CHALMERS

Exploring practices to implement continuous deployment

A case study in the telecommunication industry

Master of Science Thesis
in the Management and Economics of Innovation Programme

JAKOB DAHLGREN
OLA EDMUNDSSON

Department of Technology Management and Economics
Division of Entrepreneurship and Strategy
CHALMERS UNIVERSITY OF TECHNOLOGY
Göteborg, Sverige 2016
Report No. E 2016:081

MASTER'S THESIS E 2016:081

Exploring practices to implement continuous deployment

A case study in the telecommunication industry

JAKOB DAHLGREN
OLA EDMUNDSSON

Tutor, Chalmers: Marouane Bousfiha

Department of Technology Management and Economics
Division of Entrepreneurship and Strategy
CHALMERS UNIVERSITY OF TECHNOLOGY
Göteborg, Sweden 2016

Exploring practices to implement continuous deployment
A case study in the telecommunication industry

JAKOB DAHLGREN

OLA EDMUNDSSON

Master's Thesis E 2016:081

Department of Technology Management and Economics
Division of Entrepreneurship and Strategy
Chalmers University of Technology
SE-412 96 Göteborg, Sweden
Telephone: + 46 (0)31-772 1000

Chalmers Reproservice
Göteborg, Sweden 2016

Abstract

Accelerated advancement in technology is changing how companies are developing and deploying software. A rising concept within software development is continuous deployment (CD), in which new functionality is continuously released to customer. The new practice is associated with several benefits, however, there are not many companies that have fully succeeded in adopting CD. Instead, barriers exist, which are especially severe in the B2B segment. Research is consequently required in order to understand these challenges and develop practices for how to effectively manage them.

The purpose of this study was to investigate how a telecommunication company can overcome their specific challenges when adopting CD. To fulfil the purpose, the study includes a literature review of previous research on barriers related to the transformation. Moreover, it contains an empirical study, exploring three internal organizations and their efforts to adopt CD.

In the study, 35 barriers to adopting CD are identified and further divided into five categories: *Network diversity, Continuous Integration, Deployment, Company Commitment and Customer Resistance.* Specific for the telecommunication industry, are the technical requirements in building automation on mature products and the diversity in customer networks. There also exist major difficulties in achieving seamless updates, due to the network setups. Besides from the technical issues, organization blockers were found to hinder an adoption. To mitigate these barriers and successfully implement CD, a gradual adoption including three main stages is suggested. These steps include organizational anchoring, implementing continuous integration and preparation for frequent deployments.

Keywords: continuous delivery, continuous deployment, continuous integration, telecommunication, agile software development, change management, software upgrade

Acknowledgement

This report was the final work before graduating from the master program *Management and Economics of innovation* at Chalmers University of Technology. Working with this has been challenging, mainly because of the highly technical orientation, which has meant that we have learnt a lot. These learnings from software development processes will surely be beneficial in our future careers.

We have had fun pushing ourselves to dig deep and gain understanding, while we must admit that much credit should be given to colleagues at the case company. It has been done in collaboration with a telecommunication firm, and for that we would like to extend our gratitude. Thanks to our supervisors at the company, but also to everyone who was involved in interviews or in other ways contributed to the data collection and analysis that formed this paper. We also wish to acknowledge our supervisor at Chalmers University of Technology, Marouane Bousfiha, who has provided great support throughout this research.

Glossary

BRT - Basic regression test is an automated test on the software track

CD - Continuous Deployment

CI - Continuous Integration

GA - General availability of a software version, meaning ready for global sales and deployment

I&V - Integration and verification

LFD - Last feature drop, which is the latest date that a new feature can be delivered

LLV - Latest local version, a team branch were the team is committing their change before integrating to LSV

LSV - Latest system version of the software, containing the latest changes

SAT/XSAT – Automated system acceptance test

MMF - Minimum marketable feature

PDU - Product development unit

PL - Product lead unit

PRA - Preliminary availability of a software version

SAT - System acceptance test that decides whether a new build is accepted into the software track (the LSV)

SBL - Software baseline, an LSV candidate

Seamless upgrades - Upgrades that does not interfere with other activities or negatively affect network performance

TR - Trouble report, registration of faults.

Unit test - Early testing of a small part of the larger application

XFT - Cross functional team

Table of content

1 Introduction

The introduction chapter describes the background and research context, as well as the formulation of the research problem. It further defines the purpose and research questions of the study.

1.1 Background

Accelerated advancement in technology has created a turbulent business environment for companies involved in software development. Consequently, software development projects are nowadays commonly associated with fast-changing and unpredictable markets, complex and changing customer requirements and pressure of shorter time-to-market (Holmström Olsson et al., 2012). As a reaction to these changes, agile development methods have been advocated by most software practitioners. Agile emphasizes intense collaboration and communication within the team, as well as with the customer. It consists of short and fixed periods of development, integration and testing in small teams (Berger and Eklund, 2015). Agile is nowadays widely used among businesses, especially in the software development community in which it originated and evolved, but increasingly in other areas as well.

As described in the previous paragraph, several software companies have successfully integrated agile practices in parts of their organizations. However, Claps et al. (2015) argue that there is not many that have succeeded in adapting to continuous deployment (CD). This refers to practices in which software functionality can be continuously deployed to customers, so that feedback and usage data can be utilized throughout development, delivery and deployment of the software. Reported benefits with this approach is faster feedback, more frequent releases, higher customer satisfaction, improved quality and productivity, effort savings, reduced risk for each release and reduced development of wasted software (Claps et al, 2015, Lepännen et al., 2015 and Liaping 2015). This relatively unexplored research topic is, however, also connected with severe challenges for practitioners. Olsson et al. (2012) have explored the barriers in the transition from agile development towards CD and developed a conceptual model called “stairway to heaven”. The major barriers in adopting CD were different network configurations at the customer sites, the need of a shorter internal verification loop and lack of transparency. In an exploratory study, Claps el al. (2015) identified 20 challenges in adopting to a CD process. These challenges were categorized into either technical or social issues, and suggestions on how to mitigate some of them were made. In a similar study, the challenges facing a B2B software company were divided in into technical, procedures and customer related problems (Rissanen and Munch, 2015). Moreover, by interviewing 15 different tech companies, Leppänen et al. (2015) found that the majority of companies decided not to make a full adoption to CD. Not all companies were prepared to abandon the level of control and human decision making, which was required by the automatic software deployment. Instead they were more inclined to proceed with familiar release cycles, especially if the perceived benefits of CD were not as strong as the effort necessary for changing the development practices. Even if the topic is relatively new in academia, there is a consensus that CD presents benefits, but is also connected with major challenges for the developing company.

Previous research is valuable for organizations because it can prepare them for the transition and can be used as a systematic approach when identifying the challenges within every specific company (Claps et al., 2015 and Rissanen and Münch, 2015). However, besides from Claps et al. (2015), limited studies have investigated how these challenges can be mitigated. This is further pointed out by Lianping (2015), who argues there is not any research on how an organization can introduce CD, and further research should be focused towards understanding these challenges and developing practices on how to effectively managing them. The challenges will also to some extent depend on the industry, as Leppänen et al. (2015) describes that the characteristic of the industry is affecting the design of CD processes. In line with this, Rissanen and Münch, (2015) argue that the B2B sector is imposed by specific challenges in comparison to the B2C segment. For example, the telecommunication industry is struggling with large variation and diversity of network configurations among clients, creating limitation on delivery and deployment speed (Lepännen et al., 2015 and Holmström et al., 2012).

In conclusion, there are some potential areas where additional research would be beneficial within the field of CD and this thesis aim to contribute in two ways. First by providing local knowledge of the challenges when adopting CD in the telecommunication industry, characterized by complex and unique conditions. Secondly, based on these challenges, reviewing how a developing company can overcome the challenges and successfully adopt CD.

1.2 Case Company

This master thesis will be performed in collaboration with a large telecom firm. It is one of the major actors in the industry, with organizational units and customers all over the world. It offers services, software and infrastructure mainly for mobility, broadband and the cloud. The part of the company that is studied in this case produce and sell both hardware and software, which in different combinations and setups create large networks that provides societies with data traffic for information and communication. The first tier customers are operators who own these networks and commercialize on the traffic flowing through them, and end users are all consumers of mobile and data traffic. Being highly integrated, the case company is involved in all of the steps from development to implementation and service of the products, sometimes even operations. It is also highly distributed geographically, with a large installed customer base and organizational units and customers globally.

The software running these networks is constantly developed in order to evolve the performance. The progress requires upgrading of this software, which involves large and complex projects. As technology and customer demands evolve rapidly, upgrading of the software running the installed hardware becomes critical. In an effort to align the company to these circumstances, continuous deployment has been stated as a company-wide goal. This initiative has been initiated in a decentralized manner, leaving different internal organizations in charge of its implementation. In some parts of the organization the adoption of CD has started and is showing promising results. However, in the specific organization included in this research they have not yet started working towards deploying software continuously. Before

embarking upon it, thorough analysis of the prerequisites and the potential internal and external consequences of such transition is wanted.

1.3 Purpose

The purpose of this study was to investigate how a telecommunication company can overcome their specific challenges when adopting continuous deployment for software. In order to fulfil the purpose, the following research questions has been addressed:

RQ1: What are the challenges when adopting CD in the telecommunication industry?

RQ2: How can a *telecommunication company overcome these challenges when adopting CD?*

To answer RQ1 a literature review was conducted to identify challenges related to CD. These have been used as codes in a template analysis to find the specific challenges within the telecommunication industry. To address RQ2, an internal benchmarking with two internal organizations, who have already begun adopting CD, has been made. In addition, a theoretical framework was created based on literature within software development and change management. The approach allowed for a formulation of how to mitigate both technical and organizational barriers, which will help the case company's transition to CD.

1.4 Delimitations

There is often a distinction between continuous integration, continuous delivery and continuous deployment. The following is based on definitions of the three concepts as expressed by Rissanen and Münch (2015). Continuous integration comprises frequent integration of software that is produced through automatic build and test processes. In continuous delivery, acceptance testing and deployment to a staging environment is automated as well. Basically it is a state in which frequent deployment is possible, but not implemented. Applying continuous deployment extends it to automatically putting the changes into production.

In this study, only the concepts of continuous integration and continuous deployment are included. In the research setting, influenced by the case company, continuous delivery and continuous deployment are seen as interchangeable. The CD implementations reviewed has according to the theoretical definition been closer to continuous delivery, but in practice it is regarded as continuous deployment. This as far as the CD vision stretches, since full automation into production is not in question in this business, at least not in a foreseeable future. As Leppänen et al. (2015) found, it is common that companies do not make a full transition, because they do not want to abandon the level of control and human decision making needed for it. This makes our concept of continuous integration basically an internal product development method and continuous deployment the case in which the customer interface is altered, and software is introduced to them in a different manner. This adaptation is more aligned with the view of the case company.

The focus of this report is on a relatively high level, and contains a limited scope and depth of the investigation. Therefore, the technical details regarding challenges and solutions found are not fully reviewed. This means that the scope of the report is not to provide a complete and tailored solution for transitioning to CD, but rather bounded to highlighting potential challenges and solutions necessary to be aware of and to review more thoroughly before moving forward.

1.5 Disposition of Thesis

This section describes an overview of the included chapters of the report.

1. Introduction
The aim of this chapter is to describe the background and research context, as well as a formulation of the research problem. It further defines the purpose and research questions of the study.

2. Methodology
The methodology includes a detailed description of how the research was conducted. In the chapter, a discussion regarding research strategy, research design and quality of research is presented.

3. Theoretical review
The theoretical review presents the concept of continuous deployment and the foundational requirements for this process. In addition to continuous deployment, it entails descriptions of agile development, continuous integration, lean software development, and change management, which are seen as essential elements for an implementation of CD processes. Thereafter, the related challenges found in academic reports are compiled.

4. Empirical Findings
In the empirical findings the results extracted from the data collection are presented. This involves three organizations within the case company and their software development practices and perceived barriers regarding adoption of CD.

5. Analysis
The analysis is structured according to the two research questions stated in the introduction. In the first section the empirical data is analyzed according to previously found challenges to adopting CD. The second section focus on how the change towards CD can be implemented using the theoretical framework and by applying empirical data from the two organizations who have already set up their CD processes.

6. Discussion
In this section, the results are further concretized and discussed from the case company's current situation. Answers to the research questions are clarified, and possible future research areas are discussed.

7. Conclusion

In the concluding remarks the main findings of the study and its contribution to academia is highlighted.

2 Methodology

The methodology includes a detailed description of how the research was conducted. In the chapter, a description of research strategy, research design and quality of research is presented.

2.1 Research strategy

Since the purpose of this study was to investigate how an organization can overcome challenges when moving towards CD, it was crucial to study how the case company worked in practice. Software development will vary across organizations and industries, and assuming one absolute truth of best practice would be naive. Therefore, it seems logical to base our philosophical assumption on constructionism. A constructionist research approach implies the assumption that several truths exist and researchers should provide evidence on how different claims for the reality and truth is composed in everyday life (Easterby-Smith et al., 2012). However, the technical barriers investigated are objective and not open for much interpretation. This is something that must be acknowledged, as this is not in line with the philosophical assumption.

A constructionist research design is usually of qualitative nature, which is the chosen strategy in this thesis. The qualitative approach has allowed a deep understanding of the research subject, which is important since CD is relatively unexplored in previous research. We could capture a versatile and thick description of how the case company works with software development in practice, as well as an extensive investigation of the research context. The context, being the telecommunication industry, characterized by unique challenges that implies more prominent constraints in comparison to most other IT based industries, which CD usually is associated with. To mention two examples, the industry is dependent on the use of installed base hardware implying large and complex software updates, and is influenced by more comprehensive legal restrictions that complicate more frequent deployments. Using a qualitative strategy will therefore enable the rich description of the context needed to achieve a complete understanding of CD in the telecommunication industry. (Bryman and Bell, 2011)

A qualitative research strategy will often imply an inductive approach, where theory is generated from the research. However, it is probable that the inductive process will include aspects of deduction. Usually this is realized through an iterative process, where the researcher is going back and forth between data and theory. In this research, theory has been used as a foundation to guide the data collection in the qualitative investigation. In addition, the iterative process between data and theory has been adopted and more of an abductive approach was implemented during the research. (Bryman and Bell, 2011).

2.2 Research design

Research design is about organizing research activities, including collection and analysis of data that is most likely to fulfil the purpose of the research. The assumption that no absolute truth exists, fits a wide range of methodologies within

the constructionist paradigm. The choice of research design has been a case study, which is considered to originate from a constructionist epistemology when a single case is investigated (Easterby-Smith et al., 2012). The chosen design enables a detailed and intense analysis of the case (Bryman and Bell, 2011), which focuses on the complexity and the specific characteristics of the case in question (Stake, 1995). As mentioned before, this fits the research purpose since there is limited work performed within the field of CD. Adopting a case study also allowed a deep investigation of the software development in its practical setting. Moreover, it offered the possibility to explore how the industry imposes restrictions for CD. The remainder of the research design section will contain how data was collected and analyzed within the case study.

2.2.1 Literature review

The first step in the research process was to review previous literature conducted within the topic of CD. It provided an overview of the research subject and revealed a gap in previous work. As discussed in the introduction, we found that several scholars have identified which challenges a developing company may face when adopting CD. However, research on how these barriers can be mitigated was lacking. These insights led to the creation of our research questions:

RQ1: What are the industry specific challenges when adopting CD in telecommunication?

RQ2: How can a *telecommunication company overcome these challenges when adopting CD?*

When the purpose and main objective with the study was set, the literature search could be redirected towards building a framework to successfully answer the defined RQs. Relevant concepts within software development such as continuous integration, continuous deployment and agile and lean software development was compiled into the literature framework. The framework was complemented with change management, which was considered to be a key aspect of succeeding when adopting CD, mostly for how to overcome barriers anchored in organizational aspects. How these concepts are defined and connect to each other is discussed in detail in the theoretical chapters (3.1). The final purpose with the theoretical review was to compile the challenges previous scholars had identified for transferring to CD. They were later coded into categories, which provided the input to the analysis for fulfilment of RQ1. The literature search was conducted in a four stage approach as summarized below.

1. **Initial search**. The purpose with the initial search was to build an understanding of CD and previous research performed on the topic. It could further provide insights to synonyms and different terminologies used for the concept.
2. **Define terminologies**. Based on the articles found in step 1, the next step was to define the different terms that could be used as keywords in the following steps.

1. Continuous deployment
2. Continuous delivery
3. Agile software development
4. Lean software development

3. **Search in Multidisciplinary journal databases**. With the keywords defined in the previous step, the general search was conducted in Google Scholar and Chalmers library to identify the main articles within CD. The 50 first results on each keyword were reviewed.
4. **Search in subject specific databases.** In the final search an additional search was performed within databases within computer science. The purpose was to narrow the search and find articles not identified in the previous search. The included databases were:
    1. IEEE Xplore: digital library
    2. ACM digital library
    3. Inspec

2.2.2 Data collection

The data collection has been divided in two blocks. The first block involves collecting data from the main organization, not yet adapted to CD. In the second block, two other internal organizations were included. Both of these have started to implement CD in their product introduction processes. This setup provided learning, regarding barriers and how these can be managed, from more experienced parts of the company. The choice of internal benchmarking was made due to several advantages. Most prominent was the availability of information and transferability of practices. Moreover, different cultures that could potentially inhibit the transformation of operations could be avoided (Southard and Parante, 2007). The appropriate organization to include in the study was determined by the supervisor at the company, in order to ensure that it was possible to transfer the insights between the chosen internal organizations. The overall structure of the data collection is summarized in Table 1 and the detailed can be found in the appendix.

	Involved departments	**Number of people interviewed**	**Purpose**
Block 1 (Organization A)	R&D Product development Product management	9	Map current process Identify perceived challenges before adopting CD
Block 2 (Organization B/C)	R&D Product development Product management	6	Identify barriers and how these were solved Apply findings to organization A

Table 1, summary of the data collection

The constructionist assumption that several realities exists, means that it is important for the researcher to capture multiple perspectives from various individuals (Easterby-Smith et al, 2012). To fulfil this criterion, interviews were conducted with

employees from R&D, product development and product management to achieve a complete understanding of how CD will affect the whole organisation. For example, an employee from R&D will in many aspects not experience similar challenges as a product manager, which highlights the importance of the chosen research design. To complement the interviews, we have performed an archive analysis by reviewing internal documentation. These two collection methods will be discussed in the following sections

2.2.2.1 Interviews

The main data collection method has been in-depth interviews. Mainly since it allows for a deep understanding of the respondents' views on the product introduction process. By interviewing employees from different departments, the data collection could provide the versatile and comprehensive picture of the deployment process needed for qualitative research. Moreover, Easterby-Smith et al (2012) argues that in-depth interviews are appropriate when the researchers want to influence the respondent's world by creating a deep understanding of it. This captures an important aspect of the research, namely to map the case company´s current software development process. It is not until a complete understanding of the original state is gained that it can be suggested how they can overcome barriers when adopting CD.

The interviews have been semi-structured and can be defined as open-guided interviews. When interviewing several respondents, it provides comparability, since the same topics are covered. Thus, it enabled a comparison of different perceptions of the research subject within and across different departments as well as the organizations.

2.2.2.2 Secondary sources

To complement the interviews, internal documentation considered relevant for the study has been reviewed, increasing the understanding of how the current development processes are designed. Moreover, we could triangulate the interviews in block 2 with historical data, since the other organizations started their CD transformations a few years ago. As described by Bryman and Bell (2011) it provides the possibility to create a timeline and we could easier map the organizational change that has taken place over the last years.

2.2.3 Data analysis

Since two different research questions were stated, the data analysis was customized to both questions. This section will describe how this was made.

Two common approaches to analyze natural language are content analysis and grounded analysis. Using content analysis, the researcher is analyzing the data for ideas and constructs that have been defined before. Grounded analysis is more inductive in nature and allows the researchers to use intuition guiding them through the development of their understanding of the data. The codes consequently emerge over time. In order to analyze RQ1, a template analysis have been used, which is located in the interface between these two methods. In practice, this method creates the opportunity to define codes in advance. In this case, these are the categories of

challenges identified in the theoretical review. During the analysis of data, these codes could later be adapted toward the specific challenges that the case company is facing and RQ1 could be fulfilled (Easterby-Smith et al., 2012). In addition, data from interviews in block B was complementary in the analysis, which made it possible include challenges not realized by personnel ex-ante.

In terms of RQ2, another method for analyzing data was required. After defining the challenges in the previous stage, this question was concerned with how an organization can adopt CD and overcome these challenges. Therefore, an internal benchmarking was performed to investigate how the change and CD process could be designed. The empirical data from block B could consequently be analyzed and adapted to the main organization in regards to RQ2. In this stage, company specific challenges and technical aspects not provided in previous research were valuable. Data was also analyzed in relation to the theoretical framework, where change management was applied to the setting, valuable for managing the organizational challenges.

