Extending the development process towards Continuous Delivery and Continuous Experimentation in the B2B Domain: A Case Study

Olli Rissanen

Master's thesis
University of Helsinki
Department of Computer Science

Helsinki, November 17, 2014

HELSINGIN YLIOPISTO — HELSINGFORS UNIVERSITET — UNIVERSITY OF HELSINKI

Tiedekunta — Fakultet — Faculty	Laitos — Institution — Department
Faculty of Science	Department of Computer Science

Tekijä — Författare — Author
Olli Rissanen

Työn nimi — Arbetets titel — Title
Extending the development process towards Continuous Delivery and Continuous Experimentation in the B2B Domain: A Case Study

Oppiaine — Läroämne — Subject
Computer Science

Työn laji — Arbetets art — Level	Aika — Datum — Month and year	Sivumäärä — Sidoantal — Number of pages
Master's thesis	November 17, 2014	82

Tiivistelmä — Referat — Abstract

Delivering more value to the customer is the goal of every software company. In modern software business, delivering value in real-time requires a company to utilize real-time deployment of software, data-driven decisions and empirical evaluation of new products and features. Real-time deployment shortens the feedback loop and allows for faster reaction times, while data-driven decisions ensure that the development is focused on features providing real value. This thesis investigates practices known as continuous delivery and continuous experimentation as means of providing value for the customers in real-time. Continuous delivery is a development practice where the software functionality is deployed continuously to customer environment. This process includes automated builds, automated testing and automated deployment. Continuous experimentation is a development practice where the entire R&D process is guided by conducting experiments and collecting feedback. Such experiments can include deploying multiple versions of software with different properties, measuring their success based on some criteria and then selecting the best candidate for further development. As a part of this thesis, a deductive case study is conducted in a medium-sized software company. The purpose of the case study is to examine the development process of two different software products, and the transition towards continuous delivery and continuous experimentation. The main method for data collection is a semi-structured interview conducted on the members of the teams developing the products. The scope of this study is in the B2B domain, where the products are developed directly for other companies. The research objective is to analyze the challenges, benefits and systematic organisation of continuous delivery and continuous experimentation in this domain. As a result, the challenges and best practices in the transition are identified. The results suggest that technical challenges are only one part of the challenges a company encounters in this transition. For continuous delivery, the transition requires a company to also address challenges related to the customer and procedures. For continuous experimentation, the company also has to address challenges related to the customer and organizational culture. The benefits found from these practices support the case company in solving many of its business problems.

ACM Computing Classification System (CCS):
General and reference → Experimentation
General and reference → Measurement
General and reference → Validation

Avainsanat — Nyckelord — Keywords
Continuous delivery, Continuous experimentation, Development process

Säilytyspaikka — Förvaringsställe — Where deposited

Muita tietoja — Övriga uppgifter — Additional information

Contents

1 Introduction **1**
- 1.1 Overview . . . 1
- 1.2 Research questions . . . 3
- 1.3 Scope and limitations . . . 4
- 1.4 Contributions . . . 4

2 Literature: Continuous Delivery **5**
- 2.1 Continuous Delivery . . . 7
  - 2.1.1 Deployment pipeline . . . 9
  - 2.1.2 Impact on development model . . . 11
- 2.2 Challenges regarding Continuous Delivery . . . 12

3 Literature: Continuous Experimentation **12**
- 3.1 Continuous Experimentation . . . 12
  - 3.1.1 Components in Continuous Experimentation . . . 14
  - 3.1.2 Experimentation stages and scopes . . . 15
  - 3.1.3 Experimentation planning . . . 16
  - 3.1.4 Data collection . . . 17
  - 3.1.5 Data analysis . . . 17
  - 3.1.6 Roles in Continuous Experimentation . . . 18
- 3.2 Continuous Experimentation in embedded software . . . 18
- 3.3 Challenges regarding Continuous Experimentation . . . 20
- 3.4 Motivation . . . 20
- 3.5 Existing case studies . . . 21

4 Case study as a research method **23**
- 4.1 Case study . . . 23
- 4.2 Qualitative research . . . 24
  - 4.2.1 Template analysis . . . 26

5 Research design **28**
- 5.1 Objective . . . 28
- 5.2 Case descriptions . . . 29
  - 5.2.1 Dialog . . . 31
  - 5.2.2 CDM . . . 33
- 5.3 Context . . . 35
  - 5.3.1 Development process . . . 36
- 5.4 Methods . . . 39
  - 5.4.1 Data collection . . . 40
  - 5.4.2 Data analysis . . . 41

6 Findings **42**
6.1 Continuous Delivery: B2B challenges 42
6.1.1 Technical challenges 42
6.1.2 Procedural challenges 44
6.1.3 Customer challenges 47
6.2 Continuous Experimentation: B2B challenges 49
6.2.1 Technical challenges 50
6.2.2 Organizational challenges 53
6.2.3 Customer challenges 54
6.2.4 A/B testing 57
6.3 Continuous Delivery's benefits for the case company 59
6.4 Continuous Experimentation's benefits for the case company 61
6.5 Implementing Continuous Experimentation as a development process 65
6.5.1 Applying the general concept 65
6.5.2 Applying the experimentation model 67

