and deploy in short periods of time. The benefits of this are that the deliveries and updates can be performed more quickly. It also reduces the problems and work of each delivery.

5.4.4 Obstacles

Obstacles mentioned in the process before delivery were that it can be difficult to keep track on what should be included in the delivery and to know if everything is ready. It can also be difficult to get all the builds (in Jenkins) to go green and know if all the functionality is tested. There are many manual end-to-end tests that need to be performed as well. However, since delivery is done every sprint, it might not matter if something is missing. During delivery, one obstacle is that everything in the process is not automated. This means that humans must always be involved. If something then goes wrong, it may be difficult to find the right resources (people) to solve the problem. Another problem is that it can be discovered that something is not included, and then the delivery needs to be started from the very beginning, which takes time. Furthermore, there is no clear code freeze, e.g. changes to the source code are not stopped during the delivery, which could result in problems during the release. Obstacles after delivery can be to know what the delivery really contains, since there is no clear changelog. If something were missing then everything must be rebuilt, which again takes time. There is also a special branch with all the fixes/patches that are hard to keep track of. As well as different branches for different customers and versions of the products.

5.4.5 Solutions and Best Practices

To ease the mentioned obstacles is the participant saying that the systems should be modularized more to be able to deliver smaller parts. It should be a general best practice to enable small releases and not sending out large files to customers. Maybe even deploy/host everything within the own company, instead of delivering externally and letting the customers deploy themselves. Some customers do not want many updates because it requires resources from them to deploy it. With internal or in-house deployment would it be easier to get every customer to run on the same version, e.g.
the latest, and decrease the number of branches to maintain. One last best practice is to automate more, especially test cases, which makes it possible to build easy and to know that the delivery is reliable.
6 Development and Fieldwork Result

A walkthrough of the delivery system used by Company B was performed after the interview. A period of development and fieldwork was then spent on investigating how to improve the delivery system and understand the involved technologies.

The products managed by the participant from Company B, are in general full stack JavaScript applications. In this context, full stack means that the same language or technology is used on both the frontend and the backend. React.js, Material-UI, and Babel are used as the main JavaScript libraries. MongoDB is chosen as the database. Node.js is used as the automation and dependency manager tool. Node.js is also a JavaScript run-time environment for executing JavaScript code on the backend.

The company is currently using Elastic Beanstalk on AWS for their deployment of applications. This means AWS takes care of provisioning the virtual web servers. But it is not triggered automatically and normally the participant builds locally and uploads the generated package. So, from a continuous delivery perspective could this be improved in many ways. After an investigation of the different services provided by AWS, was CodeStar chosen for the fieldwork. CodeStar combines the functionality of many services and offers several overviews, e.g. build logs and integration with Jira (a bug tracking system). CodeStar was also chosen since the participant of Company B has not yet tested it but stated that it is a very interesting and new service. To access all the provided services from AWS can the user login via the AWS Console website. The services can then be started in different regions, based on countries (e.g. Ireland, USA, etc.). In this way can different services be started without interfering with each other. When using CodeStar, a template is used at the beginning, which is different depending on the project’s needs. The templates help the user to setup and configure the delivery system. A Git repository is then created (or connected to) at the start and hosted on AWS in a service called CodeCommit, where developers can integrate their code. When changes occur to CodeCommit, are automated builds triggered in CodeBuild. CodeBuild could be likened to an automation server. When a build is complete, is the service CloudFormation used to deploy the changes automatically. The setup is viewed in the diagram in Figure 6.
The example setup used in this fieldwork is created with a serverless architecture using Lambda on AWS. This is a long-term goal according to the participant. With Lambda can the stakeholders ignore where the code is running and there is no billing for idle servers when the code is not running or used. There is no need to manage servers or hardware infrastructure on your own.

Regarding the deployment of the applications, AWS seems to have two main choices that they call Elastic Beanstalk and EC2 (Elastic Compute Cloud). There are more however, but most of them are using each other in different ways. Elastic Beanstalk can deploy one single application quickly with predefined configuration and AWS will take care of most of the work, such as auto scaling and load balancer. But it is ineffective when dealing with deployment for multiple applications. Since it basically will be created one new machine/server for each application. Elastic Beanstalk will, just as CloudFormation abstract away technical choices that exist in EC2. EC2 provides more flexibility and is more powerful. In EC2 could the idea be to create a customized cluster of servers and upload multiple applications without starting a new server for each application. Then, for example, can Docker be used to run and manage applications in isolated containers. Docker would give self-contained systems, with libraries and settings, that will always run the same regardless of where it is deployed. Lambda on the other hand that can be used in CloudFormation, could replace a lot of this server management.

The setup and configuration of the delivery system are somewhat different
than using many standalone systems. The configuration can be set at the highest level using YAML to tell AWS which commands to run. Then on a lower level can configuration for the projects be set, depending on the type of project. In this case is it a JSON package file for Node.js. All this configuration is stored in the Git repository.