Research Questions	**1**	**2**
Method for data analysis	Template analysis	Internal benchmarking + Theoretical framework
Initial Codes	Challenges identified in previous literature	Industry specific challenges
End codes	Industry specific challenges	Formulation of mitigating actions when adopting CD

Table 2, methods for data analysis in relation to the research questions

2.3 Quality of research

In the following section quality measures of the study will be discussed, including validity, reliability and generalizability.

2.3.1 Validity

Three types of triangulation methods identified by Denzin (1978) and Patton (1999) have been used to increase the validity of the research results. The most frequent has been triangulation between sources, which was done by comparing data from respondents included in the study. It has been implemented with two purposes; first to check for consistency between respondents within the same organizations, and secondly to identify consistencies and inconsistencies across different organizations. This consequently captures a versatile picture of CD and what is required by each organization. The second triangulation method adopted, was checking the consistency of data through different data collection methods. Interviewing, which has been the main method, has been complemented with internal documentation regarding CD. This was especially valuable in triangulating interviews in block 2 with historical data, since the benchmarked organizations started their CD transformation several years ago. The last triangulation was in terms of reviewing the findings using multiple analysis. Every interview has been recorded and later transcribed, which allowed both of the researchers to review the collected material. According to what is

pointed out by Patton (1999), the goal has not been to find a consensus, but rather to introduce different viewpoints in order to reach a deeper understanding of the research subject as well as finding potential blind spots.

2.3.2 Reliability

Reliability has its origin in quantitative research and concerns to which extent the study can be repeated (Carcacy, 2007 and Leung, 2015). However, due the difficulty in replicating the exact conditions under which the data was collected in qualitative research, reliability therefore need to be conceptualized somewhat different (Carcacy, 2007). Mason (2002) argues that reliability should be concerned with demonstrating that the researcher has not been careless in data recording or analysis, and that they have not invented or misrepresented data. To increase the reliability in the study, we have presented our research processes in a transparent way, in order to simplify for an external observer to evaluate our interpretation of the social phenomena investigated. Moreover, we have continuously reflected on our interpretations, strived to be consistent in data collection, systematically analyzed data and supported our interpretation with evidence (Lewis and Ritchie 2003). Another concern regarding reliability is to verify the accuracy in data collected from original sources. In order to verify data from interviews, the previously discussed triangulation methods were used.

2.3.3 Generalization

Generalization is about the degree to which the findings are relevant in another setting (Yin, 2003). A concept within generalization is transferability, which is the possibility of making such an application. Transferability is determined by the conformity between the sending context and the receiving context (Koch, 2006). The transferability describes how well the findings of this research could be applied in another context, and in our case the transferability is considered to be low. The telecommunication industry has unique and specific challenges making the CD concept different compared to other industries. Therefore, the objective with the research is to create local knowledge within the specific context and the result should be generalized into other settings with great caution.

3 Theoretical Review

The theoretical review includes two main sections. In the first section, continuous deployment and its related concepts are introduced and defined. The second section will discuss the major challenges adopting CD that have been reported in previous research.

3.1 Theoretical framework

Adopting CD is not something that can be done in solitary, it requires serious company effort and an established set of agile processes. In their conceptual model "stairway to heaven", Holmström et al. (2012) describe the evolutionary path of a company's software development practices towards CD. Similarly, Claps et al. (2015) have created a conceptual model explaining the CD process. In Figure 1 and Figure 2, we can see that scholars are describing that CD processes originate from agile methodology. In addition, it illustrates that continuous integration (CI) is a precondition for the next phase, where functional software is deployed earlier and more frequent. The importance of an established CI server is also advocated by practitioners (Fowler, 2013 and Arnold, 2012).

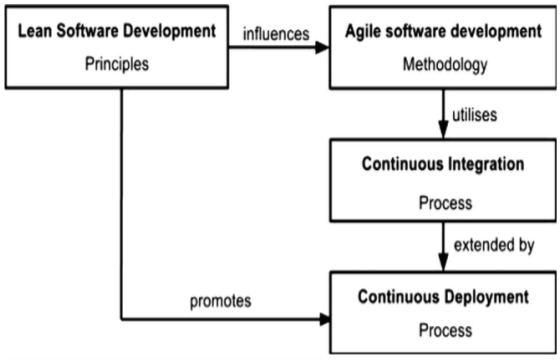


Figure 1, model explaining the CD process (Claps et al., 2015)

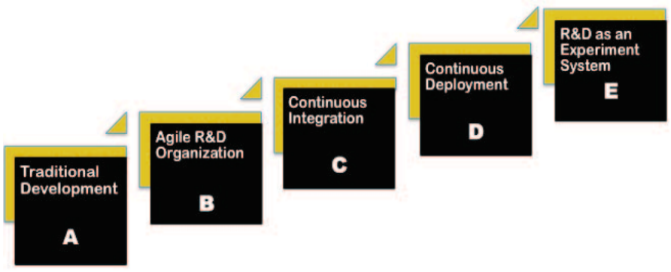


Figure 2, "Stairaway to heaven" (Holmström et al., 2012)

Claps et al. (2015) further argues that Lean software development (LSD) has both a direct impact on the creation of Agile, and promotes CD. Lean principles in combination with a CI server has shown to be a formula for success when implementing CD. A lean mind-set can reduce waste and shorten the feedback loop

from customers, two important goals of CD (Claps et al., 2015). In addition, when moving towards both CI and CD, the company will not only face engineering challenges but also non-technical barriers that are managerial and organizational. As limited research has addressed how these social challenges can be mitigated in the context of CD, we believe that change management can provide important insight. In conclusion, to fulfil the purpose of the thesis, we believe it is not only important to define and understand CD, but also agile, CI and LSD. These concepts are largely dependent on each other and one cannot exist without the other. As mentioned, we aim to introduce a new theoretical view in the context of CD, namely change management. As the developing company will face major internal resistance, we believe change management is crucial to overcome such challenges. The relation between these concept included in the framework is summarized in Figure 3 and the remainder of this section will briefly discuss these different concepts.

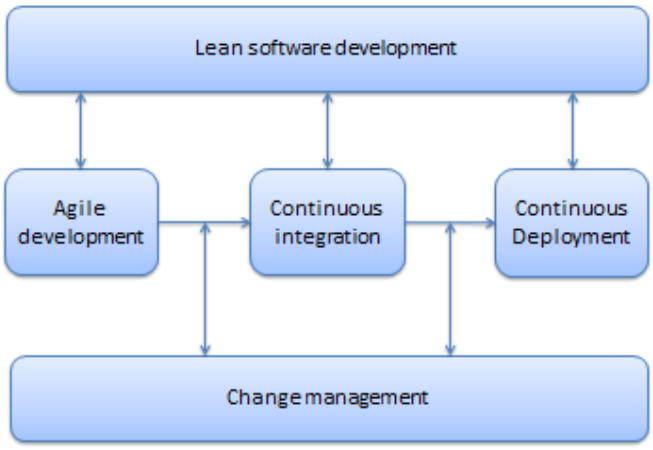


Figure 3, Theoretical concepts and their relation

3.1.1 Agile Development

The methodology behind agile software development was founded at a ski resort in Utah, where 17 people went to talk, relax and try to find a common ground. As a result, the agile manifesto was born with the purpose of “uncovering better ways of developing software by doing it and help others do it” (Fowler and Highsmith, 2001). In broad terms, agile development focuses on close collaboration and sharing of information between individuals to quickly respond to changes. The use of working software provides fast feedback and the possibility to measure actual results. Frequent interaction will also minimize the need for comprehensive documentation. Agile empathizes close customer collaborations and in agile teams the sponsor, customer, user and developer all participate, and different experiences can be merged to produce more valuable and cost-efficient result (Highsmith and Cockburn, 2001). With the advancement of technology, blindly following a plan can have

negative consequences, hence agile team's focuses on continuously responding to external changes (Fowler and Highsmith, 2001).

In addition, the manifesto includes 12 principles to guide teams implementing agile software development (Fowler and Highsmith, 2001). Williams (2012) has through a survey found that the two principles focusing on delivering working software early to customers is perceived as most important for practitioners. Delivering solutions early and often, will allow the company to create maximum business value for the customers. The result of William (2012) is similar to the findings of De Cesare et al. (2010), which indicate that satisfying customers through early and continuously delivered software is one of the most important principles.

In summary, agile practitioners highlight continuous deployment of functionality as perhaps the most important aspect of today's competitive landscape in software development. This insight points out the importance of CD and further shows the need to have an established agile mind-set when moving towards this state.

3.1.2 Continuous Integration

Continuous integration is an extension of agile processes and a prerequisite for CD (Berntsson Svensson et al., 2014; Holmström et al., 2012 and Claps et al., 2015). The concept means achieving automated building and testing, as well as frequent integration of the software (Rissanen and Münch, 2015; Leppänen et al., 2015 and Holmström et al 2012). When code for a build is written, for example a feature or a bug fix, it should immediately be tested locally. If this test is successful, it is built and pushed to a CI server. This server merges the code with the latest version on the server. If also the merge is successful, a new version of the software is built. This pipeline is made automatic for CI, taking the process one step closer to CD. Changes are quickly shared among team members and validated, and Virmani (2015) advocates sharing and integrating within the entire product level. Hence, CI can both increase the productivity and communication in the development process (Bosch and Ståhl, 2013).

Academic reviews of continuous integration claim that it can speed up release and feedback cycles (Berntsson Svensson et al., 2014 and Bosch and Ståhl, 2013). According to Rissanen and Münch (2015), slow release cycles are perceived as one of the main issues of product management. Stated by Miller (2008), the reduced working time achieved by continuous integration comes without negative effects on quality. Some research even claims that quality is one of the benefits of CI. Fowler (2006) argues that the risk of errors due to integration decreases, since this is no longer done as one large event at the end of the development process. He further describes that the increased frequency of builds and tests, and the smaller size of each build, results in that bugs are found and solved both easier and earlier. The decreased size of builds and the more frequent testing and integration, is what enables organizations to evolve towards a state of CD.

3.1.3 Continuous Deployment

The main difference between traditional software deployment and CD is the frequency of deployment. This is illustrated in Table 3 where Claps et al. (2015) have

summarized research and published material from practitioners, when comparing the two different practices.

	Traditional deployment	**Continuous deployment**
Frequency of deployment	Every six month	Daily. IMVU report deploying 50 times a day. Facebook deploys new code every day
Relative risk of deploying	Higher. Since deployment is an infrequent process, and because the software being deployed contains many changes, there is a higher chance that the deployed software contains bugs	Since changes are frequent but small, they will cause smaller problems. This is because small, incremental changes reduce the risk (in size) posed by each single change
Customer/developer feedback loop	Long - dependent on frequency of deployment	Very short, customers constantly receive updates
Undeveloped features	69% of the required features in software products are developed, leaving approximately 31% required software features underdeveloped	CD helps in decreasing development of unnecessary features due to shorter feedback loop. Features are developed constantly so that meaningful feedback can be gained from customers on what features they would like developed. This in turns helps to prevent the development of any wasted software.

Table 3, a comparison between traditional software deployment and the CD process (Claps et al., 2015)

CD is an extension of CI, where the build, test and release process are highly automated and streamlined, making every system change realisable through a simple button push (Humble and Farley, 2010 and Fowler, 2013). It consequently provides the developer with the authority to decide when to deploy the software (Leppänen et al., 2015). Common perceived benefits with CD are shorter time to market and an immediate feedback loop from customers to developers, as well as earlier return on investments with more frequent releases and substantially less overheads (Agile Alliances, 2012; Arnold, 2012 and Leppänen et al., 2015). However, establishing CD processes requires extensive instrumentation and infrastructure, in order to mitigate the risk of deploying new functionality resulting in failure (Agile Alliance, 2012). To ensure that the software will work in production, a deployment pipeline with a highly automated delivery process is necessary to test the software in increasingly production-like environments (Fowler, 2013). In addition, it necessitates a closer collaboration between development and operations also referred to as Devops (Fowler, 2013 and Smeds et al., 2015).

In a systematic analysis of software practices, Ashfaque Ur Rahman et al. (2015) identified several commonly used practices at 19 companies who had adopted CD. These findings are consequently valuable for the study and are described in the Table 4.

Automated testing and deployment	Practitioners who adopted CD is dependent on an automated test and deployment process. This allows them to rapidly deliver software to customers, as well as to early detect bugs.
Unit test	Unit tests is the most frequently used test among the CD companies and refers to the testing of individual software components. The characteristic of the test is the early detection of faults and that it is written before the code.

Code review	A practice when new code is delivered for inspection and approval, which can be both manual and automated. The benefits with code review is sharing of knowledge, detection of bugs and revealing alternative solutions.
Dark launching and Feature flag	Dark launching refers to deployment of software changes for which functional parts can be hidden from the customers. The benefit is to receive early feedback from live application, without disturbing the user. A similar practice is feature flags, enabling a feature to be turned off when the new code is malfunctioning.
Intercommunication	CD requires an increased intercommunication across teams (Claps et al., 2015) and help the adopters to achieve efficiency. The study reveals that the adopters are rather using automated tools to share information regarding development and deployment amongst team members. This is a major transformation from the traditional setup where teams relied on team and scrum meetings
Shepherding Changes	Using shepherding changes, the developers are responsible for the software changes through the whole deployment process, as well as correction of problems in production.
Monitoring	Monitoring refers to the collection of deployment data, creating performance metrics in a suitable format. This is important for quick identification of the root cause to the faults in development and deployment. It can also provide fast feedback on how the features are used in production.
Staging	Staging is a set of techniques executed after the code are written, tested and before it is deployed to the end-user. The relevant technique for the telecommunication industry is gradual rollout, were the new software is deployed to a small part of the users before it is available to all.

Table 4, Common practices adopted in continuous deployment, (Ashfaque Ur Rahman et al., 2015)

3.1.4 Lean Software Development

According to the Poppendieck and Poppendieck (2003), lean software development (LSD) “expands the theoretical foundations of agile software development by applying well-known and accepted lean principles to software development”. Connecting LSD to CD, Claps et al. (2015) argues that the lean principles - to eliminate waste, amplify learning, deliver as fast as possible and to empower the team - seems to be strongly in line with and promotes CD. Moreover, they found that a lean mind-set is critical to success when implementing CD. Similar, Ries (2011) argues that CD should be used in combination with lean manufacturing principles. When breaking down features in small batch production, the developing company can increase the speed of development and deploy faster. Achieving such processes is however challenging and there is no fixed method for how it should be applied in a CD setting. Especially not in terms of how larger features can be broken down to smaller pieces. One software company has reported to use dark features, where large features are developed in smaller batches not available for the customer to interact with before deployment. In that sense, the customer does not know a feature is developed and cannot interact with it.

Eliminating waste is a basis for lean, and in the context of software development it is crucial to accelerate the customer feedback loop. This can be realized through a data analytics platform, monitoring customer behaviour. With a more focused and customer oriented development, waste and inventory can be reduced, while experimenting with new features in specific customer segments. The strategy consequently quantifies data and makes sure that software deployed to mass-markets will be used by all customers. In addition, setting up processes that allow continuous feedback loops from customer will amplify learnings. In summary, deploying in smaller batches and acting on shorter customer feedback are

challenges based in lean principles. Adopting a lean mind-set can consequently ease the transition to CD for a developing company. (Claps et al., 2015)

3.1.5 Change management

Empirical reports on organizational change reveal that most of them fail to deliver the expected results and 70% of the change initiatives fail completely (Blanchard, 2010). Extensive research has been written on the subject, and academics have identified crucial stages in this process and the aspects that are the most important for successful change management. The findings of Blanchard (2010) highlight the importance of understanding change management to mitigate the non-technical barriers when moving toward CD and through a literature review on the subject we found six major steps that should be considered. They are summarized in the following bullet list and discussed in the remainder of the section.

- Create a vision and communicate
- Select leaders and involve
- Measure and share progress
- Adapt structures and systems
- Gradual change

Early in the process, it is important to define the change initiative. It is necessary to create a vision, and clarify where the firm is and where it is desired to be at the end of the change (Kotter, 2007; Blanchard 2010; Mento et al., 2002). According to Blanchard (2010), this should be accompanied by an explanation of why this is desired, by motivating the need for change, the benefits expected from it and the consequences it has for all stakeholders. He states that answering these questions is needed in order for involved parties to commit to the initiative from the start. It is necessary to address this initially and provide an understanding of what it will require from everyone involved and what their roles will be. This is highly relevant in restructuring for continuous deployment, as it is likely to imply changing responsibilities among team members. Blanchard (2010) also emphasizes inclusion of the customer perspective, as they too can be heavily affected by a change. Changing the deployment process is a good example of such a move. Hence, clearly expressing the goal, benefits and impact expected can elicit commitment both internally and from customers for a transition towards CD. Going through these early steps can also be helpful in establishing a plan for the implementation.

Management and committed employees can be used for further leverage, persuading others and reducing resistance. A coalition of people leading and promoting the change is needed from the start; both people with power and resources to drive the change, as well as involvement and empowerment of others to act on the vision (Kotter, 2007 and Blanchard 2010). Pilot projects can be helpful in identifying and addressing challenges, as well as to provide advocates of the change and leaders for the continued work. (Blanchard 2010) The vision and the plans made, has to be communicated well and often to make sure that everybody is aware of their roles and the overall goals. It is recommended to use all channels that can support the spread of it and that can help to inform people, and lead by example to teach the new behaviours (Kotter, 2007).

Being honest is emphasized by Blanchard (2010), to induce trust for the initiative and increase commitment. To maintain the commitment, it is important to make sure that quick results are obtained and communicated. Monitoring the progress and measuring the results derived from it can be helpful in this, as well as to keep control over the development (Mento et al, 2002). For continuous deployment, Leppänen et al. (2015) suggest measurement of the time for a code change to reach production, cycle time for deployable builds, and the degree of automation in the development and deployment.

Change necessitates a separation from the past. As behaviours and processes are supposed to change, so need the structures and systems that does not support the new way of working (Kotter, 2007; Mento et al, 2002 and Blanchard, 2010). This means assessing and adjusting staffing, training, communication, appraisal and reward systems as well as defining the new roles and organizational structure to accomplish a fit with the targeted state (Mento et al, 2010). Pointed out by Leppänen et al. (2015), one of the requirements for a successful transition to continuous deployment is developers with sufficient knowledge of continuous deployment practices. Finally, the change has to be institutionalized to avoid the risk of falling back into the old ways of working (Kotter, 2007; Blanchard, 2010 and Mento et al, 2002).

Some of the mentioned authors suggest that successful change should be followed up by more change. As Kotter (2007) explains, change is an ongoing process and not a single event. Jick highlights that it is important to change what is needed but to hold on to what works well (Mento et al, 2002). This is brought up by Reger et al. (1994) as well, emphasizing the importance of building the new organization on the organizational identity. By doing so, it is possible to avoid too big changes and a lot of resistance. When there is no crisis and no need for drastic change, a change that is small enough to be perceived as achievable, but large enough to overcome organizational inertia, is optimal according to Reger et al. (1994). This relates to what Mento et al. (2002) refer to as creative tension, a situation in which the present state of the organization is perceived as different from the ideal, claimed to foster intrinsic motivation for a change. In the case of a change towards CD, possible steps could be to sequentially implement agile, CI and CD, and even within these separate transformations successively raise the levels pursued.

3.2 Challenges to Continuous Deployment

The previous section defined the main concepts related to CD in order to build an overall understanding of the research subject. The remainder of the theoretical chapter is concerned with the challenges toward an adoption of CD. In a literature review, 55 challenges (appendix) related to CD were identified. These challenges have been defined as either technical or organizational and later divided into seven categories (Table 1). It has been used as a template to find the specific challenges within the case company, an approach suggested by several scholars (Rissanen and Münch 2015 and Claps et al., 2015). In the following section each category will be discussed.

Technical challenges	Organizational challenges
Network configurations Continuous integration Automated testing and quality Deployment	Company commitment Changing responsibilities Customer resistance

Table 1, summary of the challenges in the defined categories

Title	Authors	Context
“On the journey to continuous deployment: Technical and social challenges along the way”	Claps et al (2015)	Software
“Climbing the" Stairway to Heaven" A multiple-case study exploring barriers in the transition from agile development towards continuous deployment of software”	Holmström et al. (2012)	Multiple
“The Highways and Country Roads to Continuous Deployment”	Lepännen et al. (2015)	Multiple
Transitioning Towards Continuous Delivery in the B2B Domain: A Case Study	Rissanen and Münch (2015)	B2B software applications
Continuous Delivery Huge Benefits, but Challenges Too	Lianping (2015)	Software applications

Table 5, the papers included in the theoretical review

3.2.1 Network configurations

One of the major characteristic defining the B2B segment is the diverse product environment that is imposing complexity in the handling of different customer configurations (Rissanen and Münch, 2015). Holmström et al., (2012) found that the biggest challenge for a company within the telecommunication industry adopting CD was related to the complexity created by different network configurations at customer sites. Leppänen et al., (2015), confirm that the industry is struggling with its diversity of clients and the variation of network configurations. Furthermore, they state that the telecommunication industry experiences restrictions both in terms of delivery and deployment speed. One of the common problems related to this challenge, is when the customer has an old version of the product, but wants a new feature that consequently needs to be manually configured. In relation to this, the customers experience updates as risky and stressful. Especially if the product is complex, it will be hard for the developing company to guarantee minimal network impact. Hence, customers experience upgrades and deployment of new features, as a major risk of interfering with legacy (Holmström et al., 2012). In addition, different environments in development, testing and production present the challenges of keeping these environments similar enough, while fulfilling configuration conventions (Leppänen et al., 2015).