7 Discussion **69**

8 Conclusion **73**
8.1 Summary 73
8.2 Limitations 75
8.3 Further research 76

References **77**

A Interview questions **81**

List of Figures

1 Organization evolution path [35]. 7
2 Continuous integration, delivery and deployment. 8
3 A basic deployment pipeline [20]. 10
4 Components of the development process [20]. 11
5 Continuous experimentation infrastructure [16]. 15
6 Scopes for experimentation [7]. 16
7 Continuous experimentation architecture for embedded software [15]. 19
8 Case company's challenges in continuous delivery. 70
9 Challenges and benefits of continuous experimentation. 71

List of Tables

1 Pitfalls to avoid when running controlled experiments on the web [9]. 21
2 Research questions and research methods. 30
3 Technical challenges in continuous delivery. 43
4 Procedural challenges in continuous delivery. 45
5 Customer challenges in continuous delivery. 47
6 Technical challenges in continuous experimentation. 50
7 Organizational challenges in continuous experimentation . . . 53
8 Customer challenges in continuous experimentation. 54
9 A/B testing challenges. 57

It's hard to argue that Tiger Woods is pretty darn good at what he does. But even he is not perfect. Imagine if he were allowed to hit four balls each time and then choose the shot that worked the best. Scary good. – Michael Egan, Sr. Director, Content Solutions, Yahoo (Egan, 2007)

1 Introduction

1.1 Overview

To deliver value fast and to cope with the increasingly active business environment, companies have to find solutions that improve efficiency and speed. Agile practices [8] have increased the ability of software companies to cope with changing customer requirements and changing market needs [13]. To even further increase the efficiency, shortening the feedback cycle enables faster customer feedback. Additionally, providing software functionality to collect usage data can be used to guide the product development and to increase the value generated for customers. Eventually the company running the most and fastest experiments outcompetes others by having the data to develop products with exceptional qualities [15].

The requirement of developing value fast has led to the evolution of a new software development model [7]. The model is based on three principles: evolving the software by frequently deploying new versions, using customers and customer data throughout the development process and testing new ideas with the customers to drive the development process and increase customer satisfaction. Currently more and more software companies are transitioning towards this model, called innovation experiment systems.

The logical starting point of this thesis was an open issue by the case company: "how to validate whether a feature would be succesful or not?" To provide an answer to this question, two software development models known as continuous delivery and continuous experimentation are analyzed as possible development models for the case company.

Continuous delivery is a development model aiming to shorten the delivery cycles and by automating the deployment process. It is an extension to continuous integration, where the software functionality is deployed frequently to customer environment, and the software is developed in a way that it is always ready for release. While continuous integration defines a process where the work is automatically built, tested and frequently integrated to mainline [17], continuous delivery adds automated acceptance testing and automated deployment. Continuous delivery therefore attempts to deliver an idea to users as fast as possible.

In continuous experimentation model the organisation continuously executes experiments to guide the R&D process. The process of guiding R&D through experiments is used to validate whether an idea is, in fact, a good idea. Fagerholm et al. define that a suitable experimentation system must be able to release minimum viable feature (MVF) or products (MVP) with suitable instrumentation for data collection, design and manage experiment plans, link experiment results with a product roadmap and manage a flexible business strategy [16]. The development cycle in continuous experimentation resembles the Build-Measure-Learn cycle of Lean Startup [40].

The process in continuous experimentation is to first form a hypothesis based on a business goals and customer "pains" [7]. After the hypothesis has been formed, quantitative metrics to measure the hypothesis must be decided. After this a minimum viable feature or minimum viable product can be developed and deployed. The software is then run for a period of time while collecting the required data. Finally, the data is analyzed to validate the hypothesis, and to either persevere or pivot the business strategy. To persevere with a chosen strategy means that the experiment proved the hypothesis correct, and the full product or feature can be implemented. However, if the experiment proved the hypothesis wrong, the strategy is changed based on the implications of a false hypothesis.

Many development models that carry a different name share the key principles with continuous experimentation. One of such is the innovation experiment system [7]. It is a development model where the development process consists of frequently deploying new versions and using customers and customer usaged data in the development process [7]. The model focuses on innovation and testing ideas with customers to drive customer satisfaction and revenue growth [7]. As there exists multiple terms for a seemingly similar practice, in the context of this thesis continuous experimentation refers to a development model following these principles: linking product development and business aspect, making data-driven decisions, reacting to customers present needs, validating hypotheses with experiments and using the results to steer the development process into a right direction.

This study is an exploratory deductive case study, which explores how continuous delivery and continuous experimentation can be applied in the case company. While existing studies of applying the development models exist [34, 7, 16], none of the studies focuses specifically in the B2B domain. This study specifically aims to identify the main requirements, problems and key success factors with regards to these approaches in the B2B domain. Extending the development process towards these practices requires a deep analysis of the current development process, seeking the current problems and strengths. Adopting both continuous delivery and continuous experimentaton also requires understanding the requirements of continuous delivery and continuous experimentation, and restrictions caused by the developed software products.

In this research, the units under the study are two teams and the two software products developed by these teams. By focusing on two different products, a broader view on the application and consequences of these development approaches can be gained. The other product, a marketing automation called Dialog, is used through an extensive user interface. The other product under inspection is CDM, a Master Data Management [31] solution running as an integrated background application. The objective of this thesis can therefore be summarized as follows:

Research objective To analyze the software development process transition towards shorter feedback cycle and data-driven decisions from the perspective of the case company, with respect to continuous delivery and continuous experimentation practices.