One problem encountered during the fieldwork was to install Jest, which is the JavaScript testing framework used by Company B. It should be installed via NPM that comes bundled with Node.js. It is needed to run the automated test cases in CodeBuild. It is most likely a dependency issue but a longer period of troubleshooting will be needed to investigate this problem. But this highlights the possible problems with off-the-shelf solutions and testing. How the deployment in CloudFormation is handled is somewhat hard to understand (other services might help with that). Thus, that could also be a benefit, e.g. the developers concentrate on code and AWS handles the virtual servers and deployment. Understanding the names of the different services and understand what they do, when being new to AWS, is also challenging.
7 Discussion and Future Work

This master thesis reviewed some of the continuous delivery literature within a given timespan. Future work could be to conduct a more comprehensive literature review and analyzing the findings deeper. There is some confusion between the meanings of different "continuous" activities. Which is understandable since they somewhat overlap and build upon each other. This makes it more difficult to find related work. A multiple-case study using interviews was then conducted and this resulted in new data from the industry. Fieldwork was also conducted to give a different type of data and to understand an off-the-shelf continuous delivery system in practice. Ideally, should fieldwork have been done at all the involved companies. Interviews with more people or companies could also have been performed. However, it is questionable how much more knowledge it would provide regarding the qualitative aspect of this master thesis. Truly interesting would an investigation of a company that exists on step E in the "stairway to heaven" be to conduct, see Figure 1. When moving from step D to the highest step E, are the barriers still to be explored stated by Olsson et al. [3].

7.1 The Stairway

When looking at the "stairway to heaven", could Company B be placed at step B and close to step C in the stairway. Since they are working somewhat agile, integrates code frequently, often deliver, and using local builds with unit tests. But running builds is not a mandatory requirement and is not triggered when pushing code to the repositories. The reason may partly be that the developers work individually and feel safe with their changes without automated builds. Hence some more works need to be done for the company to truly be at step C. Despite this, they have a process that can be likened to continuous delivery. The results from the fieldwork is one direction for Company B to go, but the results also show possible limitations of AWS.

When looking at Company A, C, and D, could they be considered to fully fulfil step C in the stairway and heading very close to fulfil step D. Continuous integration is indeed a well-established practice for them. But Company A and C are, from this master thesis perspective, not delivering (and deploying) to external customers continuously. Though they might deliver to customers more often than they did before embracing the process of continuous delivery. The reason for the low deliver frequency could partly depend on the fact that the customers to company A and C do not see the need for more frequently deliveries. However, deliveries are performed continuously to internal stakeholders. Hence here could step D be considered to be fulfilled. But testing (some of it manual) is brought up as cumbersome and time-consuming. Much testing also takes place after the internal deliveries
according to Company C. In contrast to company D who does all testing before deliveries. Furthermore, some functionality is difficult to automate tests cases for. So even though it is not clearly stated in the interviews by all companies, could it be assumed that much manual testing is performed before delivering something to external customers. If an organization needs a long period of manual testing before being able to deliver, with known quality, then step D could be considered not to be fulfilled. It is at least not the truly meaning of continuous deployment, where the whole process is automated end-to-end. More about testing is discussed later in this chapter.

7.2 RQ1

Based on the results in this master thesis, is continuous delivery a process with many benefits. It is not however without obstacles and challenges. The identified benefits are clear and no obvious gap has been found. But further research could be done since there may be other benefits. The view on continuous delivery is positive both in the reviewed literature and by the interviewed companies.

Benefits are among others, reliable releases, shorter time to market and reduced defects. In the end, improved efficiency, and productivity. Benefits such as empowering teams and lowered stress found in the literature, were not explicitly found by the case studies. Other interview questions could possibly have been used to discover this. But there could also be an implicit link between reliable releases and lowered stress.

7.3 RQ2 and RQ3

RQ2 and RQ3 are discussed together because they somewhat commingle, since best practices could be solutions to the obstacles.

Complex environments, technical challenges and lack of test automation are found as the main obstacles. The build of the deliveries itself seem to be able to perform quickly for the companies, instead it is the testing that takes time. Without testing, will the delivery not have a known quality. Both Company A and B states that testing graphical user interfaces are challenging. Company A state that testing often is down prioritized, which might be a human or organizational problem. Company D state that more test automation is needed. The lack of test automation might be improved by always using TDD (test-driven development) as best practice [1]. In TDD, the programmer first designs the test cases, before writing the actual implementation. Regression testing gets easier to run since test cases will be automated using TDD. TDD is showing tendencies to reduce the defect rate and improve code quality, however, it is still debated. Both TDD and continuous integration, as well as many other good practices, is part of
the XP (Extreme Programming) methodology [2]. Truly adapting to XP or TDD, even when it comes to graphical user interfaces or anything else, could solve the lack of test automation. Tools like for instance Selenium could be used in an automated way to test web-based user interfaces end-to-end [30]. However, this might add issues of maintainability of the test cases and stability issues. Frameworks should not be used if they have bad testability or if they are too complex. There might be a market for new frameworks with good testability characteristics. Further investigation could be why more organizations are not strictly following TDD or XP. The answer may be that functionality is the highest priority, as stated by Company A. Company C states that special test resources, test facilities, and custom hardware could be obstacles. Company A talks about multiple branches that put additional resource requirements on both the hardware for automation and the developers. This seems aligned with the literature review results. Regarding hardware resource problems is AWS the solution for Company B. Resource problems could be assumed to have with financial questions to do. But further research can be conducted. It could also be assumed that security is important for Company C because of their industry orientation. However, security and privacy issues related to continuous delivery have not been discussed in this master thesis and could be interesting for future work. Company D seems to have more administrative obstacles then the other companies, e.g. what should be included in the delivery and to have the right peoples available when needed. This fact could be aligned with the human and organizational problems found in the literature review. This specific finding could depend on that the interview was performed with a software developer, hence the person is not as high up in the organization as the participants from the other companies. A solution for the company might be to improve the communication and share the responsibility for the deliveries with everyone in the development organization.