The situation is imposed by further complexity and requires substantial efforts to integrate and configure the software with third party applications and to be functional with multiple external components (Rissanen and Münch, 2015). The speed of deployment will also pose a threat if potential partners cannot update their plugins in a way that will keep up with the company's continuous updates (Claps et al, 2015).

3.2.2 Continuous integration

The technical challenges found in the literature study highlight the need for continuous integration and the requirements that comes with it. Important aspects emphasized by Claps et al (2015), refer to assuring that the software is constantly made ready for deployment, including conflicts to merge software and determining its deployability. The authors further point out the demand for resources, both hardware and software, which is necessary in order for the new software to be integrated often enough. Berntsson Svensson et al. (2014) agree by stating that providing the tools and infrastructure needed for an efficient CI process is a challenge. As a consequence, updates can be slowed down and stuck in integration queues.

In order to achieve source code control, Claps et al. (2015) also argue that one should aim to isolate one single branch of code. According to Berntsson Svensson et al. (2014), this can be harder when dealing with more complex products. Also, code dependencies create challenges in breaking down the requirements and coordinating the different parts of the integration process. Rissanen and Münch (2015) add on the technical challenges by mentioning that software often is integrated into third party applications and accompanied by several external components, which complicates the process. The report describes in general terms that for continuous integration to enable more rapid delivery, the branches of code created has to be shortened.

3.2.3 Testing and quality of the software

Producing and releasing software in a higher pace, of course raises the question of how it impacts the quality of the products. Testing of the products therefore becomes central for a successful transition towards continuous deployment. Claps et al. (2015) states that there is also a risk that minor changes in the code can generate major changes to the database schemas. As Rissanen and Münch (2015) found in their research, the task developers thought of as the largest challenge is automated testing and test environments, both the creation and maintenance thereof. It is perceived as problematic especially because it has to be built on top of a mature product. Leppänen et al. (2015), explain that it is important that these testing environments are similar to those of development and testing, which can be challenging to accomplish. Achieving fast enough feedback loops from tests has also been identified as a challenge. If it takes too long time to get feedback from the testing, the information can be out of date (Berntsson Svensson et al., 2014).

Rissanen and Münch (2015) lift the issue for managers to determine what to test during automatic acceptance testing in order to validate the software. This concern is addressed by Leppänen et al. (2015) and Berntsson Svensson et al. (2014) as well, stating that the perception in companies is often that manual tests are needed as complements to the automatic testing. Automatic tests are by many practitioners seen to provide insufficient coverage, and some manual testing of performance and security, as well as exploratory on actual devices, are common (Leppänen et al., 2015). Abandoning all manual elements, as required for CD, is thereby a major challenge. Leppänen et al. (2015) also identified problems related to legacy code, systems that have been built over a long time and which may not be properly

designed for automated testing. Creating proper automated tests for the legacy systems can generate an overload of work for the developers. Instead, the responsibility of testing is often shifted to other developers or quality assurance personnel, which may delay the detection of bugs.

The size of the code base and the number of tests needed for validation, is affecting not only the time necessary for developing the tests but the time for execution as well. In addition, acceptance testing is usually performed by suppliers and customers, requiring lots of time and resources from both parties. This is challenging to overcome, especially when the customer environments, configurations and components are diverse (Holmström et al., 2012; Leppänen et al, 2015 and Rissanen and Münch 2015).

3.2.4 Deployment

If the deployment of software causes downtime the customer will not be able to perform their job, since tasks are interrupted and data can be lost. Hence, the importance of bug-free deployment cannot be stressed enough. To handle this problem, companies can negotiate with the customer when the deployment can take place, for example when the system is closed for a short period (Rissanen and Münch, 2015). On the other hand, CD requires seamless upgrades that will deploy the features without hindering the customer of using the software during the process. However, such a process is associated with major upfront set up time, since it is extremely complicated to implement and will be very demanding for the hardware.

Another related challenge is to develop the ability to deploy in small batches, breaking down large features and gaining faster customer feedback (Claps et al., 2015 and Rissanen and Münch 2015). Rissanen and Münch (2015) argue that the ability to deliver short-lived features branches will be especially challenging for companies that have been active for a long period and might therefore be more inclined to develop long-lived feature branches. Furthermore, it could be challenging to prevent that the increased frequency of deployment leads to more software bugs. Another risk that needs to be addressed is the deployment of a batch of code that is not fully complete. This may lead customers to interact with it in an unwanted manner (Claps et al., 2015). In Table 6 the previous discussed technical categories and their related challenges are summarized.

Technical categories	**Challenges**
Network configurations	Diverse product environments and variation of network configurations Legacy code Functional with third party applications and external components
Continuous integration	Ever-ready software HW & SW resources One single code branch Code dependencies Related applications and components Small feature branches
Testing and quality	Quality and code review

	Changes to database schemas Automated testing, built on mature products Environments & manual elements in testing Regression feedback time Legacy code Code base size and number of tests Acceptance testing and user diversity
Deployment	Customer downtime Seamless upgrades Small batches Product quality

Table 6, summary of technical challenges when adopting CD

3.2.5 Company Commitment

When adopting CD, a software company reported that the biggest challenges have been organizational. This was mainly since the activities required for deployment included several divisions of the company. These divisions had different work processes, their own interests and perceived territories of control, which created conflicting goals and tension (Lianping, 2015). Similarly, Claps et al., (2015) found that a major challenge was the need for a company wide effort which further required increased collaboration between teams. A comparison between teams that adopted CD and teams that did not, showed that cross-team communication was considerably higher in the first case. Leppänen et al., (2015) have also found that top management support is a prerequisite for success. Moreover, it was found that resistance to change can be a barrier for the organization adopting CD. If superiors resist the change towards CD, or the culture is not receptive to new ideas, the change process towards CD will be strongly inhibited. Holmström et al., (2012) explains that the lack of transparency can be a barrier when adopting CD. To create an overview of the development projects, there is a need to receive better status reports from the involved teams. This will consequently increase the transparency and speed.

3.2.6 Changing Responsibility

Changes in responsibilities and roles across teams as a result of CD, will pose a serious challenge for the organization (Claps et al., 2015). The impact of changing responsibilities experienced by Atlassian, a software company, is summarized in Table 7 (Claps et al., 2015).

Role	Before	After
Software developer	Develop software and create test for that software	Now also responsible for deployments
Product manager	Manage the software product, including its roadmap of features	Experiment on features with customer, using customer usage information, to reduce unnecessary feature development and thus minimize waste
Technical product writer	Writing a single set of user documents for a software product	Write two sets of user documentation for a single product
Product marketer	Market a software product for a major version release of that product	Market a version-less software product that is constantly changing
Support team	Support a relatively set amount of bugs that are tied to a major software release	Manage an increased frequency of bugs due to the lack of major software release

Table 7, changing responsibilities when adopting CD (Claps et al., 2015)

In the CD process, the software developer will be responsible for the deployment and the decision making on whether the code meets the required quality. This is a challenge mentioned in several studies that will increase the pressure on the developer (Claps et al., 2015). This especially since their reputation is on the line and deploying a broken build can affect the relation with the customer in a negative way (Leppänen et al., 2015). When the developer has the authority to deploy features, it will also be harder for management to keep track of which features that are available for the customers. The increased responsibility for developers, in combination with the varying interval when updates take place, will also complicate the situation for the sales department constituting the customer interface. This, due to more complex tracking of deliveries and the lacking knowledge about which features that are finished. In addition, the developers need to possess additional competences within version control and automated testing (Rissanen and Münch, 2015).

3.2.7 Customer resistance

In the literature review, a major finding is that some customers do not want to receive frequent updates of features (Claps et al., 2015, Rissanen and Münch, 2015 and Leppänen et al., 2015). Deploying more frequently may increase the possibilities of deploying bugs; hence companies can be reluctant to purchase such software (Claps et al, 2015). The resistance can also be grounded in that changes will complicate their day-to-day business. When performing similar tasks, the changes from new features can make the tasks more time consuming, leading to frustration (Rissanen and Münch, 2015). An empirical example of this resistance is from a telecom provider who tried to increase the number of deployments from quarterly. When the customer was not prepared to make such a change they returned to the traditional deployment schedule (Leppänen et al., 2015).

Moreover, the CD process does not support the introduction of the updates to customers. The feature discovery consequently presents a challenge of showing the customer when they have developed new features. When customers do not realize the existence of new features, it will be harder for the developing company to illustrate the added value of having the ability to introduce new features more rapidly (Claps et al., 2015). Rissanen and Münch (2015) explain the same problem, where customers become less aware of changes when deployments are smaller and more frequent. They argue that listing the changed features in changelogs is crucial when releases are more frequent. When making the transition, scholars are suggesting that it is important finding a lead user in order to develop a successful CD process (Holmström et al., 2012 and Rissanen and Münch, 2015). In Table 8 the previously discussed organizational categories and their related challenges are summarized.

Organizational categories	**Challenges**
Company commitment	Cross-team communication Top management support Team experience Resistance to change
Changing responsibilities	Pressure to deploy frequently New competences

	New market and sales strategies
Customer resistance	Customers do not want new features Deploying more bugs Interrupting day-to-day operations Find lead user Feature discovery

Table 8, organizational challenges when adopting CD

4 Empirical Findings

In the empirical findings, the results extracted in the data collection are presented. This involves three organizations within the case company and their software development practices and barriers when adopted CD.

4.1 Organization A

4.1.1 Introduction

The product´s main purpose is to transport the traffic from point A to point B. A hop, as it is called, consists of two nodes; one transmitter and one receiver. Traffic entering the transmitter side is processed in equipment at the node, and the outgoing signals are then sent via microwaves from the transmitting antenna to the receiving antenna. The process is then repeated also on this side of the hop. It is situated upstream in the networks, which makes much of the end user experience directly dependent on the functionality of these nodes. If a product is malfunctioning, there is a risk that traffic to the lower level nodes are affected.

It is an old product that has been developed during 20 years. There are several products variants within OrgA, and the software base consists of large volumes of code. The installed base stretches to six figure numbers and the composition of equipment and configurations at each site can differ widely. Upgrading is therefore done in major projects, to keep control over the situation and make sure that the impact is kept at a tolerable level. As explained, the department is now investigating ways of improving the software upgrade process and is therefore looking into the concept of continuous deployment.

Concurrent to the normal work, a new product generation is being developed. The objective is to, in time, establish a more easily handled product, enabling faster and more efficient development and testing processes. This is done by eliminating some parts of the old complex software architecture, as well as reducing the range of hardware components. In short, they are striving to make it more simplified, to reduce the complexity by limiting the product variations and different configurations. Potentially it could include a narrow span of suggestions for customers, further reducing the network diversifications. This is claimed to facilitate compatibility aspects and enable thorough test coverage, which is important for the CI development.

4.1.2 Product Introduction

In new product development, OrgA is using a streamline model to increase speed in development of new features. The model is visualized in Figure 4 and briefly described in the following section.

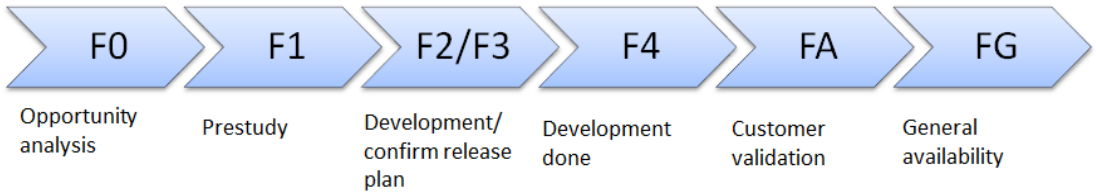


Figure 4, development flow

F0 - The purpose with the first stage is to perform an opportunity analysis, which basically is an initial analysis to identify the amount of work required and the potential business value of a specific requirement or feature. It is an operative product owner (OPO) who is responsible for this initial stage and when it is finished it can be prioritized in the product backlog.

F1 - When the feature has been accepted in the first gate, a more detailed pre-study is performed. The decision indicates that there is a consensus on feature scope, early estimated effort and an understanding of the test impact. Some of the key activities are the following:

- Anatomy to realize dependencies between features
- Feature breakdown to divide work in smaller work packages
- Developing test strategy, both in terms of quality assurance and to define which automated tests to integrate in CI

F2 - At this stage the development can begin. The development of new features will be described in detail in the 4.1.3.

F3 - The purpose with this decision is to give a clear statement with high confidence that the XFT will be able to deliver according to the release plan. This is especially important for features that includes HW development, which is associated with large investments and long lead time.

F4 - The decision that the development has been completed and is from that perspective ready to be included in a release. This means that all documentation needed, feature verification (internal) and legal and commercial activities are finished.

FA - When the decision is made to include the feature in external verification activities.

FG - The stage where PDU is considering the feature ready and recommends for the commercial sales to market (GA). All restrictions have been eliminated and it has been verified in live environment.

4.1.3 Feature flow

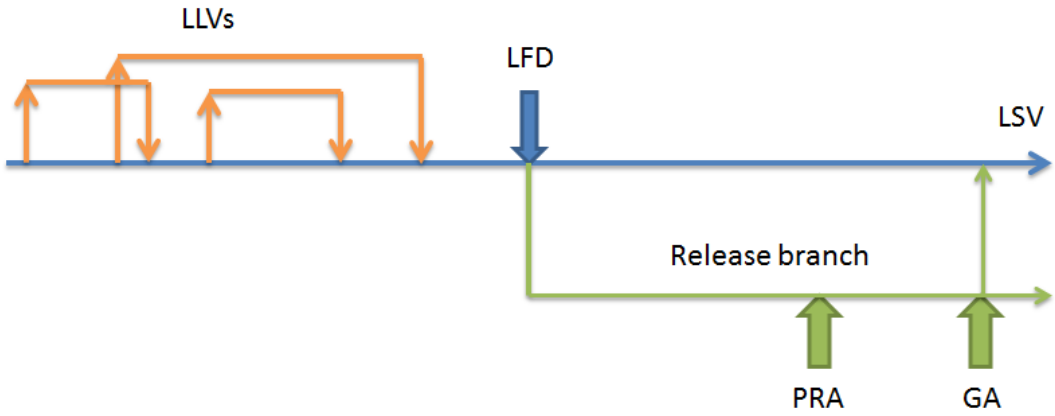
The development of a feature is usually divided into several sprints, dependent on the size of the requirement. OrgA is organized with cross-functional teams, consisting of a team leader, software developers, testers and system engineers. These teams are responsible for delivering a complete feature, with an attached regression test case that can be integrated to the CI server. Every team is committing their changes on their local team branch (LLV) that is branched out from the latest system version (LSV). Once a day, content is collected from different contributors, LLVs, and one common software base line (SBL) or “LSV candidate” is created. Teams are allowed to commit changes until the delivery closing time for the day. All team branches are however not included in the daily SBL. One team leader explains that delivery to the LSV is done when the developer is satisfied and confident that the quality is good enough and the overall direction is that everybody pushes their changes at least once a week to the LSV. After closing delivery, all the changed software modules are built, as well as the SBL. In this stage there is still manual interaction, as there is a CI team responsible to build the SBL. On the SBL, the automated tests, SAT and XSAT, are run and these either accept or reject the SBL. If the candidate passes the SAT and XSAT, it gets accepted as the new LSV. On the other hand, if it is rejected the whole package is declined and one must identify who is responsible for the failure and should fix it. Since several teams have committed changes to the SBL this can be a complex process. Moreover, on the LSV there is a final automated test, namely a basic regression test (BRT) that is continually running and contains a more complex test environment. The results from team deliveries, build information, SAT and XSAT are visualized on an online dashboard.

The automated tests do however not provide wide test coverage, since much of the old functionality is not automated. Moreover, these tests are not automatically triggered, instead they have to be manually executed. Because the automatic coverage is low, a large part of the testing activities is performed manually. These tests are conducted by a test organization that continually are performing manual regression test on functional, system and network level. The feedback from the manual test is in term of TRs, which are located in an internal IT system. It is described that these are not as visual as the result from the automated tests.

4.1.4 Release flow

As mentioned before, each XFT develops features on an individual LLV before merging it to the LSV. In every project, there is a deadline after which it is no longer accepted to deliver changes that will be included in the new release. This date is called the last feature drop (LFD). After the LFD, they branch out from the LSV, creating a new separate release branch. The quality is not considered to be high enough to deliver to customer, so at this stage the focus is redirected towards fixing bugs and trouble reports (TR) that have been identified by testers within the teams or the test organization. Since much of the system tests are manual, it is declared that substantial time is required to achieve the desired test coverage. A team leader describes that at this stage, usually a couple of months before release, there is a PRA/GA rush, and a majority of the tester´s focus on the release branch. As a consequence, approximately two thirds of the team direct their efforts to work on TRs

on the same branch, while the rest of the team continues to deliver to the LSV. This, consequently, creates the problem with insufficient test coverage on the LSV, since testers are busy working on the release branch. After a period, when the quality is considered good enough, the FA decision is made and preliminary availability (PRA) is set. It means that the feature is included in external verification activities and a chosen customer can take part of the new release for labtest and live application testing. This is called first order administration (FOA), and the participating customer is consequently a FOA-customer. Hopefully after 1-2 months of customer testing the new release will be available for all customers, GA. From the feature perspective the FG decision is made when the development organization states that feature is finished and recommended for the commercial sales to the market. The described process of new software releases is visualized in a simplified illustration in Figur 5



Figur 5, illustration of release flow

4.1.5 Challenges adopting CD

This section will highlight the different barriers that have been indicated as potential challenges for OrgA regarding a transition towards CD. The insights mainly come from interview sessions with personnel from different departments involved in the product introduction chain. Data has been further validated and extended with internal documentation.

4.1.5.1 Technical Challenges

As stated, it is an old product with many different variants and hardware setups. The product exists with countless individual configurations at customer site, creating highly diverse networks. In addition, the customers have in the past set up their networks without significant guidance regarding its composition, something that has increased the diversity even more. It is therefore not possible to cover all customer configurations in testing, since it would require an endless matrix to maintain the quality. Several interviewees express that the software has been poorly managed for many years, which has created a complicated architecture. It is said that an intentional change in one part of the software, can lead to an unintentional and often unknown change elsewhere. It is hard to understand the system and predict how an action will affect it. Instead, the current system requires a lot of human interaction, and developing a more automated build process has been stated as a necessity for

increased frequency of builds. Some challenges have been mentioned, such as enabling one master script to be used and reducing the number of build servers needed in the process. Another common concern raised during interviews is that testing currently to a large extent is performed manually. The automation coverage is currently low and, as it is explained that the test bench is constantly a bottleneck hindering progress and they have underestimated the effort required to reach the desired improvements. One other explanation is that the automation of test cases has previously been executed in separated silo, by the test organization, and lacked support from other areas. Now they are trying to implement and drive the test automation in the whole development function, including the XFT.

It is expressed that for CI, it is necessary to achieve a system with faster feedback from automated tests, to improve the integration frequency on the LSV. The complex environment is, however, making it extremely difficult and time requiring to write test cases. Implementing CI with a single software track therefore requires a simplified test environment, which will improve the integration and test writing frequency. Another problematic area, pointed out by a test engineer, is that some test cases are not reliable and can give false verdicts. Consequently, manual analysis and judgment is required, which involves unnecessary man hours. It also creates an uncertainty within the organization, if an uncertain build manually passes and possibly will degrade the previous state. This fact, in combination with several team commits on every SBL build, has in some way created low sense of ownership among developers and a reduced responsibility taking for fault correction. The low ownership is further increased by a functional mindset, as product and test code are usually divided in separated sections. As test code have a lower priority in the organization these also remain unsolved, which further induces unstable test system.

In relation to this, several interviewees have expressed that a single software track is a requirement for adopting CD. This, in turn, raises the precondition to make frequent and small changes that are committed directly to the LSV. Today, OrgA has an extra step, as the XFT are developing on LLVs before committing their changes to the SBL. The bottleneck of adopting a main track is to establish fast test gate sweeps, which is challenging due to the noted variety of configurations. Backward compatible changes are also mentioned as a requirement when moving toward CD. Moreover, when delivering more frequent, code dependencies become important to handle. One interviewee states that it is important to have a plan for how to handle this with CI and more frequent deliveries. One example is to have a strategy on how to handle a major holdup. If not, teams might continue to develop new code without integrating it into the LSV. This leads to a major integration later, which would probably crash the system.