1.2 Research questions

The broader research objective is explored here. The research questions are structured to progress from a higher-level perspective to a more detailed view. The first two research questions are needed to study the requirements and benefits of these practices to rationalize the decisions to embrace these practices. Then, the third research question corresponds to the operational adjustments required to adopt these practices.

RQ1: What are the B2B specific challenges of continuous delivery and continuous experimentation?
Software development practices and product characteristics vary based on the domain and delivery model. Typical B2C applications are hosted as Software as a Service (SaaS) applications, and accessed by users via a web browser. In the B2B domain, applications installed to customer environments are very common. Running experiments on customer environments also requires a new deployment of the software product, possibly causing downtime and changing some functionality of the software. The purpose of this research question is to identify the challenges in these practices in the B2B environment. The research question is answered by conducting an interview to discover the current development process and challenges of the case company, and using these results and the available literature on continuous delivery and continuous experimentation to map the initial set of challenges these approaches will face in the case company.

RQ2: How do continuous delivery and continuous experimentation benefit the case company?
To rationalize the decision to adopt continuous delivery and experimentation in a company, the actual benefits to the business have to be identified. This research question aims to identify clear objectives for what is achieved by adopting continuous delivery and continuous experimentation. Sections of the interview aim to identify the current perceived problems of the case company related to deployment, product development, collecting feedback and guiding the development process. These problems are then compared to the benefits of these approaches found from the literature review.

RQ3: How can continuous experimentation be systematically organised in the B2B domain?
When the benefits and requirements are identified and the process is approved

by a company, it has to be applied in practice. This research question investigates how to apply continuous experimentation in the context of the case company. This question is answered by listing the deviations a continuous experimentation model by Fagerholm et al.[16] encounters in the case company. These deviations are based on the findings of the first research question.

1.3 Scope and limitations

This study focuses on the development process of the case company and its two software products in the B2B domain. Case studies only allow analytic generalisations, and the findings cannot be directly generalised to other cases. This limitation applies especially to applying continuous delivery and continuous experimentation in practice, since software products and customers inevitably differ. The background theory is drawn from literature focusing on continuous delivery, continuous experimentation, Lean Startup and innovation experiment systems.

The empirical research of this thesis has been carried out in the context of a medium sized information technology company, Steeri Oy. The company specializes in customer data on multiple fronts: CRM, BI, Master Data Management and data integrations, among others. The empirical research has applied case study methodology to offer deep understanding of the phenomena in their natural context. The tradeoff of a case study research is that it does not provide much room for direct generalizations outside the empirical context.

This thesis is organized as follows. The first chapter is the introduction at hand. The second and third chapter summarize the relevant literature and theories to position the research and to educate the reader on the body of knowledge and where the contributions are intended. The fourth chapter briefly explains the usage of case study and qualitative research as research methods, to provide background for the decision to utilize them in this particular study. The fifth chapter presents the research design: objectives, context and applied methods. The findings are then presented in the sixth chapter, organized according to the research questions. The seventh chapter summarizes and interprets the main results. Finally, the eight chapter summarizes the results of thesis and answers to the research question, discusses the limitations of the thesis and introduces further research avenues.

1.4 Contributions

The research objective of this thesis included the analysis of both continuous delivery and continuous experimentation. The third research question focuses only on continuous experimentation, as the attempt to study both approaches simultaneously would have required the analysis of too many variables at

once.

This research contributes to continuous delivery research by identifying issues found when the software is produced for customer companies. The study suggests that the challenges faced in the B2B context are multidimensional, and related to the technical, procedural and customer aspects. The findings are in align with and could be considered as extending some of the theoretical contributions by Olsson et al. [35], who researched the transition towards continuous delivery and an experimentational system. However, the study also suggests that continuous delivery corresponds to many of the case company's primary needs, such as improving the reliability of versions and reducing the time required for deploying versions.

This thesis also contributes to continuous experimentation [16] and organization evolution path [35] discussions by analysing the theories in B2B domain. These research areas are increasingly interesting in research, but none of the existing studies focuses purely on the B2B domain. There is growing research interest in these areas, however, most of the research takes place in B2C domain. Thus, the primary contribution of this thesis is to analyze the effects of this phenomena in a new domain.

The contributions to continuous experimentation research are firstly identifying the challenges a company faces in the transition towards an experimental system. These challenges are examined by utilizing two different software products, to provide two different perspectives. Secondly, the benefits of continuous experimentation are mapped to actual needs by the case company. Thirdly, an initial continuous experimentation model [16] is mapped to the case company's context.

2 Literature: Continuous Delivery

Software development models have been moving towards shorter feedback loops since iterative software development methods. Agile software development emerged from the iterative software development methods, with early implementations being development in the 1990's [12]. The general principles of agile software development are breaking the development to short iterations, maintaining a short feedback loop with the customer by having a customer representative, valuing working software over documentation and responding to change. The highest agile principle is to satisfy the customer through early and continuous delivery of software. Continuous delivery therefore has its roots in agile software development. Many agile models, such as Scrum [43], Extreme Programming [6] and Feature-Driven Development [37] further define their own strategies based on the agile principles. Scrum, for example, defines a framework for managing product development, without much focus on technical tools. Extreme Programming, on the other hand, includes many technical features, such as always programming in pairs, doing

extensive code review and unit testing.