The access to more effective and easier tools are brought up by all companies, except Company D, as solutions to some obstacles. However, in the literature are tools not discussed as much. Solutions and best practices are also more abstract in the literature. For example, is version control often discussed in the literature, but this is a very established practice according to the case studies. Version control is something needed even before starting with continuous integration. Regarding easier tools could AWS be one solution. However, it should be clearly stated from this master thesis standpoint, that AWS is not a recommended chose for everyone. Only one company is using AWS in this thesis, so more research is needed. But it is one alternative that could suit some organizations. Costs have not been considered a variable in the fieldwork and every organization needs should be evaluated beforehand. Only maintaining one branch or one version of the product seems to be best practice. Best practices can in addition to
this also be summarized with always using a "lean" mind-set and follow the principles behind the Agile Manifesto [31]. Such as, work iteratively and release working software frequently. Invest in automation, good design, flexible architecture, quality and testing from the beginning of the project.
8 Conclusion

In this master thesis has continuous delivery been investigated from a software engineering perspective. Continuous delivery is viewed in general as a process with many visible benefits, such as shorter time to market and improved productivity. But organizations are struggling with some obstacles, such as complex environments, multiple versions to maintain, lack of test automation, and organizational problems. The problems seem particularly challenging when it comes to testing web-based user interfaces and custom hardware. Best practices regarding continuous delivery are well defined and help to reduce the obstacles. But they can, on the other hand, be a bit abstract. Some of the best practices are to invest in automation and version control as much as possible. However, there still seems to be a research gap concerning the obstacles and organizations have a problem to always be ready to deliver functioning software. Some suggestions for best practices or solutions to the obstacles can be easier tools and higher priority on testing. There are many accepted methods and principles out there that could help if they were followed, such as the Agile Manifesto, Scrum, TDD, and XP. To use other frameworks or an off-the-shelf solution like AWS could be another solution for some organizations. The results from this master thesis might be used to reach the highest step in the "stairway to heaven". Further research can be to investigate deeper if there are anything that can tackle the identified obstacles.
9 References


10 Appendix

10.1 Interview Questions

<table>
<thead>
<tr>
<th>ID</th>
<th>Related RQ</th>
<th>Type</th>
<th>Question</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>All</td>
<td>Close-ended</td>
<td>What role do you have in the organization you are working for?</td>
</tr>
<tr>
<td>2</td>
<td>All</td>
<td>Open-ended</td>
<td>How do you perceive or interpret continuous delivery?</td>
</tr>
<tr>
<td>3</td>
<td>All</td>
<td>Close-ended</td>
<td>How many people in your organization are dedicated to work with the continuous integration and delivery process?</td>
</tr>
<tr>
<td>4</td>
<td>All</td>
<td>Close-ended</td>
<td>How often (daily, weekly, monthly) does your organization deliver to stakeholders, e.g. perform a release of software or artefact?</td>
</tr>
<tr>
<td>5</td>
<td>All</td>
<td>Close-ended</td>
<td>How long does the delivery process usually take to perform?</td>
</tr>
<tr>
<td>6</td>
<td>RQ1</td>
<td>Open-ended</td>
<td>What are the reasons or benefits for your organization to use continuous delivery?</td>
</tr>
<tr>
<td>7</td>
<td>RQ2</td>
<td>Open-ended</td>
<td>What are the obstacles in the process <em>before</em> the delivery?</td>
</tr>
<tr>
<td>8</td>
<td>RQ2</td>
<td>Open-ended</td>
<td>What are the obstacles in the process <em>during</em> the actual delivery?</td>
</tr>
<tr>
<td>9</td>
<td>RQ2</td>
<td>Open-ended</td>
<td>What are the obstacles in the process <em>after</em> the delivery?</td>
</tr>
<tr>
<td>10</td>
<td>RQ2</td>
<td>Open-ended</td>
<td>Do you have any solutions for how to solve or ease the obstacles regarding continuous delivery?</td>
</tr>
<tr>
<td>11</td>
<td>RQ3</td>
<td>Open-ended</td>
<td>Do you have any best practices or advices for conducting continuous delivery?</td>
</tr>
<tr>
<td>12</td>
<td>RQ3</td>
<td>Open-ended</td>
<td>What different systems does your organization use to maintain or conduct continuous delivery?</td>
</tr>
</tbody>
</table>