Quality is seen as an issue that will need to be handled for a change towards CD to be possible. Today, the quality on the LSV is fluctuating and is more or less only ready for deployment every half year. It is expressed that a long time is needed for bug fixing to achieve a sufficient quality level before release. One interviewee explains it as a negative spiral, occurring since resources are split between the LSV and PRA branch. The worse quality on the LSV, the longer time testers and developers have to direct their efforts to the PRA. At the same time, the quality is

decreasing in the LSV as new functionality is continually added. Moreover, focus is claimed to be on testing legacy and not enough on the functionality delivered. In the next step, when these new builds are part of the legacy, the SAT tests often reveal a lot of errors. The complexity of these upgrades entails certain risks of quality deficiencies, which can cause network downtime. Quality related issues has meant that these activities are performed during night time and that substantial resources are allocated to the deployment process. Unforeseen software bugs or incompatible configurations revealed during deployment can be very costly for customers. Activation of an upgrade also leads to a few minutes of network downtime, which poses a barrier to frequent releases. Avoiding the activation downtime requires a solution with double cores, which has been used in another organization and is claimed to be possible for OrgA as well. However, the fact that OrgA is dependent on so many nodes, makes a solution even more complicated.

It is also expressed that the team developing a feature should be responsible for writing the tests before an F4 decision can be made. However, according to a source within PDU, integration of the tests takes too long and is therefore done too long after delivery of the feature. An explanation mentioned is the complex test environment making it time consuming and difficult to write automated test cases. It is also expressed that OrgA is lacking competence in terms of automated testing and that there has been low support from management regarding automatization. As a result, when the F4 decision includes the completion of test cases, the feature lead time has increased and it is delivered closer to the LFD, even if the feature was completed much earlier. In line with this, the target of reducing lead time has not been reached and it is described to been challenging due to the low amount of features, where most of them are large in nature. Also, it is claimed that high quality focus is influencing lead time negatively in the short run and that there is low commitment for this goal in the organization.

4.1.5.2 Organizational challenges

From several interviews it has been emphasized that before adopting CD, they have to successfully implement CI in the development organization. A common concern is the lack of management support and an underestimation of the efforts and investments required to succeed. One interviewee describes that one miscalculation is the effect of having development on three different sites and the differences in culture. To overcome these differences, it has been argued that an increasingly involved leadership style must be established. Today, there is rather an assumption that the collaboration and alignment between sites will work seamlessly. One other perception is that the organization does not work and follow up as initially defined. Moreover, that the overall goals with CI and automation could be much better communicated in order to reach organizational consensus. It has been expressed that OrgA uses several KPIs in the development, but are they are not adequately communicated and understood in the XFT. Some off them are actually seen as unnecessary and irrelevant from their perspective. KPIs were they can actually see the improvement is considered to be motivating. In addition, regarding quality and TRs, the broken window effect is referred to, as it is easier to leave faults unsolved when there are many of them. In line with the previous, communication between

development sites seems to be problematic. Today, development is divided in three different locations. This is claimed to complicate communication, and to make it more time consuming to come to agreed upon solutions to issues in the development. According to a few sources this poses a major barrier, and it is crucial to be able to manage site development with an increased amount of deliveries into the CI server, as well as achieving an internal consensus on future goals.

Different views internally on CD are brought up, as the objectives vary between departments. The sales organization and product management are commonly mentioned as possible resistors. They are deemed to be more cautious and distrustful regarding the impact on quality from products released into the networks as the frequency of upgrades increase. Interviews with a product manager confirms this scepticism towards more frequent releases, but reveals a very positive attitude towards continuous integration and earlier as well as more frequent testing in parts of customers' networks. The need to also understand the current internal organizational structure is emphasized by one interviewee, as a change towards CI may imply alterations to it. As an example, it is mentioned that the current test organization might become redundant with CI. Interviewees in OrgA have also expressed low trust in the coverage possible by automated testing and resistance in completely abandon manual testing. Moreover, a sales representative explains that they currently lack the internal system to support a version-less product, which CD would imply.

Challenges related to breaking down features in smaller functional parts are viewed as a barrier. Currently Org A is working on a concept called minimum marketable feature (MMF), as they are aiming to rapidly deliver the essential functionality. This has several value enhancing aspects, such as faster return on investment and earlier feedback from customers, making it possible to readjust development. The problem of breaking down features is, however, expressed as a challenge in product management. It is described that product management needs to see the value of it, adapt their way of working and prioritize the list of requests so that it can be broken down and delivered in smaller pieces. Today it is expressed that they rather believe it is more efficient to develop and deliver everything at once. A suggested explanation for this is an unhealthy partition between R&D and product management. Another statement is that product management does not have the time to familiarize with the functionality, not until there is a crisis close to deployment, forcing them to prioritize. On the other hand, from the point-of-view of product management, it is important to understand the customer and if they really are demanding new features on a continuous basis.

It is crucial that a new release does not cause problems to the network operations, which is further complicated by the diversity of customer network configurations. Today, customers do not trust new software releases, since they generally mean big problems. Software bugs and other issues usually complicate the upgrades, and can be very costly for the customers. Consequently, customers also thoroughly test new software before deploying it in their live networks. In addition, customer has established systems to revise new software two times a year, most customers even more seldom than that. A major challenge to adopting CD is consequently perceived

to be resistance from customer. It takes time to build trust as one respondent stated. Finding lead users for the CD concept makes it necessary to identify customers that are willing to change their own structure regarding deployments; customers that are willing to take more risks and reduce their testing of new software, in order to roll out new functionality fast and more frequently.

The challenges perceived at OrgA are summarized in Table 9 according to frequency stated in interviews. However, the frequency should be interpreted with some caution. As data have been collected from different department, some challenges with lower frequency could still be as important as those with high frequency, but only realized and brought up by one part of the company. With this said, those that have been mentioned in a majority of interviews are of course important barriers that needs careful attention.

Challenges	Frequency	Explanation
Diverse network configuration	High	Diverse and local customer configurations making it hard to achieve automated testing and sufficient test coverage
Legacy product	High	Difficulty to build automation on a mature legacy product
Automated test coverage	High	Old functionality is currently manually tested, which means a low automation coverage (15%). There has been a lack of management support.
Regression feedback	High	The runtime of SAT and XSAT is long. Creating long internal feedback loop.
Network impact	High	Software updates associated with high risk and negative network impact
Ever ready software	High	Months of bug fixing on GA branch. Dividing resources between LSV and GA. Software deployable every 6 month
Customer trust	High	Customers do not trust the quality of new updates and have substantial verification system not adopted to CD.
Short branches/smaller deliveries to main track	Medium	Adopting software practices where small changes are continually made into one software main track. Integration frequency and test runs are bottlenecks.
Different development sites	Medium	Coordination between sites making it hard to achieve a consensus in goals/vision. Also, to maintain the CI machinery with more frequent deliveries.
Internal resistance	Medium	All departments do not see value with CD, especially those closer to customers. Resistance to abandon manual testing.
Unclear goals	Medium	The goals regarding CI/test automation have been vaguely defined and communicated in the development organization.
Management support	Medium	CI/CD requires severe investment and management support, which have historically been lacking. Site development also needs more active leadership.
Test environment	Low	False verdicts in test cases creates manual work and low sense of ownership from designers

Code dependencies	Low	When integrating code frequently, code dependencies can be problematic.
Reduce build time	Low	Aspects of build process are still manual and every builds cannot be built on the same server. Not all product can be built by script.
Acceptance testing	Low	Testing is performed both in development and by customer
Breaking down features	Low	Product management does not see value in breaking down features.
Backwards compatible changes	Low	For CD it is described that backward compatible is a requirement
Lead user	Low	Finding customer that is willing to reduce testing, take more risk and adopt frequent deployment

Table 9, summary of the perceived challenges towards CD at OrgA

4.2 Organization B

This section describes another organization in the case company, who introduced a CD setup in 2014. It will present their product, the current processes for new software development and the challenges and strategies highlighted by employees involved in their transformation from traditional software development to CD. The organization is large and involves around 2000 persons in 10 sites. Today, there are hundreds of thousands of these products out in the world. It is a big and complex product, with several million lines of code that has been developed for almost 20 years.

4.2.1 Continuous Deployment Process

The overview of the development process at OrgB is summarized in Figure 6. It is based on several XFT, each team responsible for an entire feature from development to writing automated test cases. As can be seen in the picture, they are using a software main track where feature content can be added continually. New commits from XFT are tested, before delivering approved builds to the CI server. The quality is checked in block tests and the results are visualized for immediate bug fixing, supported with fast build time and block test, allowing the designer to receive fast feedback. After completion of a build, the software is installed on a node, tests are run and results are analyzed and visualized in a completely automated chain. These tests are performed frequently as long as there is new software to test. In addition, there is also a separate team dedicated to main track management, which is responsible for the quality and feedback regarding the main track. This group can remove deliveries or order design stops, if necessary to improve the state of the main track. Their CI machine has made it possible to have almost constantly deployable products a condition for CD.

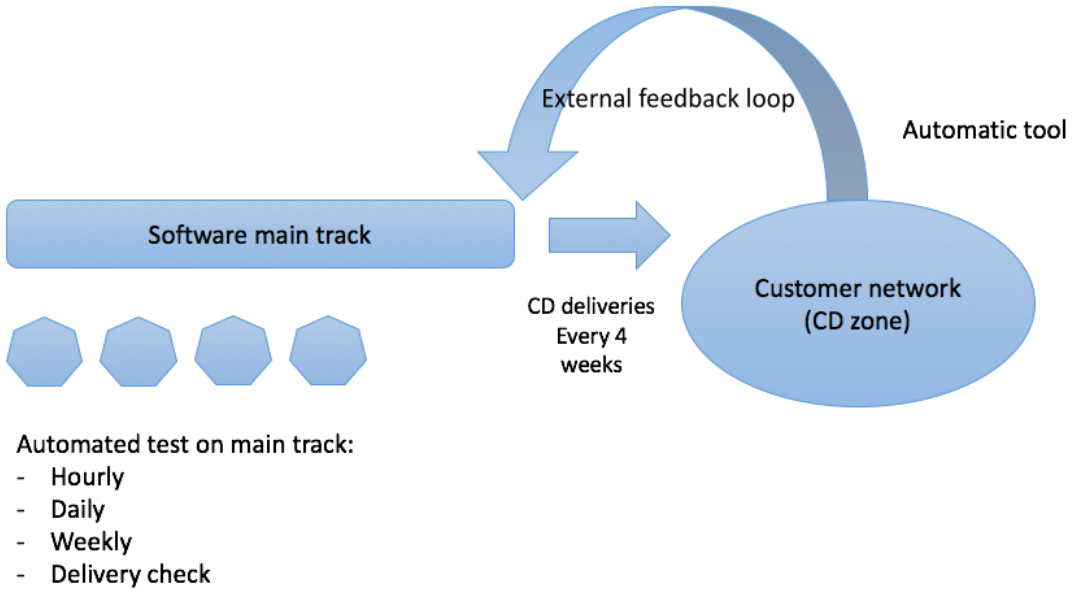


Figure 6, Continuous Deployment process at OrgB

Deployment is made every fourth week, including software not officially released, both in terms of corrections and new functionality. It is given to chosen CD customers and is initially tested in customer lab environment, then deployed live in a small, but representable part of the network. The initiative is seen as long term partnership with these key customers for specific products. It is described that the main purpose with CD is to secure legacy functionality and characteristic and testing new functionality. CD has allowed an external feedback loop, where the results are analyzed and provided back to the development. All CD deliveries are taken from main track, and release track is kept short. Even if frequently deploying to customer, GA release is still maintained at every six months.

4.2.2 Barriers to Continuous Deployment

During the spring 2010, OrgB performed an analysis of the current situation and decided that a radical change was required. The response was the decision to implement lean and agile methodology in the organization. A system breakdown during the summer became a further eye-opener that a change was needed. Management realized that the first step in their transformation should be to implement CI, which was seen as a precondition for lean and agile. In the beginning of autumn 2010, the implementation of CI began and specific goals related to the following were defined:

- Efficiency
- Lead time
- Quality
- Empower people

4.2.2.1 Continuous Integration

In the beginning of the CI transformation, OrgB experienced low and fluctuation quality on the LSV of the software. Software deliveries took too much time, and after

every design delivery a lot of time was needed for debugging to achieve sufficient quality to even start testing. The debugging was required before starting feature verification, light system testing, and then also for the product to be considered good enough for heavy system tests. Features were commonly developed on side tracks, and merged through large big bang integrations at the end of the track. Feature development had to be stopped long before release, to instead focus on correcting faults and stabilizing the product.

The main objective with CI was to achieve constant high quality and enable a faster internal feedback loop. In order to accomplish this, they adopted a 3-step process. In general, they needed to drastically reduce build times, to minimize the lead time from designer to test on all system levels and finally to automate testing. These initiatives will be discussed in the remainder of this section.

Before CI, the builds were slow and unreliable, because of large volumes of code. Also the testing in the design process was a problem, due to long execution time for block tests and load module. Furthermore, the results from builds and testing were not visible for the designers and fault correction times were long, stretching over days or even weeks. In other words, developers did not know if the delivery was successful or not.

Initiative	**Why?**
Reduce build time	Too many commits per build Failing builds occurring frequently
Reduce block test time	Slow tests and feedback
Increase visibility of failed builds	Correcting time was long due to low visibility for the developer
Reduce number of side tracks through backwards compatible features and small frequent changes	Focus test capacity to one track

Table 10, initiatives in the CI implementation

Decreasing the time for the build process and design near tests, along with an increased visibility of errors, led to fewer failing builds and a dramatic reduction in time to fix the rest of the errors. This is believed to be due to the increased visibility of build errors and an intensified sense of ownership of these problems among developers. To reduce the delivery time, the build-test-delivery flow was also changed into one where designers were only to do the most important quality assurance activities before delivery and the less important after delivery. Additionally, system engineers were asked to make all features backwards compatible to avoid side tracks. With one single SW branch, they could also direct their entire test capacity towards that single track and achieve much better test coverage.

The automation of integration (node) test did not have the same effect on correction time as in the build process. The reasons behind, were considered to depend on several factors:

- A more complex environment and lower reliability for these tests, which sometimes lead to false fail verdicts
- Contained many deliveries from different teams in each run.
- It may be an old problem that have existed in earlier test runs and therefore caused by someone else
- The correction time is longer than the time between test runs

These factors eroded the confidence in the test results and the responsibility taking for the issues, since the problem probably was caused by someone else.

To mitigate these issues, the reliability of the tests was increased with improvements of the IT environment, the test environment, the test tools and the test suit. In addition, the test environment problems were separated from product faults and a team was instituted, dedicated to test environment issues, allowing developers to ignore these ones. Furthermore, it was expressed that shortening the test run, was the most important aspect to achieve automatisation coverage, since many new deliveries per test run will decrease the potential success rate. For example, the likeliness that a group of commits is fault free, if each commit has a 95% chance of being fault free will decrease exponentially according:

- 1 commit: 95%
- 5 commits: 77%
- 10 commits: 60%
- 20 commits: 36%
- 50 commits: 7.7%
- 100 commits: 0.6%

It is described that they are now managing to run tests with a frequency so high that a majority of tests are accepted. In contrast, before they implemented CI, they could not perform a single test without discovering an error. Discovering many faults at a later stage means extra work, as it has to be reported to design, who should perform an analysis and later an attempt to correct and test it again. Instead of focusing on automatization, testers are rather spending their time correcting faults. If the organization is succeeding in running the test more frequent, the quality will be higher in each run and tests can run even more frequent. It is described to be a positive loop when the tests are run so often that the possibility of findings errors is relatively small. At the same time, it is a threshold that needs to be surpassed.

Initiative	**Trouble areas**
Automate node tests and visualize Improve IT/test environment etc Separating product fault and test environment problems Instituting a team for test environment More frequent testing	Low reliability in test results Lack of ownership Long correction time

Table 11, Initiative for automating integration tests in CI implementation

A remaining problem was that the correction time was longer than the main track was allowed to be broken. Consequently, OrgB introduced main track management. This

is, as earlier mentioned, handled by a specific function, responsible for the quality of the main track and the flow of feedback from the tests. For this task, the group has the authority to remove builds that cause problems and to order design stops, meaning that some or all teams are only allowed to deliver corrections and not new functionality. This also allowed them to correct error outside the main branch and re-delivered when it is corrected. Throughout the improvement process, OrgB had a strong focus on reducing time to feedback and these initiative heavily reduced the time for node testing to designer.

To achieve a constant high quality and automate all integration and verification tests OrgB used a systematic approach. Test cases was gradually automated and test suits run could be executed more frequently. The main track previously established was a major part in making this possible, as test efforts could be focused towards one single branch. Also in this stage, the results were visualized on the monitors to improve time to feedback to designer from heavier and more complex testing environments. The results of the initiative were that they are now able to run 90% of the test scope every week, previously performed once before an GA release. This has a very positive impact on quality and the product is almost constantly on a deployable level. In addition, all of the previously mentioned debugging periods, which is done before testing of the software can start, is more or less eliminated. However, test result evaluation and verdict have been more challenging to automate. The errors from these system tests are often complicated and take long time to correct and the challenge has therefore been to reduce that time. Two different efforts, both with promising results were set up to handle this problem. Machine learning and artificial intelligence has been an area for automating test result evaluation and verdict. Moreover, they have co-located some test teams with cross-functional correction teams and allow them to work closer together. This has enabled significantly improved correction time and they are considering expanding to cover all system test areas.

There has also been a wish to branch out the release version later in the process, closer to the official release. The strategy involves taking all FOA deliveries from main track, making a design stop prior to final branch, targeting acceptable release quality before the release branch is created, and making it ready for deployment on time. New feature development is kept to the main track. Maintenance releases are then made from the release tracks, but first after being implemented and tested in the main track. This effort has built on the intensified focus on tests in the main track and an increased coverage of these tests, and has led to an increased feature content and reduced TR mapping between branches. The improved control over main track quality also offers a better starting point for the work with the next release. The initiative discussed on regard to automate system verification test and branching strategy is summarized Table 12.

Initiatives	**Solutions**
Automate I&V	Gradually automation Increase frequency of test runs
Automate verdict and evaluation	Machine learning and artificial intelligence Co-locating testers and teams

Later branch out	FOA deliveres from main branch Quality before release branch is created Feature development on main branch

Table 12, initative to achive an constant deployable product.

4.2.2.2 Organizational changes

When the initiative to implement CI was decided, severe investment was required. For example, they needed to purchase additional test equipment to spread the tests on more test sites, new IT equipment to support testing and allocate employees actually implementing the change. They managed to pursue CI within current budget by postponing other investments and activities. Due to increased productivity the work previous delayed was however done within a year. After three years they could perform much more testing per month with less personnel and a constant higher product quality. It is described that it should not be seen as a cost, it is rather an investment towards improved quality, efficiency and lead times. However, investments of this size must be decided and supported by PDU and PL managers.

Moreover, when initiating the transformation in 2010, radical organizational changes were implemented. Mainly the centralized line organization was restructured into a decentralized organization based on XFTs. A major challenge is described to be related to large scale development and achieving consensus among thousands of employees and to strive towards the same goal. These kinds of change processes face resistors and it is described to be those who are expected to personally lose status after the change. Moreover, it is usual that people's attitudes to the change are scattered and in those cases, the effect will not be great. OrgB met the challenges by continually communicating the vision and why the change was needed. There is a need for consistency since people that not are affected yet, will probably not pay full attention. Moreover, they recommend including everyone that is interested, for example through open space seminars.

As one main goal with the transformation was to reduce lead time and improve time to feedback, OrgB needed a strategy to achieve the desired time reduction. It is described that XFTs was the most critical aspect for achieving the objective. With the XFT, they became responsible for completing an entire feature and handovers could be eliminated. This means that for every new feature, the XFT is also responsible for developing the regression test cases, used in the CI server. In order to avoid delays of feature introduction into the CI server, the definition of complete has been changed to included activation in CI. The change was actively driven and supported by PDU management. Since development is located in 10 different sites, it was challenging to spread the new practice. Nowadays, all XFT are co-located which facilitates communication. This required broadening of competences on sites, since most of the testing and systemization competences were located centrally. To achieve a decentralized system across different development sites with XFT required a great effort over several years of relocating employees and programs for competence widening. Moreover, handling the transition, one suggestion method is to allow the XFT to be coached by an experienced scrum master. In order not to lose momentum is the progress, it has also been important to constantly visualize and communicate the improvement that have been reached, as well as avoiding big impediments, causing mass-collision.