Continuous delivery has also been influenced by Lean manufacturing, which is derived from the Toyota Production System [36]. Lean manufacturing is a streamlined production philosophy that aims to identify the expenditure of resources not used in creating value for the end customer, and eliminate these activities. Lean software development [38] has been developed based on the same idea. The purpose of lean software development is to eliminate waste, or anything that doesn't bring value to the customer, and to pursue perfection through smoothing the development flow. The principles in lean software development are eliminate waste, amplify learning, decide as late as possible, deliver as fast as possible, engage everyone, create quality and see the whole.

Lean software development, developed based on the lean manufacturing principles, is a development approach attempting to eliminate unnecessary work and to create value for the customer. Lean Startup [40] is a method for developing products based on the lean software development principles. The key component of Lean Startup is the Build-Measure-Learn loop, which guides the development process by defining a loop process where ideas are turned into products, which are then measured and the idea pivoted or persevered forwards.

The software development models have moved towards even shorter and shorter feedback loops in the recent years. A primary purpose of Lean Startup is to assess the specific demands of consumers and to fulfill those demands with the least possible resources. The Build-Measure-Learn loop is used to learn the customers actual needs and consecutively to steer business and product strategies to the correct direction. The tight customer feedback loop ensures that the product development doesn't include features and services not wanted by the customer.

Continuous delivery is a design practice where the software delivery process is automated, and thus the software can be deployed on short notice. A problem with agile software development is that no attention is paid to environments and tools used in software development, causing bottlenecks in the supply chain. In continuous delivery the technical solutions such as test automation and virtual environments are emphasized, smoothening the supply chain. Olsson et al. have investigated the evolution path of software companies, and have identified continuous delivery and "R&D as an Experiment System" as final stages [35] of the evolution path. Continuous delivery is also seen as a prerequisite for continuous experimentation, as it provides a way to quickly deploy new features.

In this chapter we first introduce the general principles of continuous delivery and continuous deployment. Then we introduce the software deployment pipeline and the effects of continuous delivery on the development model. Finally, we list challenges faced when applying continuous delivery.

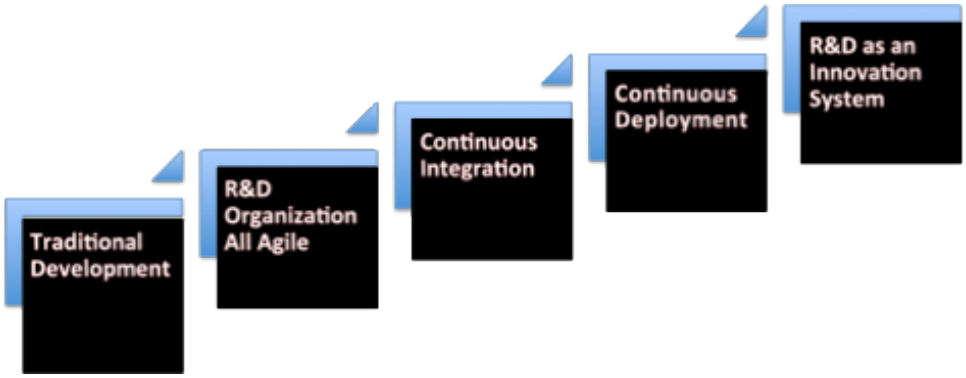

Figure 1: Organization evolution path [35].

2.1 Continuous Delivery

To be able to react to customers changing requests and manage multiple customer environments, the software product has to be deployed frequently and with different configurations. As the case company is working in an agile manner, requests change and adjust often. To improve the software product, the case company has a need for a deployment approach that has the following benefits:

- Fast feedback on changes
- Automation of repetitive actions
- Validating that deployed code has passed automated tests
- Traceable. large history records and changelogs
- Configurable per customer environment

Fast feedback is used to validate the functionality of the software and to ensure that quality requirements are met. Automation of repetitive actions ensures that the actions are performed exactly as instructed, without room for manual error. Reducing the amount of manual work also improves efficiency. With traceable history records, troubleshooting process can be shortened. With changelogs, the customer can be kept informed of the changes in new versions. A design practice that fills the aforementioned benefits is continuous delivery.

Continuous delivery is an extension to continuous integration, where the software functionality is deployed frequently at customer environment. While continuous integration defines a process where the work is automatically built, tested and frequently integrated to mainline [17], often multiple times a day, continuous delivery adds automated acceptance testing and deployment

to a staging environment. The purpose of continuous delivery is that as the deployment process is automated, it reduces human error, documents required for the build and increases confidence that the build works [20]. Continuous delivery solves the problem of how to deliver an idea to users as quickly as possible.