Another organizational barrier when adopting CD has been changing culture. Before, code was committed in larger batches and it was rational to believe that it was another fault that caused the failure. In addition, with more sequential development, responsibilities were defined between functions, creating a sense of “someone else is responsible”. A major part has been to change this mentality and create a joint ownership of all involved. It is described as a challenging task that takes time and does not happen overnight.

4.2.2.3 Continuous deployment

The implementation of CI was critical to constantly have quality on the product. This is, however, far from everything that needs to be in place for adopting CD. The following section will describe what OrgB did when adopting CD.

When the actual implementation toward CD started at OrgB, it was initiated by two factors. The first was time to feedback, in this case the external feedback loop. Internal testing does not detect all faults and there are some faults that cannot be recreated in labs. One driver was therefore quicker feedback from live application in customer networks. The second motivator was increased quality in the GA release. Achieving faster feedback makes it cheaper and more efficient to correct faults, and was expected to increase the overall quality.

However, adopting CD was faced with customer resistance. OrgB have large and complex products, previously associated with negative impact during deployment and the customers were used to complications at deliveries. The customer was aware that there was always a risk involved with taking a new software release. Installing new software on a network of nodes requires serious resources, as it can only be done with limited speed and during the night. Adopting CD can therefore be conceived as taking this risk more frequently and to higher costs. In addition, the customers have comprehensive quality assurance procedures before updating their network, usually involving months of testing. CD was completely different to this way of working, with more frequent and smaller changes compared to previous with a few large deliveries. They had to develop the capability to perform a more limited quality assurance and be able to roll back the software if it caused problems, requiring a change of focus from striving to prevent problems to being reactive. To overcome the customer resistance, OrgB had to motivate the change with the perceived benefits. Working with CD, the customer would experience a closer collaboration with PDU, which can be valuable due to access to expertise. Moreover, they will receive faster access to functionality and detect problems at an earlier stage. In this way, the operator will prepare the whole network for faster and more reliable roll outs of the software.

Company drivers	**Customer resistors**	**Customer benefits**
Faster time to feedback (external) Higher quality in GA	More frequent risk taking Changing quality assurance procedure New capabilities being reactive	Closer collaboration with PDU - Increased feature understanding - Faster fault turnaround Earlier access to new features Earlier detection of problems - Also in devices and surrounding system

		Earlier adaption of surrounding systems Faster roll out of GA

Table 13, the stakeholder's perception of CD

It has been described that some customers are more likely to adopt CD. Usually it is operators who want to profile themselves as premium suppliers and therefore want first access to new functionality. In those customers, OrgB could find lead users and form a partnership. The collaboration with a few key customers is seen as a long term engagement, with the purpose to secure legacy and ensure that new functionality works. After the first implementation in 2014, they have received very promising results and actually proved that it works in reality. As a result, the customer skepticism has started to wear out. Moreover, one aspect has been to always include functionality that increases customer value. Otherwise, it will be hard to convince customers to change their processes and accept the increased risk of more frequent deployment. This is challenging since they have maintained two GA releases that can trigger an unevenly flow of feature, peaking around these releases. In order to create an continuously and even flow, production leveling in internal processes is needed. OrgB has also actively driven this change by creating awareness in PDU.

When deploying every month, it is also explained that it may require additional administrative work with documentation on every CD release. It is necessary to make the delivery of documentation more continuous, and find ways of keeping the customer informed about the changes. OrgB has had meetings with customers and continuously communicated the contents of the releases, but the bottom line is that this requires a defined strategy for how to handle documentation regarding CD deliveries.

OrgB´s product does not contain several variants, as is the case with OrgA, it is rather a united product. However, there exist several optional functions, possible to switch on and off. The customer is also using a diversity of hardware combinations, making it impossible to cover all customer configurations in lab environment. In order to mitigate these settings, a clearly formulated test strategy is required on how these functions are activated and deactivated in their test facilities to reach the desired test coverage. Also, it has required a representative installation of lab equipment to cover the hardware setup at customers. As it is not possible to achieve 100% coverage, it is about focusing on the commonalities and taking a calculated risk.
With more frequent deployment to customers, the updates need to have high reliability and low impact on the customers' networks. For OrgB, every time an update is initiated, the network is switched off for a couple of minutes. This is not ideal, however at the current status, it is considered to be good enough. However, this problem is a great barrier toward CD and requires comprehensive product development to get in place. Furthermore, one challenge for OrgB has been to detect if the new updates causes problems in the customer network and in that case, be able to roll back the software. In order to solve this problem, they have developed an automatic data analysis tool, which are 10x more thorough and 1000x faster than traditional manual methods. The tool is operationalized out in the field, where the CD-zones are monitored. The traditional FOA procedures are being replaced, and instead of having teams dedicated to one customer network, the teams have now

been combined to one central function that monitors all of the networks. This team works closely with PDU. This provides them direct access to the developing organization and thereby the people who are supposed to solve potential issues, which enables fast corrective actions. It is emphasized that the previous support process, involving CSRs that are first created and sent by someone and then have to pass an additional number of steps before it finally reaches PDU, is not appropriate for this new way of working.

OrgB have a continuous feature development and when deploying, a snapshot of the latest software is released. This implies that half-ready features will exist, no matter when it is taken and present the challenges of always making deliveries legacy safe. Otherwise there will be a risk that deployment causes failure on something that previously was working. It is described that if they are performing changes that possibly can ruin legacy, they are switched off at the moment. This boils down to always making sure to make legacy safe deliveries.

As it was before CD, they frequently performed interface practices on node external interfaces, something that usually caused problem for external systems. When deploying more frequent it is not possible and there is consequently a requirement to be backward compatible. Hence, they have defined new design rules and it is no longer allowed to commit incomplete interface changes, which creates restrictions to how you can systemize. Moreover, they have tools that automatically detect compatibility issues and dedicated compatibility tests. In those cases incompatible changes cannot be avoided, it is described important to have strategies for how to handle such events. In line with this, a methodology is also needed to deal with changes associated with high risk. Previous, this kind of changes was performed when it was long time to customer deployment. In today's product introduction model there is no such thing as long time to deployment. They have managed this barrier to break down large changes with high risk and are making more frequent smaller changes, where every part is less risky. Since CD requires these kinds of new processes, it is important to have practices in place for this when adopting CD. Especially, developing an understanding and alignment in the whole organization on how to deal with these changes. Another challenge described, that needs to be handled, is emergency support for the software and how to deal with sudden and unexpected network failures. The case company's support organization is only concerned with software that is officially released. At the moment CD is unreleased software that is deployed in the CD zones. In order to mitigate this challenge, WCDA has implemented a solution based on a combination of internal support that is available 24/7 and people on standby around the world. This is, however, described to have several solutions, but the most important to be prepared before adopting CD.

The challenges previously discussed and how they have been managed in OrgB is summarized in Table 14.

Challenges	**Explanation**	**Solutions**
CI process	To establish CI have required serious investments	Postpone other activities. Management support Several stage to automate testing

Improve build process	Slow and unreliable builds with long correction time and a lack of visualization.	Several initiatives (figure x) Build monitors creating visualizing and creating ownership
Automation on a mature product	Slow testing and long feedback loop Debugging software before testing	Investment in test environment etc Visualize the result Automate legacy and increase frequency in testing
Test environment and low ownership	Low reliability in test results and more deliveries per run created lack of ownership and long correction time	Separating product fault and test environment Instituting a team for test environment
Automate verdict and evaluation of system test	Complex problem and long correction time	Co-locating teams Machine learning and artificial intelligence
Every ready software	Before CD the quality was fluctuating and required debugging before test and long correction time before release	Automation of test (above) Main track management team Reduce branching
Network configuration	Divere customer configuration	Strategy to activate/deactivate functionality and representative hardware in lab environment.
Customer trust	Due to network impact during updates, CD meant a frequent risk from the customer, as well as resources to update more	Find lead user that are willing to change their verification process. Premium supplier who wants the new features first.
Organizational change in large scale development	Organizational inertia during such a large change. Hard to achieve consensus and spread new practice on all development sites	Communication and share why a change is needed. Project for widening of competences Visualize and communicate progress Coaching (scrum master)
Culture / new resp. and roles	Changing the culture was required in order to creating a sense of joint ownership	Communicating the change and reasons behind
Improving lead time	Long lead time for feature development due to several handovers.	XFT Complete responsibility of features (avoiding handovers)
Network impact	In CD fast and reliable updates are required, which is challenging to achieve.	Requires serious product development to reduce network downtime when updating.
Faster detection and correction of problem	When deploying more frequent one must be able to detect bugs are fix them fast	Automatic data analysis tool Team responsible to manage the feedback flow
Backward compatibility	Changes can cause problem for external system if not backward compatible.	New design rules Tools for automatically detect compatibility breaks Strategy for incompatible changes
Continuous adding of customer value (unven flow)	For customer to consider CD, each deployment must create value for the customer. 2 GA releases can trigger an uneven flow of feature.	Levelling of processes. Management actively drive the change and create awareness in PDU.

Risk in deployment	High risk changes and delivery of incomplete features must be managed when CD deliveries are snapshot of the main track	Break down large changes in small frequent Architecture that allow to turn on/off feature Backward compatible changes
Documentation	CD requires new documentation processes and evaluation of how much to provide each month	Continuously adding documentation to inform the customer.
Emergency support	CD deliveries cannot be supported by the usual support organizational.	New setup to deal with emergency network failure.

Table 14, summary of the challenges experience at OrgB when adopting CD

4.3 Organization C

The following section will highlight the findings from interviews and documents regarding OrgC and their ongoing transition towards CD. OrgC is a newer product generation and the development of OrgC started about 10 years ago. The organization is large, technology is still developed rapidly and new features and functionality is delivered with every software release. Similar to OrgB, this product constitutes the end contact point to the consumers.

4.3.1 Continuous Deployment process

OrgB is considered to be predecessors to the CD initiative at OrgC. OrgC could consequently take part of previous experience and avoid pitfalls made by the closely related organization. The initial driver for CD was internal and they perceived CD as a possibility to achieve constant high quality on the main track. There existed some early external resistance and their first attempt to convince customers failed. However, the next year, in the beginning of 2015, their first CD customer was determined. Customer resistance was also short-lived. Today, they have 7 CD customer and several customers desire to sign up. It is expressed that the external drivers have been an important factor to motivate the change. An explanation to the customer momentum toward CD is the large part of new functionality continuously released - functionality that several premium suppliers want to be first to launch in their network, which can be leveraged in marketing strategies. In addition, adopting the software continuously will increase the confidence in the product. At GA, the rollout will be much faster, since much of the validation has already been conducted.

The development is conducted on a software main track, where feature content is continually added in small deliveries. Since the product is complex and is developed by over 100 XFT, it has consequently been divided in modules. Every module passes in a flow with gates, where tests are triggered, executed and given a verdict automatically. Before the modules are integrated on node level and allowed into the main track, there is an additional gate in a node baseline check that is running on an hourly basis. The results of these tests are visualized and separated between product and environment faults. Finally, there is a verification test on the compounded product.

There is a bi-weekly delivery to 7 CD customers, who are performing a fast lab test before the release is deployed into a limited live network. The first day after upgrade,

the KPIs are checked on an hourly basis, thereafter they are controlled daily by PDU. When issues and TRs are found, they are directly handled by PDU with a correction target on next available delivery.

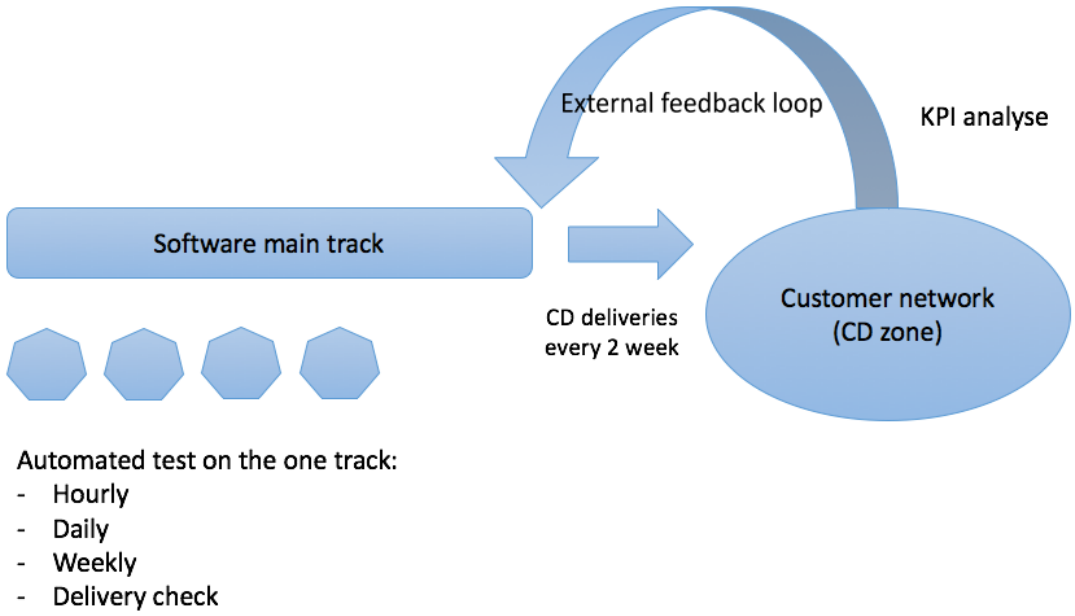


Figure 7, continuous deployment at OrgC

4.3.2 Barriers to Continuous Deployment

CI is considered the foundation in OrgC´s CD machinery. It provides the infrastructure for the developers to continually check in changes to the main track, as well as having a verification loop continually running to secure legacy. OrgC has experienced a huge advantage in relation to automation, compared to the other two departments since it is a younger product. When the development was initiated in 2007, there was already a consensus in the organization that automation was crucial, which facilitated the adoption of CI. However, a problem mentioned is that establishing CI rarely is allowed to cost money. One interviewee describes that in general, the company have not been investing enough money to achieve the required quality for CD. It is said that investing money in CI has been an important learning for OrgC. After adopting CI, they are now, 5 years later, able to run the same amount of test cases every week, as they previously did once every half year. The increased frequency that automated tests have meant also required investment in a new machine part, containing a wide variety of test environments. One challenge when automating testing, highlighted in interviews, has been to automate verdict in complex environment and deciding when it is a product or environment fault. To mitigate this, they have developed a software solution, performing the analysis and visualizing the result. Getting to this state has required a systematic way of working, in order to understand, and instruct the machine, when the fault origins from the test cases.

Moreover, OrgC experienced difficulties managing the CI funnel, which origins from large scale development. The development organization consists of more than 100 XFT delivering to the same track and test machinery, making the system vulnerable. When the external deliveries are at such a scale, it presents the problem of identifying which delivery is causing the failure. For this reason, it is explained to be a constant battle keeping consistent quality on the main track. Occasionally, a delivery stop can be decided, when the quality is far from sufficient. On the other hand, a delivery stop is extremely costly with more than 100 team not allowed to deliver new functionality. To manage this problem, they have divided the product into modules for pre-integration and verification. In line with the former problem, OrgC is consequently struggling with having the software in an ever ready state of deployment. When deploying as frequent as every two weeks, the quality needs to be sufficient for a fast lab test, before it is deployed in a live network. A potential vision is to deploy the software in CD releases nationwide, but it is expressed that they are not there yet, quality wise. In addition, even if able to deploy every second week, these CDs drops do not contain all the functionality wanted. In order to mitigate the quality problem, an internal presentation suggest that the XFT must feel responsible to keep quality and understand the customer plans and be involved early.

The situation is further degraded with an uneven flow of requirements, a problem triggered by maintaining two GA releases per year. It is described that this setup influences a behavior in planning, not appropriate for CD. Planning is still adapted towards the requirement deadline of these two releases, which creates two peaks in requirement inflows every year. The problem boils down to the willingness to include more requirements than can be handled, as well as the difficulty to communicate the release scope six months before release. Since this scope is promoted to the customers, it cannot be removed and creates several feature delays and a “tail heavy” development. To solve this problem they are working on improving the planning prediction, including a longer horizon where they can acquire more detailed knowledge of the requirements beforehand. However, it is pointed out that they must be aware of the trade-off between early dedicating resources to a requirement, with the risk of it being outdated or deprioritized.

After adopting CD, OrgC gained customers at a rapid pace and is now collaborating with seven CD customers. Managing all these different customer interfaces is said to be a challenge. It demands human resources in the form of project managers and on-site support dedicated to every single one of these customers, as well as an increased number of employees in the KPI team that is monitoring the upgrades and customer networks. It is expressed that the amount of customers consequently has created internal work that they were not prepared and staffed to manage. In other words, they pushed the system, realized bottlenecks and are now having trouble handling it. To mitigate this challenge of moving too fast, they are not working with more customers at the moment, instead focusing on stabilizing the current situation, while improving quality. Another related challenge has been different definitions of CD and what it actually means, both among customers and local internal organizations. A product manager describes that on a majority of customer meetings, CD is brought up as a universal solution to a variety of different problems. At the

same time, it is not obvious what the customers want to achieve with CD. The word CD is hyped and a common definition is consequently needed.

Before adopting an agile methodology, the development at OrgC was associated with large changes that often interfered with legacy. These changes could be modified and later tested again, until the system worked again. When deploying to customer every second week, it is not possible to work like this. As a consequence, changes must always be backwards compatible. Furthermore, if a big change is required, it must be broken down in smaller changes, making sure it does not interfere with legacy. In other words, they had to develop practices of making commits that always are legacy safe. As the releases are the latest accepted versions of the main track, there is also a requirement of the software architecture to be able to turn on and off incomplete features and system improvements.

Even if these actions have been taken further, monitoring after delivery is required. Therefore, there is an internal KPI team dedicated to analyze the customer zone, in which they deploy. They are extracting data in order to detect changes on the KPIs, and this can trigger new actions. It provides instant feedback from the live network, reducing the lead time from design to customer. If the customer experience severe problems with the software, it can also be rolled back. Here it is suggested to include collaborative customers that are willing to allow time to find root causes, and not immediately roll back at any indication of trouble. Adopting CD has allowed them to reduce the GA rollout time markedly, which means both less costs for the customer, as well as earlier payment for OrgC. The way of working has basically moved maintenance work usually performed after GA to finding faults before GA, using the external feedback loop to make corrections. The external loop can be further improved by allowing the XFT to look into the automatic data collection server in order to understand how the feature is used. Also, by allowing more access for the CU to collaborate with the XFT and by motivating the teams to keep a constant high quality.

OrgC experienced some internal resistance before adopting, especially from product management, as they did not realize the value of CD. There was a fear of deploying functionality to customers that was not mature, and that the company's quality stamp would be damaged. However, it is described that since the work actually started with CD, it has more or less been a fact that CD is the future. This has been easier to motivate with many customers excited about CD. Moreover, there seems to have been an internal mismatch regarding the objective with CD between product management and PDU. PDU has on several occasions been delayed with features and missed the GA release. It is perceived that the feature could as easily be included in next CD drop two weeks later instead. This is however, a major misconception, as the customer cannot use it in the whole network and a regular customer cannot take part of the feature. As a product manager explains “It is hard when PDU uses CD to get away and avoiding plans, but that argument is not arguable against customers”. This problem origin from the lack of knowledge in the PDU and XFT and is consequently mitigated by educating PDU about the purpose of working this way.

In line with this, OrgC has been working to communicate the purpose and overall goals of the CI/CD process. There has not been a consensus in the organization of why they have performed this change, which makes it difficult to conduct a meaningful internal discussion. It is highlighted that it is important to understand the purpose for the organization that is releasing products. There can exist different final goals with CD, such as improved quality or more frequent releases. These two end-states will require different optimizations points in the CI machinery and must consequently be understood from the beginning. OrgC´s future goal is to achieve GA releases every month. This is an end-goal, determining a frequency of how often different activities must be performed in relation to the goal. If not having this anchoring to the purpose and goal, the desired frequency of testing can only be speculated. A practical issue that OrgC had to manage when adopting CD was how the documentation should be introduced to customer. Several actions have been considered to improve this problem, such as an automated tool for release note and having them in an area where CU has access. It is further expressed that the customers also need to receive the notes in time before software preparation. A summary of the chapter regarding challenges at OrgC is summarized in Table 15.

Challenges	**Explanation**	**Solutions**
CI process	CI requires severe investments to automat testing	Dedicate resources
Test environment	Difficulty to automate in complex test environment and to separate product and environments fault	Software tool and visualize between pass, product fault or environment fault
Large scale development/main track	With more than 100 XFTs, the CI funnel is extremely difficult to manage	Dividing the product into modules that are pre-integrated and verified
Ever ready software (quality)	Cannot deploy nationwide with their CD drop. Need to avoid peeks in quality	On-going challenge. XFT need to feel responsible to keep quality, understand customer plans and be involved early.
Uneven flow of requirements	Peaks in requirements inflow and too much included in every release	Longer planning horizon with earlier in-depth knowledge about feature
Reliable updates	Large changes risking to interfere with legacy is no longer possible. Delivery of incomplete changes also needs to be managed	Small and backward compatible deliveries Software architecture that allows to turn off/on system improvement and feature
Deploying to multiple customers	Rapidly growing their CD customers created the challenge of managing several customer interface	Not expanding their CD collaborations. Automate KPI analysis
Lack of knowledge of CD (external)	Different external definition of CD and its benefits	On-going challenge
Internal resistance	Fear of deploying bugs to customers and damaging the quality stamp	External drive due to financial reasons of receiving functionality early
Lack of organizational consensus	Different conceptions of the purpose with both CI and CD	Understanding and communicating the end-goal (on-going)

Align PDU and product management	Different objectives and understandings of CD	Need to collaborate closer and XFTs need to be educated and motivated
Documentation	Improving/adapting delivery documentation in term of legacy and new functionality changes between releases	Continually updated and delivered to customers in time. Automation of release note and available for CU
Acceptance testing	Customers are still performing acceptance testing before deploying to live networks	On-going. Quality needs to be further improved.

Table 15, summary of challenges experienced at OrgC when adopting

5 Analysis

The analysis is structured according to the two research question stated in the introduction. In the first section the empirical data is analyzed according to previous found challenges when adopting CD. The second section focus on how the change towards CD can be implemented using the theoretical framework and by applying empirical data from the two organization adopted CD processes.

5.1 Challenges Adopting CD

In the empirical chapter, different barriers were identified from the three organizations and they are summarized in Table 16, according to the categories created in the literature review. During the investigation the close link between CI and automated testing became evident. Hence, we have redefined and merged the two different categories to one common topic in the analysis. Since limited challenges were found related to changing responsibilities that category was eliminated. Additionally, there were challenges regarding the deployment process that was not of technical nature, as identified in the theoretical reviews. As a consequence, the analysis now also includes practical issues regarding that category.

The main perspective of the analysis will be directed toward the perceived challenges identified at OrgA. However, since both OrgB and OrgC have introduced CD, these findings will be used in validating purposes, as well as complementing with barriers not realized at OrgA.

Challenges	OrgA	OrgB	OrgC
Network configurations			
1. Legacy product	X	X	
2. Diverse customer configuration	X	X	
3. Deploying to multiple customers			X
CI/automated testing			
4. CI process	X	X	X
5. Ever-ready software (quality)	X	X	X
6. Automated test on mature product	X	X	
7. Test environment	X	X	X
8. Manual/slow builds	X	X	
9. Regression feedback time	X	X	
10. Manual aspect in testing	X	X	
11. Single software track	X	X	X
12. Small frequent deliveries	X	X	X
13. Long feature lead time	X	X	
14. Code dependencies	X		
Deployment			
Technical challenges			
15. Network impact	X	X	X
16. Backward compatible	X	X	X
17. Reliable upgrades	X	X	X
18. Fast detection and correction		X	
19. Delivery of incomplete changes		X	X
20. High risk changes		X	X
Practical challenges			
21. Documentation		X	X
22. Uneven flow of requirements	X	X	X
23. Support function		X	
24. Breaking down features	X		
Company commitment			
25. Lack of management support	X		
26. Site development/large scale	X	X	X
27. Internal resistance	X	X	X
28. Lack of organizational consensus	X		X
29. Unclear goals	X		
30. Culture change		X	
Customer resistance			

31. Lack of customer trust/ find lead user	X	X	X
32. Network downtime	X	X	X
33. Acceptance testing	X		X
34. Lack of CD knowledge			X
35. Product restriction (hardware focus)	X		

Table 16, challenges when adopting CD in the telecommunication industry

5.1.1 Network configuration

As described in the theoretical chapter, diverse network configurations constitute a major challenge adopting CD, especially highlighted within the telecommunication industry (Holmström et al., 2012). Our empirical study at OrgA confirms this finding, as all interviewees mention this as a barrier towards CD (challenge 2). This problem also exists at OrgB, but with a slightly different character. With one universe product, the diversity instead origins from the possibility to switch features on and off, and different hardware setups at customer sites. This problem was not as prominent with the more modern OrgC product. However, one aspect brought up at OrgC was the difficulty of managing multiple customer interfaces in their CD deliveries (challenge 3).

5.1.2 CI and automated testing

A precondition for CD is to have an established CI server up running, in order to assure that the software quality is constantly ready for deployment (Claps et al., 2015). This has however been described as major challenge for all involved organizations (challenge 4). At OrgB and OrgC, substantial investments and management support was required to overcome the barrier, elements that have been absent at OrgA. Challenge 5 describes the difficulty in achieving the software in an ever-ready state of deployment. At OrgA, the quality of the LVS is fluctuating and it is only deployable every six months. An interesting finding is the negative spiral that has been established, due to the long time of correcting TRs on the GA branch. In turn, this leads to decreasing quality, as resources are divided between branches. This problem was also experienced within OrgB before their transformation in 2010, were several months of bug fixing on the software track was required before it could be deployed.

The main constraint for OrgA is that the automation of tests is not on the required level to support CI. A difficulty is to build automated testing on a mature product, with highly differentiated customer networks (challenges 6), also mentioned as a barrier by Lepännen et al. (2015). Another bottleneck for test automatization is that test benches hinder progress and that there is a scarcity in test equipment. Besides from the technical challenges, the required effort to increase test coverage has been underestimated and the goal regarding test automatization has been down-prioritized, unclear and vaguely defined (challenge 29). One barrier relating to automated testing not perceived at OrgA is the automation of verdict and evaluation for tests. In fact, at OrgA, verdict and evaluation is not even considered to be part of the automatisation process. This will therefore present a major challenge for them in future when moving towards CD.

In the theoretical chapter it is described that the legacy system can generate an overload of work for developers, which shifts the responsibility of testing to quality

assurance personnel (Leppänen et al., 2015). This is similar to how the processes are designed at OrgA. With limited automated testing, manual testing is managed by a test organization (challenge 10). The results from this manual testing are also less visual. As a consequence, this reduces the speed of the internal feedback loop and decreases ownership among developers. Moreover, the regression feedback time (challenge 9) from SAT and XSAT does not allow more frequent deliveries on the CI server. Instead, several sources have confirmed that XFTs are developing in average a week on their local LLVs, before merging into the main track. This conflicts with the argument of Rissanen and Münch (2015), that to achieve CI, a requirement is more rapid deliveries and reduced amount of code branches. The challenge for OgaA is consequently to ensure faster gate sweeps on the initial tests, allowing more frequent deliveries to the CI server (challenge 9). Another problem is the low reliability of some test result, due to unstable test environment (challenges 7). In combination with many commits per test run, it creates low ownership of fixing the errors, as well as manual analysis. The complex test environment also makes it time consuming and difficult to write test cases, which has increased the feature lead time. In addition to the manual aspect in testing, part of the build process still is manually handled (challenge 8), which needs to be fully automated to speed up feedback time.

Challenge 11 is related to having a software main track where feature content can continually be added. For OrgA, this is challenging due to the long regression time and complex test environment previously described. Another barrier to adopting a main track, experienced at OrgC, is the amount of deliveries and test machinery focused towards one track. With many external deliveries, it presents the problem of identifying which delivery that caused the failure. This is a problem of scale, with more than 100 XFT and 10 different development sites. OrgA on the other hand, is dependent on less teams and 3 development site, and to maintain a single track is expected to be less challenging in their smaller organization. Moreover, when adopting a single software track, a strategy for how to handle code dependencies is an absolute requirement (challenges 14).

One challenge indirectly related to CI is long feature lead times (challange 13). Before their agile/lean transformation, OrgB experienced long lead times due to handovers and a function based development. Applying XFTs could remove handovers and inject complete responsibility of a feature, which resulted in a 50% drop in development time. However, the same problem still exists at OrgA, even as the same initiative has been implemented. The explanation is lack of test automation competence and a complex test environment, making it difficult and time consuming to write test cases. In addition, since OrgA is developing large features, the resistance toward using MMF can also be seen as a cause.

5.1.3 Deployment

The deployment is an extremely critical process in the telecommunication industry, since updating a node is causing network downtime (challenge 15). An important concept within CD is seamless updates. It means that features are deployed without hindering the customer usage, which is extremely complicated and will require severe investments (Claps et al., 2015). However, due to the procedures when updating a node, seamless updates has not been applied in any of the three

organizations. Therefore, until network downtime can be fully eliminated, it is about exploring ways to reduce the downtime to an acceptable level for the customer and achieving fast and reliable updates (challenge 17). The impact on the network performance is however higher for OrgA than for OrgB and OrgC, because of its position in the network, and consequently it will constitute a larger challenge for this organization.

In addition to the network impact, deploying more frequently requires better quality and approvement from the customer. Challenge 17 (reliable updates) is related to these two aspects, as adopting CD requires a reduced internal verification process from the customer side. It will however be difficult to motivate for OrgA, due to the recurrent history of quality related problems during software introduction. According to Claps et al. (2015) one barrier to CD is preventing that the increased frequency of deployment leads to more software bugs. In relation to this, challenge 18 describes a challenge experienced both at OrgB and OrgC, namely finding an efficient way to quickly detect faults when introducing new functionality and being able to correct the error or rollback the version. Three important challenges, in order to not allow frequent deployment leading to more bugs are 16, 19 and 21. Changes need to be backward compatible in order to not cause problems for external systems (16). A software architecture that handles delivery of incomplete features is needed, since CD deliveries are snapshots of the latest version of the main track (20). High risk changes also have to be managed, as it would no longer be possible to commit these long before deployment (21).

One problem area for OrgB and OrgC is the difficulty to achieve an even flow of features (challenge 22), as they have two main releases dictating the inflow of requirements. This problem is highly relevant in terms of lean software development, arguing for an even production flow to reduce waste. An additional aspect of the problem, when adopting CD, is in terms of value creation for the customer. For the customers not to lose incentives to frequently update their network, value adding functionality must be incorporated in each CD delivery. This is however expressed to be challenging, due to the structure of the deployment process and path dependencies in behavior. Even if OrgA has not implemented CD, the same kind of behavior exists, where there is peak of work around the main releases. The phenomenon also creates a low transparency of quality, as several integrations are focused toward the LFD. Today, OrgA has large features with long lead time, which present the challenge in breaking down features (challenge 24), similar with the concept of small batch production (Claps et al., 2015). This requires alignment between PDU and product management regarding this issue, and in OrgA it is complicated by the fact that they are divided in two separate units. The methodology of breaking down features in terms of deployment is not performed at OrgC, but in OrgB it is part of an extension of the current CD processes in order to direct the development. The final practical issue is that the support function needs to be adopted to frequent deployment (challenge 23).

5.1.4 Company commitment

There are not only technical challenges when adopting CD, some are social in nature. For example, Leppänen et al. (2015) argues that top management support is

a prerequisite for success when adopting CD. In line with the previous, challenge 25 states that support from management has been lacking and strongly inhibited the implementation of CI at OrgA. It is expressed that enforcing the changes necessary has been especially challenging, as the transformation requires investment and a clear priority in the organization. The goal regarding the CI implementation has also been vaguely defined and under communicated in the development unit (challenge 29). In contrast, OrgB and OrgC both experienced strong support from management, which made it possible to make severe investment to support the transformation.

Another organizational challenge (26), experienced in all organizations, was the diverse locations of development sites. This is challenging due to several reasons. It becomes more complex to align goals and overcome cultural differences. With more frequent deliveries to the main branch an intensified communication is also a needed, both across teams and sites. A relating challenge (28) expressed, is the difficulty in establishing consensus within the organization and therefore strives towards a common vision. This develops into an even more important issue, as there is currently internal resistance (challenge 27) towards more frequent releases from CU and product management at OrgA. These two units does not see the value of deploying more frequently, which complicates the implementation, as they are the face towards the customer. The final challenge in relation to the category, expressed at OrgB, is the difficulty in changing the culture to create a mentality of joint ownership. At OrgA this has also been lacking, especially in relation to automatization.

5.1.5 Customer resistance

As defined in the literature review, adopting CD will often be faced with resistance from the customer. The customer might not desire new features, since frequent deployments can mean more bugs or it can interrupt their daily operations. Resistance from customers is a highly relevant finding in the empirical study and the challenges 31-35, are directly connected to this issue. Due to a history of unreliable quality, the customer does not trust OrgA's releases as it often is service impacting. As a consequence, they are currently avoiding adopting new releases if possible. In addition, there is no way to update a node without causing an activation downtime, which is another explanation why customers are resistant to more releases. A challenge identified at OrgC, is the lacking customer knowledge of CD. After adopting CD, they have experience various definitions of CD and expectations on the concept as a universal solution to all problems. Lastly, a specific challenge for OrgA is the nature of their product. With less functionality released and more hardware priority in regards to sale, it is expected to be difficult motivating the change for lead users. In comparison, the other organization releases considerable more functionality and are less depended on hardware. Hence, product restriction will present a challenge for OrgA and to convince customer may be difficult.

5.2 Developing practices to mitigate an adoption of CD

Based on the challenges found in the previous sub-chapter, findings from the empirical part as well as theory are analysed and used to provide suggestions on how to successfully adopt CD.

5.2.1 Organizational anchoring

The first step that will be discussed comprises the organizational aspects around CD and for this analysis, the theoretical section of change management will be relevant. As identified, there are several challenges of social nature that are vital to manage when adopting CD.

- Lack of management support for CI
- Unclear and insufficiently communicated goals in relation to CI and automation
- Site development
- Internal resistance and lack of consensus in the organization
- Uneven flow of requirements
- Requirement of new internal culture of ownership

In the theoretical chapter, five important steps for organizational change were identified. These will guide the following analysis and describe how the change toward CD can be designed, as well as how the challenges can be mitigated.

Define a vision

According to Blanchard (2010), a change should be initiated by defining the initiative and create a vision. This will also create the possibility to create internal consensus regarding CD, a problem OrgC is struggling with today. A logical suggestion is to manage these issues before starting to implement CD. Relevant for these insights, Wood (2016) suggests that before adopting CD, the developing company needs to evaluate if the customers can and are willing to handle an ever-changing product. If the answer is no, then CD may not be the ultimate solution. An alternative strategy is to allow a more continuous stream for early adopters, but at the same time continue to offer the more traditional approach for those customers not willing to change. These issues are especially important to explore before the change, as network downtime and quality related issues is expected to inhibit customer's willingness to adopt CD.

However, in the empirical findings it is revealed that customers desire less painful software upgrades and an improved software quality. If seamless upgrades were the case, it is even described that most customers would always update to the newest software version. These aspects have been improved at OrgB and OrgC by implementing CI and CD. The quality issue is related to CI and will be further discussed in 5.2.2. Adopting CD has also increased the speed and predictability of software rollouts, basically by moving maintenance work to before GA through utilization of the external feedback loop. Faster feedback from live environment to developers further allows for a constant high quality. These are consequently strong reasons why a change towards CD also with benefit the customer at OrgA. There are also other potential setups that can be realized by CD, such as increasing the number of GA releases per year (OrgC) and making CD deliveries available in the whole network (OrgB).

For OrgA there is however no logical reason, at the moment, to increase GA releases or make CD drops deployable nation-wide, since customers do not demand frequent deliveries of new features. In Table 17, the benefits with CD from the customer perspective is included, as well as the expected value for customers of OrgA if adopting CD.

Benefits for customer	OrgA
Earlier access to functionality	No
Higher quality at GA	Yes
Faster rollout	Yes
More frequent GA releases	No
Closer collaboration with suppliers	Yes

Table 17, summary of the potential benefits for customers and the expected preferences for OrgAs customer

Based on the previous section, the vision with CD at OrgA could reasonably be to assure quality to maintain a constant deployable level. This stretches as far as to set up the processes to continually utilize external test settings before GA, similar to the two other organizations' current processes. It does not involve more frequent GA releases or nation-wide deployment. In addition, those customers that not are interested in CD can continue with the traditional approach. If this setup is realized, the benefits is expected to match the customer needs, as higher quality is a precondition and it will increase the pace and predictability of software rollouts. A simplified, but possible CD setup is summarized in Figure 8.

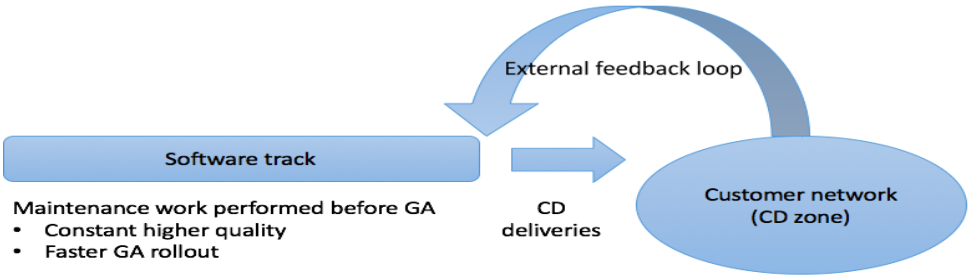


Figure 8, potential CD setup for OrgA

When the vision is defined it needs to be complemented with an explanation of the motivation behind the change and the expected benefits, as well as the consequences for its stakeholders. A general summary of the motivation, expected benefits and what it is required by its stakeholders to implement CD is visualized in Table 18. An understanding of these issues is needed for all parties to commit to the initiative and to know what is required of them. This is something that has been underachieved at OrgA, since the clarity of goals related to CI and test automatisation has been insufficient and lacked commitment. As suggested by Holmström et al. (2012) is that product management is committed to change, as they are the interface between the company and the customers. These are the once that

will convince customer of the value and are consequently vital in the transformation to CD.

Vision	Constant deployable software quality
Motivation behind change	Low and fluctuating quality Customers rarely update to new versions
Expected benefits	**Internal** Improved quality Faster payment **Customers** Improved quality Faster and more reliable rollouts Closer collaboration with supplier
PDU requirement	Closer collaboration with PL Commitment from management CI implementation
XFT requirement	Commitment to quality CI implementation
PL requirement	Closer collaboration with PDU Commitment from management Allocate the required funds Convince CD-pilot
Customers	Long term commitment Reduce verification process

Table 18, stakeholder analysis

Management support and leadership

Blanchard (2010) argues that quick results must be achieved and communicated when implementing a change. This could be an explanation to the low commitment in OrgA, basically that results have not been achieved. Furthermore, as employees experience goals as vaguely defined, it will be problematic to communicate actual and meaningful improvements. The empirical part also revealed issues around KPIs that needs to be handled in order for them to be effective. Today, several KPIs exist, but some of them are not realized or avoided from the XFT, which undermines their purpose. A suggestion, as described by Ashfaque Ur Rahman et al. (2015) is to break down the vision into key performance metrics that are accepted, understood and measured across the whole development unit (not separating test and design etc). In this way, the organization will strive toward the same defined vision, which is highlighted as a crucial aspect at OrgC. It may also reduce finger pointing, when communicating performance across teams (Ashfaque Ur Rahman et al., 2015). One important part for OrgA is consequently to review the communication flow in regards to the KPIs, making sure information is not lost between hierarchical levels. Another critical aspect to commit the XFT, is to immediately visualise progress and gain momentum in the transformation (Blanchard, 2010). This is something that both has been highlighted in the empirical study and literature review to motivate.

Structures and systems
Several scholars argue that change necessitates a separation from the past, and that structures and system need to change to support the new way of working. Mentioned by Lepännen et al. (2105), the developers need to have knowledge of CDs and the developing practices that comes with it. In this case, the XFT needs to be motivated and committed to keep constant quality on the LSV. As described at OrgB this requires a shift in culture were everyone is responsible. However, in OrgA there exists blockers that needs to be solved before this culture can be realized. First of all, an overwhelming amount of TRs has created a broken window symptom that is inhibiting commitment. This problem can only be solved by PL and PDU management, by allowing more man-hours dedicated to increase quality, rather than developing new functionality. The unstable test system and several commits per builds is also inhibiting a sense of ownership from the individual designer. These more technical issues that needs to mitigated when changing development practices is described in 5.2.2.

One challenge identified in OrgB and OrgC is that the two main releases are dictating the inflow of requirements. It is described that levelling of processes and changing behaviour is difficult and neither of the organization has achieved an even flow. In the literature chapter it is described that to separate from past behaviour, adjustment in staffing, training, communication and appraisal and reward might be necessary (Kotter, 2007; Mento et al., 2002 and Blanchard, 2010). These are consequently aspects that can be implemented in CD and not only to establishing an even flow. A suggestion is to adapt requirements definitions and institute key metrics accordance to CD. This needs to be further supported with education and training to create awareness in PDU. An important factor relating to education is to train the developers in appropriated developing methods and a clear example of this is unit test. Unit test is an approach rooted in test-driven development and described in the theoretical chapter as an important concept for CI and CD. However, due to the legacy debt and a knowledge gap, this has not been implemented at OrgA. When rewriting some of the product, instituting unit test as a standard practice would therefore be beneficial. This needs to be supported with the appropriate training sessions in order to developing competences. An additional strategy to widening of competences in regards to these test-driven methodologies is strategic recruitments, as well as the use of external consultants. Finally, a learning from OrgB was to use external incentives to achieve a change in behaviour. They were able to find a pilot customer which created a sense of urgency to solve some of their technical issues and enforced the change.