In an agile process software release is done in periodic intervals [8]. Compared to waterfall model it introduces multiple releases throughout the development. Continuous delivery, on the other hand, attemps to keep the software ready for release at all times during development process [20]. Instead of stopping the development process and creating a build as in an agile process, the software is continuously deployed to customer environment. This doesn't mean that the development cycles in continuous delivery are shorter, but that the development is done in a way that makes the software always ready for release. Continuous delivery differs from continuous deploy-

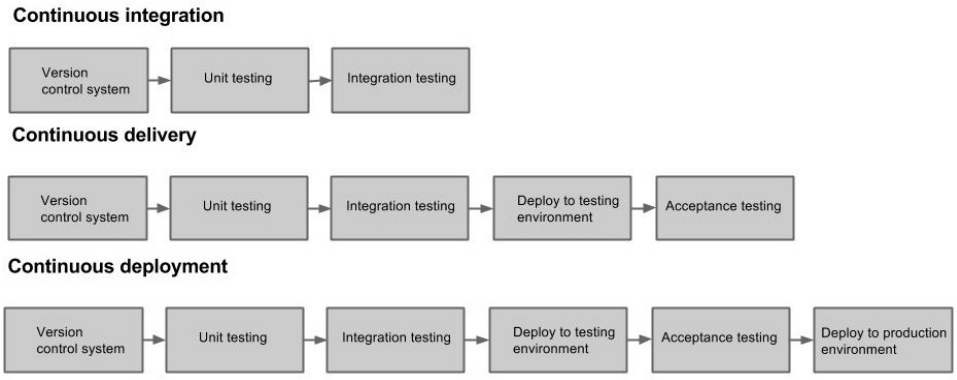

Figure 2: Continuous integration, delivery and deployment.

ment. Refer to Fig. 1 for a visual representation of differences in continuous integration, delivery and deployment. Both continuous delivery and continuous deployment include automated deployment to a staging environment. Continuous deployment includes deployment to a production environment, while in continuous delivery the deployment to a production environment is done manually. The purpose of continuous delivery is to prove that every build is deployable [20]. While it necessarily doesn't mean that teams release often, keeping the software in a state where a release can be made instantly is often seen beneficial.

Holmström Olsson et al. have researched the transition phase from continuous integration to continuous delivery, and have identified barriers that companies need to overcome to achieve the transition [35]. One such barrier is the custom configuration at customer sites. Maintaining customized solutions and local configurations alonside the standard configurations creates issues. The second barrier is the internal verification loop, that has to be

shortened not only to develop features faster but also to deploy fast. Finally, the lack of transparency and getting an overview of the status of development projects is seen as a barrier.

Olsson et al. define continuous deployment as a step in the typical evolution path for companies [35]. The practice of deploying versions continuously allows for faster customer feedback, ability to learn from customer usage data and to focus on features that produce real value for the customer. The authors state that in order to shorten the internal verification loop, different parts in the organization should also be involved, especially the product management as they are the interface to the customer [35]. They also note that finding a pro-active lead customer is essential to explore and form a new engagement model.

2.1.1 Deployment pipeline

An essential part of continuous deployment is the deployment pipeline, which is an automated implementation of an application's build, deploy, test and release process [20]. A deployment pipeline can be loosely defined as a consecutively executed set of validations that a software has to pass before it can be released. Common components of the deployment pipeline are a version control system and an automated test suite.

Humble and Farley define the deployment pipeline as a set of stages, which cover the path from a committed change to a build [20]. Refer to Fig. 2 for a graphical representation of a basic deployment pipeline. The commit stage compiles the build and runs code analysis, while acceptance stage runs an automated test suite that asserts the build works at both functional and nonfunctional level. From there on, builds to different environments can be deployed either automatically or by a push of a button.

Humble et al. define four principles that should be followed when attempting to automate the deployment process [21]. The first principle states that "Each build stage should deliver working software". As software often consists of different modules with dependencies to other modules, a change to a module could trigger builds of the related modules as well. Humble et al. argue that it is better to keep builds separate so that each discrete module could be built individually. The reason is that triggering other builds can be inefficient, and information can be lost in the process. The information loss is due to the fact that connection between the initial build and later builds is lost, or at least causes a lot of unnecessary work spent in tracing the triggering build.

The second principle states that "Deploy the same artifacts in every environment". This creates a constraint that the configuration files must be kept separate, as different environments often have different configurations. Humble et al. state that a common anti-pattern is to aim for 'ultimate configurability', and instead the simplest configuration system that handles