Gradual adoption
Emphasized by Reger et al. (1994) a change initiative usually needs to be broken down in smaller achievable steps, avoiding too big changes. In line with this, OrgB implemented CI in several steps. Moreover, it was highlighted that a premature adoption of CD can have negative consequences. For example, increasing the already existing resistance from customers and damaging the company brand. Avoiding too large change initiatives through stepwise adoption can also reduce the

internal resistance to CD (Reger et al., 1994). Hence, a gradual transformation towards CD is suggested for OrgA. First it is important to understand the purpose and vision, while preparing the organization to implement the change, as discussed in these first two sections. Secondly, CI and automation needs to reach the desired state that will be discussed in the following section. The final step is preparing for actual external CD testing, which is presented in 5.2.3.

Summary

In summary, OrgA needs to define the vision with CD, break down and express what is required of all involved parties, actively drive the change with the rights leaders and truly prioritize the change on management level. The vision needs to be decomposed into key metrics so that result and progress can be visualized and communicated in the development. These metrics needs to be understood and accepted in the XFT to gain commitment. There also exist structures that needs to be adapted to CD, such as behaviour in relation to quality and the inflow of requirements. To overcome these path dependencies relevant education and trainings are required. Finally, the change needs to be designed in a stage-wise process, avoiding too big changes and resistance. These stages are presented in Table 19.

Theory	**Suggestion**
Create a vision	Constant deployable software Create an internal consensus of the change and of what is expected of each party to achieve the vision
Management support	Management must be involved in the change and mitigate site development Allocate resources to the change Select leaders to drive the change internally
Measure and share progress	Break down the vision into key performance metrics that are accepted, understood and measured across the whole development unit Visual progress to create ownership
Adapt structure and systems	Define requirements to match CD Establish new culture of joint ownership Educate and train according to the new developing practice External incentives can be used to change behaviour
Gradual adoption	Stepwise adoption of CD • Organizational support (Step 1) • Implementation of CI (Step 2) • Preparation for CD (Step 3)

Table 19, applying change mangement in CD

5.2.2 Implementation of Continuous Integration

This analysis section will focus on how OrgA can successfully implement CI. The challenges 1-14 is related to technical issues with CI, but the implementation has also been inhibited by a lack of management support and unclear goals (challenges

25 and 29). Three main areas have been identified as initiatives required to be implemented before adopting CD, namely:

- Prioritization and support
- CI architecture
- Commitment to quality

In Figure 9 the steps are illustrated in an hierarchical aspect, by highlighting the responsible departments. As explained, management from both PDU and product management must be responsible to extricate resources to support CI and prioritize it within the whole development unit. This has also been emphasized in previous research, as Claps et al. (2015) and Lepännen et al. (2015) argue for a clear implementation strategy from top management. When it comes to the actual design of CI, it must be carried out by whole PDU organization to achieve viable result. At the final stage, the XFT and individual designers must be educated and committed to ensure a constant quality, in order to utilize the more efficient system. The arrows illustrate the iterative process between all layers, in which management must support all initiatives, and the XFTs will be a big part in implementing CI.

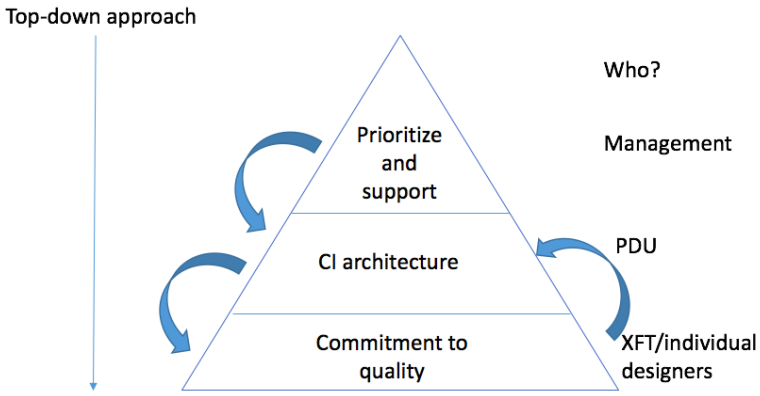


Figure 9, the required steps to implement CI

Prioritize and support

The first initiative to implement CI is based on the previous step, stating that the whole organization must commit to the transformation. It is described that most of the challenges are relatively uncomplicated to solve technically, but difficult to implement as it depends on different people and departments, who may not share the same goals. Hence, this stage is crucial in order to provide the right condition for a successful implementation. The key point is that OrgA has to be prepared to push for a change, to prioritize CI over other activities and to invest enough time and resources for a successful transformation. In addition, even if it has been proved possible to implement CI without an increased budget from product management, support from that department will definitely make the transition easier.

CI architecture

With CI prioritized within the organization, the actual work to implement the appropriated practices and architecture for CI can actually start. This part of the analysis has been divided in two parts, were the first will focus on the challenges related to diverse network configurations, the initial part of the design process and the problematic issues of developing on side tracks. More concretely challenges such as:

- Diverse network configurations
- Automated testing on mature product
- Long/manual build time
- Complex test environment
- Establish a software main track

The complex legacy and diversity in network setups and customer configurations are challenges without quick and easy solutions. OrgA has started on a new generation of the product, aiming for a more standardized hardware setup in the customer network. This could help improve the automated test coverage, as the configuration variation will be limited. The overall test strategy for OrgB has been to focus on commonalities in the different networks and thoroughly test these aspects. In combination with the new generation offering limited configurations, OrgA could use this approach as well, working with automation of the most common cases and fast detection and correction of the rest.

A possibility with the new generation is for individual designers to start using unit tests. Unit tests are one of the most important parts of CI and later CD, as they are fast to write and execute. Hence, it constitutes a great way of finding faults at an early stage (Rathod and Surve 2015 and Farcic, 2014). Using this method with faster feedback on new code will increase the quality before integration. The use of unit test has previously been inhibited by a legacy debt and consequently something that to some extent can be avoided during the development of the new generation. As with all new practices, this needs to be supported with the appropriated education and training of designer. Another suggested method for improving software quality is code reviewing, highlighted during the empirical findings and theoretical review. Code review is not only a way of detecting defects, but also to share knowledge between team members and realize alternative solutions (Baccheli and Bird, 2013). Ashfaque Ur Rahman et al. (2015) has found that using code review is a commonality for companies that has adopted CD and can be done both by manual analysis and automated software tools.

A concern for OrgA is that the build automation is not on the level to support CI and the integration rate to the LSV is too low. In order to move towards CI, the build time must be faster and fully automated, ensuring faster feedback and enabling frequent deliveries to the LSV. OrgB experienced the same problem with slow and unreliable builds and several solutions for a faster build process was implemented. However, the main reason to the decreased time to fix faults and failing builds was due to the increased visibility, creating a sense of ownership among designers. As described in the empirical study, one challenge is to develop a master script for the build process,

which is currently developed for the new product. This will allow for an automated process and enable OrgA more frequent build. However, the technical solution needs to be supported by commitment from designers, something that will be further discussed later in this section.

One condition for a successful CI process has been a software main track where features continuously can be added. Today, OrgA does not utilize a single track, as code is developed on team branches before integration. It is suggested that the new product will enable the possibility to eliminate these branches and integrate directly to the LSV. However, to manage this transition several initiatives are required, such as the mentioned suggestion to reduce configurations, unit test and a faster and automated build process. In addition, frequent and small deliveries is required to avoid large integrations of code, which are likely to break the system. The need for backward compatible changes is also highlighted in all organizations.

To support a main track, OrgA has to reduce the regression feedback time from the integration tests (SAT and XSAT). These runs include commits from several teams, which creates low ownership among developers to fix errors. It is suggested that the new product will allow to increase the speed and coverage required of the “gate sweeps”. In addition, as some test cases are unstable, the results often require manual analysis to decide whether it is a product fault or if it relates to the test environment. The complex test environment also making it time consuming and difficult to write the automated tests. Similar problems related to the environment was experienced at OrgB, when automating their node level integration test. To solve the problem, improvements to the IT environment, test environment, test tools and the test suit were made in order to increase the reliability of these tests. Moreover, a separation was made between product and environment faults, where the test issues now are handled by a dedicated team. This is similar to how OrgC has designed its processes, as this organization has automated the process of determining whether it is a test case problem or a product fault. These setups also allow the designer to ignore test issues and focus on fixing errors concerning the product. For OrgA it is consequently important to invest in improving the reliability of the test systems and to avoid false verdict. In combination with the condition the new product can enable, this will increase the incentive for designer to frequently check in their changes to the LSV and receive fast feedback on their code.

To further achieve high quality on the main track, OrgB has implemented a main track management team. They have the responsibility for the quality on the main track, and the authority to order design stops and to remove problematic builds. A potential trade-off with such a solution is however that ordering a design stop is extremely costly, especially for OrgC with large development organization. In addition, maintaining a main track can be extremely challenging in a large scale development organization as illustrated at OrgC, where it necessitated a separation of the software main track into modules. This is valid critique against implementing a main track and having team managing it. However, OrgA delivers markedly less functionality and the developing activities are not on the same scale, so this solution seems to be a suitable strategy for improving the low quality.

The previous section has to a great deal focused on different initiative in order to introduced a single software track. OrgB has showed that a main track can be used to reduce the time working with the quality on the release branch. This way more focus can be put towards the one branch, which leads to less fluctuating quality, less TR mapping, increased content and a better start for the work with the next release. The following section will continue to discuss how the test automation can be further improved when the previous mentioned condition with a main track is established. This part of the analyse is related to challenge such:

- Fluctuating quality (not ever-ready)
- Automated testing on a mature product
- Manual aspect in testing
- Verdict and evaluation

In order to reach an ever-ready state of deployment, OrgB has implemented several solutions over a long period of time. It has meant efforts focused on automating tests on all levels. According to Rissanen and Münch (2015), automated tests and managing their environments are the toughest challenges for developers, which is also the case for OrgA and automation of tests are not on the level to support CI.

To further address constant quality, OrgA needs to automate all test activities, both in terms of functional and non-functional tests, and visualize the results to improve time to feedback. These kinds of tests are today to a great extent performed manually at and not visual on the monitors, which is increasing the time to fix TRs. When these test are automated, the frequency of the tests can drastically increase, as illustrated by both OrgB and OrgC. Instead of performing these tests every six month, automation made it possible to run them on a weekly basis and drastically decrease time to feedback for designers. However, to realize an CI machinery with the required test coverage, investment in test equipment and facilities is a fundamental requirement. This will enable the frequency of test, which was highlighted in the empirical study as the most important part in the automation. With an increased frequency of tests, less new commits are included in each test, less faults found and which delivery that was causing a failure will be more visual. The manual work and the repeated execution can consequently be reduced and the efficiency will increase drastically. It is described to be a positive cycle were higher frequency leads to better quality, which further leads to an increased frequency. By making the results from these test visual on the monitors, the time to feedback could further be improved. The following test strategy can be considered to improve the likeliness of findings errors when dimensioning automation of test cases:

- Test cases that fail often should be executed often
- Test cases that fail infrequently should be executed infrequently
- Test cases that fail most often should be executed first in each test run
- If a group of test cases always fail or pass together: include only one test case per test run, and change it between runs

A further aspect for automation is evaluation and verdict in testing and this is an area that has proved difficult for OrgB and OrgC to solve. In OrgA the definition of automation does not even include evaluation and verdict. The first decision should reasonable to change that definition. To solve the barrier, OrgB is assessing machine learning and artificial intelligence as potential solutions for this challenge. For now, they have co-located test teams and cross functional test teams to reduce delays. As Leppänen et al. (2015) states, difficulties in automation of tests can be delegated to other developers or quality assurance personnel, with delays as a consequence. Until higher degree of automation is achieved within OrgA, co-locating test organization and XFTs for closer collaboration may be a solution to consider.

In conclusion, to improve automated testing, OrgA can implement several of the suggested solutions. First of all, a software main track is the fundamental condition for CI. Investments are required to increase the reliability of tests, and the separation between product and environment should to be automated and visualized to create ownership. The test environment needs to be simplified to ensure faster testing, both in terms of execution and writing. As the quality on the LSV is considered to be low, a team responsible to secure quality on the LSV is also an attractive solution. To further address a constant quality, OrgA must automate the whole test pipeline and achieve an increased frequency of tests, which evidently is the key to improve quality. Also in this stage a simplified visualization process of heavier test results would be beneficial for commitment.

Commitment to quality

When the change has been prioritized from management and an appropriate CI condition is implemented, the final stage is gaining commitment to quality from the XFT. Problematic issues inhibiting commitment and ownership on the operational level is:

- Low ownership among designers
- Automation has been driven by individual testers
- Culture rooted in functional software development
- Lacking competences regarding automated testing

There are several aspects inhibiting ownership among individual designers and the XFTs. The unpredictable test environment is creating an uncertainty in whether a reported error is an actual product fault or a false verdict. Hence, as previously defined, a major priority must be to simplify the test environment and automatically separate between test issues and product faults. Today, it is expressed that designer actually avoid correction of TRs due to this problem, in combination with several commits per build. Immediate and fast correction can therefore induce a self-reinforcing situation. A relevant concept to eliminate this kind of behaviour is shepherding changes, meaning that the developers are responsible for the changes throughout the whole deployment process, as well as fixing faults after deployment. Several software companies have used a call policy, requiring any deployment problem to be fixed at the earliest time (Ashfaque Ur Rahman et al., 2015). A similar approach could be adopted to OrgA, by implementing a policy of immediately fixing

TRs on the LSV and make team responsible for parts of the product. Another contributor to this behaviour is the already existing TRs and that it is easier to leave faults unsolved when there are many of them. This problem can only be solved by management and the dedication of resources toward fixing TRs and it will be on the expense of developing new features. The visual separation of product and test faults could also promote responsibility toward faults in test code. Test code have not been allocated high priority and potentially one part in the unstable test system. By making them visual and deducting a team to these issues, as OrgB, could consequently be beneficial for the implementation of CI.

An initiative that to some degree has been implemented, is to drive automation in the whole development unit. It has previously been performed by testers in solitary, which is considered to be a remainder of the historical functional software development, separating design, system and test. This can be seen as low agile maturity that needs to be moderated into an a cultural of taking joint ownership. The problem is further complicated by the shortage of testing competences at OrgA. When OrgB started their transformation, massive efforts were directed towards designing a decentralized organization with complete XFTs. It required several years of relocating employees and programs for competence widening. In order to improve the agile practices, one suggestion is also to use a scrum master to coach the teams. A possible resistor to the change could however be the test organization that neither trust nor is accustomed to the way of working. This is personnel can and must be leveraged to improve automation. The lack of automated competence consequently has no immediate solution, but requires resources devoted to education and competence widening.

In summary, the last puzzle piece in the CI machinery is commitment from the XFT. These are the ones actually developing the software and needs to feel motivated to maintain the quality. There are several technical issues that is inhibiting ownership such as test environment and many commits per build. If these issue are solved it is suggested to implement a call policy to immediately fix TRs. CI also requires a modification in corporate culture were automation is a mutual responsibility. There is also indication that there is lacking competences in term automating testing that can be solved by rotation programs and leverage manual testers. An additional strategy is to use scrum coaches to improve the agile maturity within the XFT.

The areas analysed throughout the chapter are summarized in Table 20 where the different initiatives and their purposes are described.

Areas	**Initiative**	**Purpose**
Prioritize and support	Management commitment to prioritize CI Investment in test equipment	Creating the opportunity for CI

CI (stage 1)	Utilize the benefits with new product Unit test and code review Automate and visualize builds Reduce side tracks Simplify test environment Main track management	Restrict network configurations to enable test coverage Improve quality before integration Improve frequency of builds and integration rate to LSV Focus test capacity Easier to write tests Faster to execute (minimal time to feedback) More frequent integration to LSV
CI (stage 2)	Automate all testing/eliminate manual testing Improve visualization Test often to reduce number of commits	Keeping constant quality on the LSV Visualize results from heavier testing
Commitment to quality	Clear distinction between product and test faults Team dedicated to test code Competence widening Scrum coaching Call policy to commit to fixing errors	Better feedback to create ownership Creating commitment and competences to keeping high quality XFT can focus on product code

Table 20, the stages in implementing CI

5.2.3 Continuous Deployment

The empirical study reveal that starting to deliver functionality more frequently, first requires years of developing the CI process. This needs to function well in order to reduce the risk that the increased releases cause trouble for the customers and damage the supplier's trustworthiness. As many parts of the company are working towards CD with the same or similar customers, it is expressed that an organization handling CD poorly can harm the progress of others. With CI in place though, CD releases can offer important advantages, as seen from the benchmarked organizations. When CI and internal quality is addressed, a move towards CD is however associated with a number of challenges that needs to be overcome before an implementation:

- Minimal network impact and risks
- Fast detection and correction of errors
- Finding lead users and customer resistance
- Breaking down features and commitment from product management
- Even flow of requirements
- Documentation
- Support CD deliveries

As Rissanen and Münch (2015) emphasize, CD requires minimal network impact, which can be very difficult to accomplish. For the supplier, this means both bug free software and a seamless upgrade process. Interviewees within OrgA have been quick to state that seamless upgrades is a difficult concept in Telecom. However, it claims it to be possible through the use of double cores, although, it definitely requires resources directed to product development. Network downtime is also

caused if the new update consists of bugs, which stresses the importance of working with the quality of the software itself and is deemed at least as important for CD. As earlier stated, OrgB expressed that a sufficient coverage from automated tests combined with fast detection and correction of defects is the preferred approach. However, this requires a lead user that are willing to deal with the network downtime.

To ensure high quality software, a few important initiatives have been found in the empirical study. OrgC and OrgB have included the possibility to turn on and off deliveries. This corresponds well to the use of dark features, mentioned by several scholars in relation to CD (Claps et al., 2015 and Ashfaque Ur Rahman et al., 2015). CD releases has meant that incomplete changes are being delivered and a way of assuring that these does not negatively affect the system is important. Functionality deployed also have to be backwards compatible, which has been set as a requirement within OrgC and OrgB. This has mean new design rules for developers and tools to automatically detect breaks in compatibility. In general, legacy safe deployments are crucial, and high risk changes should be broken down into smaller ones, as is emphasized within OrgB. These quality assuring initiatives are deemed very important for OrgA, since the customers' trust in the quality is currently low and the impact in case of failures is disastrous. Ensuring quality and gaining customer trust is essential for OrgA, as many customers do not want to upgrade more than what is absolutely necessary with the current quality level.

Continuously deliveries to network clusters allows for many corrections to be made before GA, by the utilization of the external feedback loop. OrgC has a team dedicated to collecting data from these networks during upgrades. This solution involves measurement and analysis of KPIs in live networks at each CD customer site. In OrgB, an automated tool is used for faster and more efficient monitoring, overseen by a centralized function that is working closely to PDU. Both of these setups has led to a reduction of GA upgrade time, however the automated tool is making the work regarding monitoring considerably less. A modified version of the automatic tool used at OrgB could consequently be inherited to effectively manage the feedback loop. Moreover, to develop the ability to fast fast correct the identified errors, a suggested method is to allow the central function provide feedback to the right developer. This process needs to be efficient in order to include correction in the next CD delivery. An interesting improvement enabled by the previously mentioned tool is also to allow the XFT to monitor how the features are being used. This will provide better feature understanding and the development can be modified according to that.

In order to adopt CD, the customers also need to buy in on the concept, which is a major barrier for OrgA today. In addition, as illustrated both in previous research (Holmström et al., 2012) and the empirical findings, product management and PDU must work closer together when starting to adopt CD. Product management is the interface towards customer and consequently a critical part in convincing customers to see the value with this way of working. As previously discussed, slow and unreliable upgrades mean large costs for the customer and are preferably avoided. By learning of the upsides from the other two organizations, product management

can communicate its advantage by emphasizing on the results of faster and more predictable updates. This is the considered the main selling point, as early access to new functionality is not deemed to be as interesting as it is in OrgC. It will however require customers that are willing to take risks in parts of their networks, and allow time for identifying root causes to potential issues. It also necessitates customers with fast verification processes, which can pose a challenge for OrgA because of the current trust issues. Although, OrgB and OrgC have focused CD towards smaller but representative parts of customer network. Moreover, Woods (2016) suggest that at an early stage CD can be applied in a less critical part of their network. When the previous stage of CI is on the required level, a CD setup for a small and less critical part of a customer network could be a good next step for OrgA in order to explore the external feedback loop.