the cases should be implemented. Another principle, which is the main

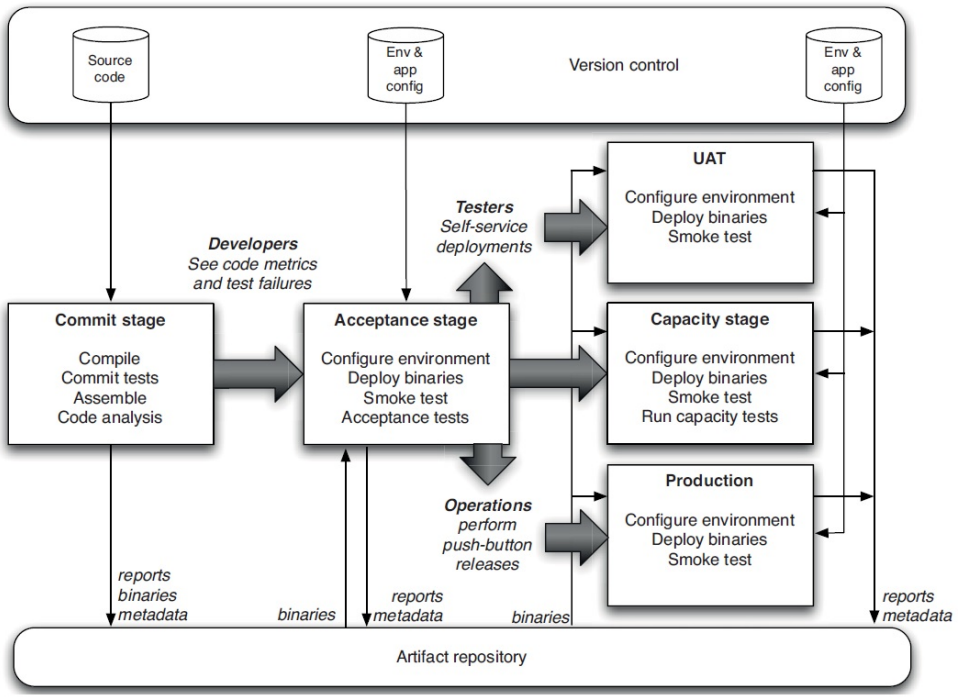

Figure 3: A basic deployment pipeline [20].

element of continuous deployment, is to "Automate testing and deployment". Humble et al. argue that the application testing should be separated out, such that stages are formed out of different types of tests. This means that the process can be aborted if a single stage fails. They also state that all states of deployment should be automated, including deploying binaries, configuring message queues, loading databases and related deployment tasks. Humble et al. mention that it might be necessary to split the application environment into *slices*, where each slice contains a single instance of the application with predetermined set of resources, such as ports and directories. *Slices* make it possible to replicate an application multiple times in an environment, to keep distinct version running simultaniously. Finally, the environment can be smoke tested to test the environments capabilities and status.

The last principle states "Evolve your production line along with the application it assembles". Humble et al. state that attempting to build a full production line before writing any code doesn't deliver any value, so the production line should be built and modified as the application evolves.

2.1.2 Impact on development model

A picture of the typical development process in continuous delivery is shown in Fig. 3. After the team pushes a change to the version control system, the project is automatically built and tests are triggered stage by stage [20]. If a test stage fails, feedback is given and the deployment process effectively cancelled. In a continuous delivery process, the last stages are approved and activated manually, but in a continuous deployment process the last stages are triggered automatically as well.

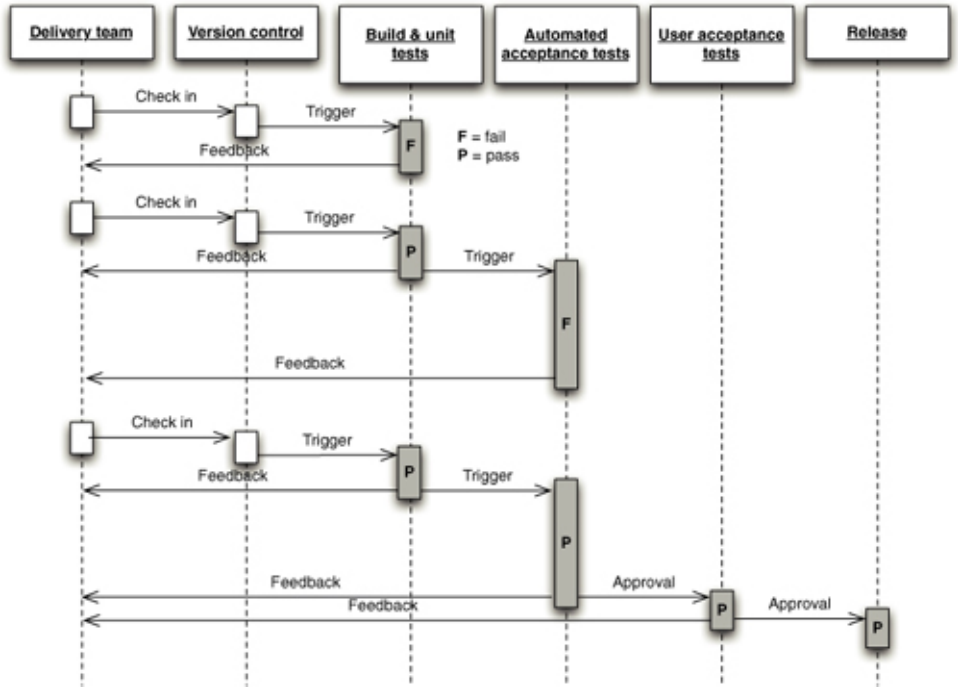

Figure 4: Components of the development process [20].

With automated deployment and production release, they become rapid, repeatable and reliable [20]. The releases can then be deployed more frequently, and at some point it might be possible to step back to early versions. This capability makes releasing essentially risk-free. When a critical bug is introduced, the version can be reverted to an earlier instance without the bug, which can then be fixed before a new release is made. The capability to deploy often also moves the development towards shorter features and smaller changes, which cannot cause as many integration problems as long-lasting feature branches with many commits [20].

2.2 Challenges regarding Continuous Delivery

Neely and Stolt found out that with a continuous flow, Sales and Marketing departments lost the track of when features are released [34]. Continuous delivery requires the company as a whole to understand the continuous flow process. Neely and Stolt solved this challenge by providing mechanisms by which the parties in need of knowledge regarding release times could track the work in real time [34].

One of the largest technical challenges is the test automation required for rapid deploying [21, 20]. The automated tests are often divided into multiple independend testing stages. Humble et al. note that automating the tests provide immediate value when implemented in the early stages of a project. It can, however, be a daunting task for larger projects in matured stages. Humble and Farley also bring up that implementing automated acceptance testing can be expensive and laborious, especially if communication inside the team is not effective [20].

Implementing the deployment infrastructure also requires knowledge from the development and operations team [20]. If the operations team doesn't understand the deployment process, the risk of making errors increases. Along with the technical implementationg, training and educating developers and operations team on the subject might be necessary depending on the prior knowledge levels.

Another challenge is to sell the vision and reasoning behind continuous delivery to the executive and management level [34]. Neely and Stolt state that it is important to set clear objectives for what the company is trying to achieve through continuous delivery. The authors also comment that there are many "shiny" that might cause distraction on the journey towards continuous delivery.

3 Literature: Continuous Experimentation

3.1 Continuous Experimentation

Continuous experimentation is a term created by Dane McKinley, a Principal Engineer at Etsy [2]. The problem that continuous experimentation addresses is that organizations often have many ideas, but the return-on-investment might be unclear and the evaluation itself might be expensive [28]. McKinley defines the key aspects of continuous experimentation as making small, measurable changes, staying honest and preventing the developers from breaking things. With staying honest, McKinley implies that the product releases are tracked and measured, so that it's possible to tell whether things went worse or better. McKinley also states that design and product process must change to accomodate experimentation, and that experimenting should be done with minimal version of the idea. When experimentations

are implemented and measured, counterintuitive results can be found, and planning to be wrong should be considered [2].

While continuous experimentation is only loosely defined by McKinley, it poses resemblance to multiple development models. One of such models is the Innovation Experiment System [7]. The key principles in these development models are data-driven decisions, linking product development and business aspect, reacting to customers present needs, turning hypotheses into facts, failing fast and steering the development process. Fagerholm et al. define that a suitable experimentation system must be able to release minimum viable products or features with suitable instrumentation for data collection, design and manage experiment plans, link experiment results with a product roadmap and manage a flexible business strategy [16]. The Build-Measure-Loop of Lean startup is similar in the sense that controlled experiments drive the development.

In Lean Startup methodology [40] experiments consist of Build-Measure-Learn cycles, and are tightly connected to visions and the business strategy. The purpose of a Build-Measure-Learn cycle is to turn ideas into products, measure how customers respond to the product and then to either pivot or persevere the chosen strategy. The cycle starts with forming a hypothesis and building a minimum viable product (MVP) with tools for data collection. Once the MVP has been created, the data is analyzed and measured in order to validate the hypothesis. To persevere with a chosen strategy means that the experiment proved the hypothesis correct, and the full product or feature can is implemented. However, if the experiment proved the hypothesis wrong, the strategy is changed based on the implications of a false hypothesis.

Holmström Olsson et al. have researched the typical evolution path of companies [35]. The final stage of the evolution phase is when development is driven by the real-time customer usage of software. Defined as "R&D as an 'experiment system'", the entire R&D system acts based on real-time customer feedback and development is seen as addressing to customers present needs. At the same time deployment of software is seen as a "starting point for 'tuning' of functionality rather than delivery of the final product". While the evolution from continuous delivery to innovation system wasn't explored, Olsson et al. anticipate that the key actions required are the automatic data collection components and capability to effectively use the collected data.

An experiment is essentially a procedure to confirm the validity of a hypothesis. In software engineering context, experiments attempt to answer questions such as which features are necessary for a product to succeed, what should be done next and which customer opinions should be taken into account. According to Jan Bosch, "The faster the organization learns about the customer and the real world operation of the system, the more value it will provide" [7]. Most organizations have many ideas, but the return-on-investment for many may be unclear and the evaluation itself may be expensive [28].

Continuous delivery attempts to deliver an idea to users as fast as possible. Continuous experimentation instead attempts to validate that it is, in fact, a good idea. In continuous experimentation the organisation runs controlled experiments to guide the R&D process. The development cycle in continuous experimentation resembles the build-measure-learn cycle of Lean Startup. The process in continuous experimentation is to first form a hypothesis based on a business goals and customer "pains" [7]. After the hypothesis has been formed, quantitative metrics to measure the hypothesis must be decided. After this a minimum viable product can be developed and deployed, while collecting the required data. Finally, the data is analyzed to validate the hypothesis. As the purpose of continuous experimentation is to design, run and analyse experiments in a regular fashion, quickly deploying experiments to the end user is a requirement. In the organization evolution path [35] continuous delivery is seen as a prerequisite for the experimental system.

Jan Bosch has widely studied continuous experimentation, or innovation experiment systems, as a basis for development. The primary issue he found is that "experimentation in online software is often limited to optimizing narrow aspects of the front-end of the website through A/B testing and inconnected, software-intensive systems experimentation, if applied at all, is ad-hoc and not systematically applied" [7]. The author realized that for different development stages, different techniques to implement experiments and collect customer feedback exist. Bosch also introduces a case study in which a company, Intuit, adopted continuous experimentation and has increased both the performance of the product and customer satisfaction.

3.1.1 Components in Continuous Experimentation

Kohavi et al. investigate the practical implementations of controlled experiments on the web [28], and state that the implementation of an experiment involves two components. The first component is a randomization algorithm, which is used to map users to different variants of the product in question. The second component is an assignment method which, based on the output of the randomization algorithm, determines the contents that each user are shown. The observations then need to be collected, aggregated and analyzed to validate a hypothesis. Kohavi et al. also state that most existing data collection systems are not designed for the statistical analyses that are required to correctly analyze the results of a controlled experiment.