An additional argument for involving product management in this stage is the possibility to break down features and receive faster feedback from live environment. They are responsible for the backlog, but today at OrgA, prioritizing functional parts of a feature is only performed during critical time constraints. An explanation to this behaviour is that it will require an in-depth analysis and that there exist some resistance to this way of working. Furthermore, critic against CD in general at OrgA is the hardware focus and the insufficient quality, which decrease the commercial incentive of selling software. Hence, it will require a change of mentality from product management to commit. Commitment is further required, since they initiate the activities performed at PDU and investments are required to support CD. In summary, product management has a crucial role in this work, both in terms of internal and external concerns. To overcome the resistance, PDU must first of all achieve a more constant internal quality with CI, then product management must understand the value of adopting CD. A possible way to mitigate this, inspired by OrgB and OrgC, is learning from other organizations who have already made the transition and who can provide tangible evidence of the positive effects.

One challenge that has appeared is to achieve an even flow of requirements, as previously discussed. As GA is still set every six months, it is claimed to foster certain behaviours. LFD is still seen as the deadline for deliveries, and deliveries are being concentrated to the time before this point. This has yet to be solved, however OrgC is aiming to prevent this phenomenon by working with the planning process. It is expected to help if improvements are made to planning prediction and a better initial analysis of the requirements. Although, more time in the planning phase of requirements is a step back regarding agile and lean, basically as it slows down the overall process. The root cause of this issue is the misfit between large GA packages and CD, but until this can be changed the way forward could be to work with the structures and systems that currently supports this faulty behaviour, and arrange it to promote a more even flow of requirements. OrgB expresses the need for creating awareness within PDU, as the problem is also a question of mind-set. It is apparent that OrgA’s customers are not pushing for faster releases, and GA will probably remain two times a year in the foreseeable future. A strategy for handling the flow of requirements in this situation will therefore be essential.

Customers also have to be provided with documentation before every CD release (Claps et al., 2015), which requires an adaption of release notes. These should preferable be automated in order to continually deliver to customers in time. In addition, OrgB has identified a need to adapt the support setup for unexpected network failures in the CD process, as unreleased software is currently not supported. It requires more availability of internal support personnel and a standby solution around the world. The same demands would apply to OrgA in an introduction of CD, as support is currently arranged around FOA and GA releases, and a customer request system with long lead times.

Summary

CD will require a well-functioning CI process and careful quality assurance. If possible, eliminating the activation related downtime would be beneficial, but as long as GA is not more often than twice a year, it is considered to be more important to reduce the quality related risks and uncertainties. Important concepts for quality assuring CD deliveries are dark launching, backwards compatible changes and break-down of high risk changes. Remaining issues or bugs that slips through the testing can be detected and solved more efficiently with the use of an automatic tool, similar to OrgB. The importance of including product management in this stage, is also highlighted as a mitigating action to convince customers and to gain further internal support. Appropriate customers must fulfil the requirements of a representative network and willingness to engage in a long-term partnership and deal with risk. Educating and communicating internally the need for an even flow of deliveries, and backing it up by supportive systems has also been identified as necessary for CD. Otherwise, it will difficult to include functionality that will increase customer value. Lastly, the documentation and support processes have to be adjusted to fit more frequent releases.

Preparation areas for CD	**Recommended solutions**
Minimal network impact and risks	Explore possibility to eliminate activation downtime Dark launching Backwards compatibility Breaking down high risk changes
Fast detection and correction of errors	Automated monitoring tool Manage external feedback flow to XFT
Commitment from product management	Interact with/convince customer Promote faster and more reliable upgrades Breaking down features Allocate investments to CD
Finding appropriate lead user	Representativeness in network Long-term commitment Risk awareness
Even flow of requirements	Institute supporting structures and systems Create awareness in whole organization
Continuous flow of documentation	Review and adjust process Automate release notes

Support setup	Adapted customer support to manage unreleased software

Table 21, preparation for implementing CD

6 Discussion

In this section, the results are further concretized and discussed from the case company's current situation. Answers to the research questions are clarified, and possible future research areas are pointed out.

6.1 Results

This study has explored which barriers that exist when adopting CD in the telecommunication industry. Furthermore, based on these challenges, it addresses how a developing company can successfully adopt CD. The discussion will be structured according to the research questions initially stated.

RQ1: What are the industry specific challenges when adopting CD in telecommunication?

To fulfil the initial part of the purpose, data was collected from three internal organizations and contextualized with the aid of the theoretical review conducted. Awareness of the barriers is important for companies in their efforts to apply CD in their software development processes. In the analysis 35 challenges were identified, as visualized in Table 16. These have been divided into five categories; network configuration, continuous integration, deployment, company commitment and customer resistance.

A transition to CD requires a CI process with the software in an ever-ready state of deployment. As the empirical study reveals, this implies several technical as well as organizational challenges. Some of them are more specific to the telecommunication industry. This industry involves diverse customer networks configurations, which makes it difficult to achieve sufficient automated test coverage. In combination with developing automated test on mature products, the diversity has strongly inhibited the implementation of CI. The situation is further complicated by the fact that software updates in this industry historically have constituted large and complex projects. The quality of the software has therefore fluctuating in accordance to these releases, and achieving a flow with consistently high quality is a challenge that must be managed.

Another severe barrier is the difficulty to achieve seamless updates, due to the setups of the networks. Each studied organization's software updates are consequently causing temporary activation downtime for the customers. To be able to successfully release software to customers more often, this downtime has to be eliminated. As a part of the upgrade process, the customers are performing acceptance testing. This also hinders an immediate deployment. In these settings, CD consequently requires a partnership between supplier and customer. However, it was reported challenging to find users willing to adapt to the CD concept. This is expected to be especially difficult for OrgA, with relatively low feature drive, the network setup and a history of quality related issues.

The mentioned technical barriers will require time and resources to solve. However, based on this study the critical challenges are deemed to be the organizational ones. Even if the technical issues attract the most attention in terms of barriers, the empirical study indicate that the technical barriers are manageable when effectively driven in the whole organization. The remainder of this chapter will focus in the second research question:

RQ2: How can a *telecommunication company overcome these challenges when adopting CD?*

Based on the challenges identified in relation to the first RQ, several mitigating actions are required before adopting CD. In the analysis, three main stages were realized that can be performed to improve the likeliness of successfully implementing CD, as visualized in Figure 10.

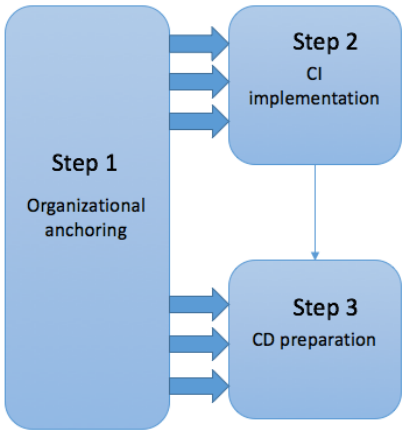


Figure 10, suggested stages before adopting CD

Step 1
To overcome the organizational challenges, change management has been used to analyze the data. In the analysis, five key actions were found to be relevant in order to mitigate an adoption of CD, as summarized in Table 19. These actions are by the authors considered to be the success factors for implementing CD and to solve the technical challenges that exist. It is not considered that the technical challenges are too severe, or that company has lacked the competencies to implement these changes, it has rather been de-prioritized in the organization. The organizational anchoring must be an on-going process, throughout the whole implementation process, as illustrated in Figure 10.

Step 2
In the analysis, it was realized that implementing CI is dependent on activities in different parts of the organization, as visualized in Figure 9. First of all, a top down strategy from management is required to establish the following:

- Priority in the development unit
- Investment in test/IT equipment

- Man-hours dedicated to CI and automatisation

When the support is in place, the development unit will have the required resources to successfully implement CI. As identified in the analysis there are some critical technical aspects in getting CI in place, including a main track and automating the test pipeline. For reducing the number of code branches, several of the previous mentioned challenges needs to be solved. This includes to automate the build process, to simplify the test environment and to always deliver backward compatible changes. To improve quality, unit test and code review is also suggested. The initiatives enabling a main track is summarized in Table 22.

Main track	**Purpose**
Unit test	Early detection of bugs Reflection before coding
Code Review	Sharing of knowledge within team Improve quality
Simplified test environment	Better test coverage in integration tests Faster execution time of integration tests Faster time to write automated tests
Automated build process	Frequent deliveries to the main track
Backward compatibility and practice of small deliveries	Legacy safe deliveries

Table 22, initiative that could enable a software main track

With a single software track, test capacity can be focused and feature content can continuously be added in smaller deliveries. To achieve the frequency of test activities required for a constant quality, all test cases must be fully automated. A critical aspect is making the result visual to faster correct potential TRs. A main track management team can also be leveraged to improve the feedback flow and to maintain constant quality by removing troublesome builds. To further avoid manual interaction, a separation between test and product code is required and teams dedicated to test code problem will be beneficial. These are actions that will aid the developing company to maintain a constant quality on the software track.

Automation	**Purpose**
Automate legacy	Reduce manual aspect in testing Increase efficiency
Make the result visual	Improve feedback loop Faster correction time
Automate separation of product and test	Avoid manual analysis XFT can focus on product faults
Main track management	XFT can focus on product faults Address constant quality

Table 23, initiative to improve automation and maintain constant quality

The final requirement in an efficient CI machinery is gaining commitment from the XFT and individual developers. In the analysis several initiatives were identified in relation to this issue. Competences in automation is required to support CI and programs dedicated to competence-widening can be utilized. Scrum coaches can also be a mitigating action to increase the agile maturity in the teams. To support a constant quality, TRs cannot exist and it is recommended to institute a call policy of immediately fixing faults, which requires clear responsibilities areas between teams. A higher priority of test code must also be highlighted to improve the stability of the test system. It can further be beneficial with teams directly responsible for these kind of problem, as previously mentioned. In overall to gain ownership on the operational level, education on why the change is required and the XFT need to realize the importance of constantly keeping quality high.

Step 3
The final stage before CD is about managing more frequent deployments and the related preparations. First of all, this stage should not be considered until CI and a constant high quality is achieved. When such a stage is reached, the following preparations can be considered.

One major barrier is the difficulty to achieve seamless updates in the telecommunication industry, which is a precondition for the theoretical definition of CD. It is suggested that a system with double cores is a solution for eliminating the activation downtime. However, this has not been applied in any of the included organizations and it requires further product development. Achieving this is by the authors considered to be highly desirable, due to several reasons. The updates will be able to take place during daytime and include more nodes per day. Hence, it will reduce costs and increase the speed of rollouts. It will also expand the incentives for customers to always update to the latest version, which means more frequent payments. However, until such a state is realized, the objective could be to reduce network downtime related to quality issues in the software. This means using small backward compatible changes, avoiding to affect external system, and dark launching to evade that half-finished features are interfering with legacy. Another crucial aspect is to develop the ability to fast detect and correct errors. This should be automated in order to avoid any manual aspects involving unnecessary work and resources.

In order to mitigate customer resistance to CD, it is crucial that product management gets involved at this stage. Adopting CD is dependent on the collaboration with customers and one possible selling point is faster and more predictable software rollouts and increased quality at GA. One could also promote a closer collaboration with the supplying company, something highly valued by customers who have adopted CD. In addition, allowing customers to only deploy in limited and less critical parts of their networks can increase the external incentive. However, there also need to be requirements, which appropriate CD customer must fulfil, such as:

- Representativeness in the network

- Long-term commitment
- Risk awareness

The support from product management is also required as they are responsible for the budgeting for PDU and CD will require investments. In addition, managing challenges such as an even flow of requirement and breaking down feature a modification of work processes is needed. There are some practical issues in relation to documentation and setting up support processes that needs to be defined before adopting CD.

6.2 Limitation and Future Research Areas

A finding in this study is that CD also means big adjustments for the customer when the suppliers is adopting CD. However, a limitation in the result is that it is purely from the supplier's perspective due to difficulties in gaining access to customers. By not including the external perspective the results does not describe the complete picture of the reality and what barriers the customers perceived during the transformation. Consequently, research that in detail investigates what CD requires of the customers would be valuable and is further pointed at by Claps et al. (2015).

Moreover, for a company adapted to CD, the next step is continuous experimentation. This treats how the external feedback loop can be leveraged to direct the R&D efforts (Holmström et al., 2012). In Holmström et al. (2012), this step was included in their conceptual model. However, no empirical findings could support the difficulties of implementing it, since no company has adopted the approach. Therefore, this seems to be the next logical step and something that future researchers could explore.

7 Conclusion

In the concluding remarks the main findings of the study are briefly summarized and the contribution to academia is highlighted.

The purpose of this master thesis was to contribute within the field of continuous deployment in software development. In more detail, it addresses how a developing company can overcome the barriers that actually hinders a successful implementation. The study has been performed within the context of the telecommunication industry, exposed to especially demanding conditions. Previous research has described that both technical and organizational challenges exist, but not adequately how the adoption of CD could be designed. This paper has been an attempt to fill that gap by including a case company and three internal organizations where CD partly has been adopted. The findings from the empirical part was contextualized with a theoretical framework, where literature within software development and change management was included. This approach allowed for a detailed investigation on how to solve both the technical and organizational issues. The findings are valuable for practitioners in a similar context, considering to implement CD.

References

Agile Alliance, Continuous Deployment. <http://guide.agilealliance.org/guide/cd.html>, 2012. (Accessed 17.02.16).

Ashfaque Ur Rahman, A., Helms, E., Williams, L. and Parnin, C. Synthesizing Continuous Deployment Practices Used in Software Development, Department of Computer Science, North Carolina State University, 2015

Arnold, S. Make Continuous Deployment Practical and Cost-effective with Rational ALM Tools.<http://www.ibm.com/developerworks/rational/library/continuous-deployment-rational-alm/index.html>, 2012 (accessed 17.02.16).

Bacchelli, A. and Bird, C. Expectations, outcomes and challenges of modern code review, In: Proceedings of the International Conference on Software Engineering, 2013

Berger, C. and Eklund, U. Expectations and Challenges from Scaling Agile in Mechatronics-Driven Companies – A Comparative Case Study, In Software Engineering, and Extreme Programming. Springer International Publishing Switzerland, 2015, pp. 15-26.

Berntsson Svensson, R., Debbiche, A. and Dienér, M. (2014). Challenges When Adopting Continuous Integration: A Case Study. In: *Product-Focused Software Process Improvement.* [online] Helsinki: Springer International Publishing Switzerland, 17-32.

Blanchard, K. Mastering the art of change. www.trainingjournal.com, 2010, pp. 44-47.

Bosch, J and Ståhl, D. Experienced Benefits of Continuous Integration in Industry Software Product Development: A Case Study. 2013.

Bosch, J and Ståhl, D. Modelling continuous integration practice differences in industry software development, In: The Journal of Systems and Software, 2014, pp. 48-59.

Bryman, A. and Bell, E. Business research methods. Oxford University Press, 2011.

Carcary, M. "The Research Audit Trial – Enhancing Trustworthiness in Qualitative Inquiry." *The Electronic Journal of Business Research Methods Volume 7 Issue 1 2009, (pp.11 - 24), available online at www.ejbrm.com*

Claps, G., Berntsson Svensson, R. and Aurum, A. On the journey to continuous deployment: Technical and social challenges along the way, In: Information and Software Technology 57, 2015, pp. 21-31.

De Cesare, S., Lycett, M., Macredie, R.D., Patel, C and Paul, R. Examining perceptions
of agility in software development practice, Commun. ACM 53 (2010), pp. 126-130.

Easterby-Smith, M., Thorpe, R. and Jackson, P. Management research. SAGE publications, 2012.

Farcic, V. Continuous Deployment: Strategies. <http://technologyconversations.com/2014/12/03/continuous-deployment-strategies/>. 2014. (Accessed 02.03.16).

Fowler, M. and Highsmith, J. The Agile Manifesto. Software Development Magazine, 2001.

Fowler, M. ContinuousDelivery <http://martinfowler.com/bliki/ContinuousDelivery.html>, 2013. (Accessed 17.02.16).

Fowler, M. Continuous integration <http://martinfowler.com/articles/continuousIntegration.html>, 2006. (Accessed 10.03.16).

Highsmith, J. and Cockburn, A. Agile Software Development: The Business of Innovation, In: Computer, Vol. 34, Issue 9. IEEE, 2001, pp. 120-127.

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

Koch, T. Establishing rigour in Qualitative Research: the decision trail. Journal of Advanced Nursing. 53, (1), 91-103. 2006

Kotter, J. Leading change: Why transformation efforts fail, Harvard Business Review, 2007.

Leppänen, M., Mäkinen, S., Pagels, M., Eloranta, V., Itkonen, J., Mäntylä, M. and Männistö, T. The Highways and Country Roads to Continuous Deployment. The IEE Computer Society, 2015, pp. 64-72.

Lewis, J., & Ritchie, J. (2003). Generalising from qualitative research. In J. Ritchie & J. Lewis (Eds.). Qualitative research practice: A guide for social science students and researchers (pp. 263–286). London: Sage.

Lianping, C. Continuous Delivery: Huge Benefits, but Challenges Too. The IEE Computer Society, 2015, pp. 50-54.

Mento, Anthony J., Jones, Raymond M., Dirndorfer, Walter. A change management process: Grounded in both theory and practice. Journal of Change Management, Vol. 3, No. 1, 2002, pp. 45-59.

Miller, A.: A hundred days of continuous integration. In: Agile Conference, AGILE 2008, pp. 289–293 (August 2008)

Olsson H.H., Alahyari H., Bosch, J. Climbing the ‘‘Stairway to Heaven’’ – a multiple-case study exploring barriers in the transition from agile development towards continuous deployment of software, in: Proceedings of the 38th EUROMICRO Conference on Software Engineering and Advanced Applications, 2012, pp. 392–399.

Mason, J. (2002). Qualitative researching (2nd ed.). London: Sage.

Poppendieck, M. and Poppendieck, T. Lean Software Development: An Agile Toolkit, Addison-Wesley Professional, 2003.

Rathod, N. and Surve, A. Test Orchestration: A framework for Continuous Integration and Continuous Deployment. International Conference on Pervasive Computing, 2015.

Reger, Rhonda K., Mullane, John V., Gustafson, Loren T. and DeMarie, Samuel M. 1994. Creating earthquakes to change organizational mindsets. Academy of Management Executive, Vol. 8, No. 4, 2010, pp. 31-46.

Ries, E. The Lean Startup: How Today's Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses, Crown Business, USA, 2011.

Rissanen, O. and Münch, J. Transitioning Towards Continuous Delivery in the B2B Domain: A Case Study, In: Agile Processes, In Software Engineering, and Extreme Programming. Springer International Publishing Switzerland, 2015, pp. 154-165.

Smeds, J., Nybom, K.,Porres, I. DevOps: A Definition and Perceived Adoption Impediments. C. Lassenius et al. (Eds.): XP 2015, LNBIP 212, pp. 166–177, 2015.

Southard, P. and Parente, D. A model for internal benchmarking: when and how? Emerald Group Publishing Limited, 2007.

Stake R.E. The Art of Case Study Research. SAGE publications, 1995.

Virmani, M. Understanding DevOps and bridging the gap from Continuous Integration to Continuous Delivery, in: Fifth international conference on Innovative Computing Technology, 2015, pp. 78–82.

Williams, L. What agile teams think of agile principles, Commun. ACM 55 (2012) 71-76

Wood, N. <https://www.madetech.com/blog/continuous-delivery-organisational-challenges> 2016 (accesed 24.02.16)

Yin, R.K. Case study research: design and methods, 3nd ed., Thousand Oaks: Sage, 2003, Applied Social Research Methods Series, Vol. 5.

Appendix

Interview guide Block A

General information
Information about non-disclosure and how the recording and findings will be used.
Describe the purpose of the study.

Q1 Background
Can you briefly describe your background at the case company?
What is your current role and what responsibilities do you have?

Q2 Current processes
Can you describe the process for product introduction?
How are new software features developed?
How are software releases deployed?

Q3 - Strength and Weaknesses
What are the main strengths with the way of working?
What are the main weaknesses with this way of working?
What would you like to change?
How are these issues being handled?

Q4 - Barriers to CD
Which barriers do you think exist for adopting CD?
What will it require to overcome these barriers?
What in the current way of working needs to be readjusted?

Open ended question
Is there anything you think we should further explore?
Is there anything you want to complement that has not been covered in these questions?

Interview guide Block B

General information
Information about non-disclosure and how the recording and findings will be used.
Describe the purpose of the study.

Q1 Background
Can you briefly describe your background at the case company?
What is your current role and what responsibilities do you have?

Q2 Current processes
Can you describe how you are using CD in software development and deployment?
How are software features developed?

Q3 - Strength and Weaknesses
What are the main strength with the way of working?
What are the main weaknesses with this way of working?
What would you like to change?

Q4 - Barriers to CD
What were the condition before starting to adopt CD?
Which barriers have existed during the implementation of CD?
How did you overcome these barriers?
What were the main practices that made CD possible?

Open ended question
Is there anything you want to complement that not been covered in the questions?
What advice would you give to an organization before implementing CD?