The components introduced by Kohavi et al. are aimed primarily for A/B testing on websites. Three ways to implement the assignment methods are shown. The first one is traffic splitting, which directs users to different fleet of servers. An alternative methods is server-side selection, in which API calls invoke the randomization algorithm and branch the logic based on the return value. Last alternative is a client-side selection, in which the front-end system dynamically modifies the page to change the contents. Kohavi et

al. state that the client-side selection is easier to implement, but it severely limits the features that may be subject to experimentation. Experiments on back-end are nearly impossible to implement in such manner.

A different model for general experiments, not limited to A/B testing, has also been recently researched. While Fagerholm et al. state that "A detailed framework for conducting systematic, experiment-based software development has not been elaborated" [16], have they created an initial model consisting of required building blocks for a continuous experimentation system, depicted in 4.

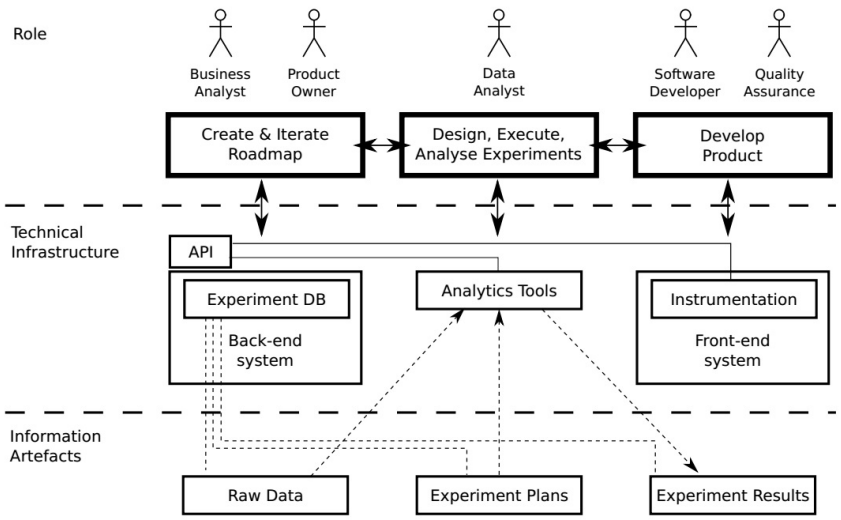

Figure 5: Continuous experimentation infrastructure [16].

The primary components of the introduced infrastructure are experiment database, analytics tools and instrumentation. The database stores raw data collected from the instrumentation and information related to the experiments, such as the logic for conducting the experiments and the experiment results. The analytics tools are used to perform data analysis on raw data from the experiment database to produce results regarding the experiment. The instrumentation is a part of the delivered software used to gather data regarding pre-defined metrics.

3.1.2 Experimentation stages and scopes

Fig. 5 introduces different stages and scopes for experimentation. For each stage and scope combination, an example technique to collect product performance data is shown. As startups often start new products and older companies instead develop new features, experiments must be applied in the correct context. Bosch states that for a new product deployment, putting

a minimal viable product as rapidly as possible in the hands of customers is essential [7]. After the customers can use the product, it is often not yet monetizable but is still of value to the customer. Finally, the product is commercially deployed and collecting feedback is required to direct R&D investments to most valuable features.

	Feature Optimization	New Feature Development	New Product Development
Pre-Development	Upsell advertising	Participatory design	Collaborative innovation
Non-Commercial Deployment	A/B testing	Feature alpha	Usage behavior tracking
Commercial Deployment	Multi-variate testing	Usage metrics	Cross-sell actions

Figure 6: Scopes for experimentation [7].

3.1.3 Experimentation planning

In continuous experimentation literature, the common way to plan experiments is to utilize the product strategy or roadmap [40, **?**]. In the Lean Startup Build-Measure-Learn loop, the Learn phase is essentially a decision to whether to persevere or pivot the product strategy. However, this requires that the Measure phase is extensively planned in advance, so that a clear hypothesis exists. Essentially, the Build-Measure-Learn loop is very similar to the scientific Plan-Do-Study-Adjust loop. The single exception is that the Build-Measure-Learn loop is based on an assumption that the customer needs cannot be known even with planning.

Fagerholm et al. emphasize the importance of analysing the data to test business hypotheses and assumptions, which have been created from the roadmap. These proven hypotheses can then be used to react to customers needs. The process for conducting experiments therefore starts from the business vision, then moves through planning and design phase to the technical implementation of the experimentation and data collection instrumentation.

The first step required for a controlled experiment according to Kohavi et al. is to define an Overall Evaluation Criteria (OEC) [30]. This criteria is the metric that the organization agrees to optimize with the experiment. Kohavi et al. cite Lewis Carroll saying "If you don't know where you are going, any road will take you there". Attention has also to be paid to prevent running underpowered experiments, by determining the minimum sample size required for an experiment to be statistically significant [29].

3.1.4 Data collection

To collect, aggregate and analyze the observations, raw data has to be recorded. According to Kohavi et al., some raw data could be for example page views, clicks, revenue, render time and customer feedback selections [28]. The data should also be annotated to an identifier, so that conclusions can be made from it. Kohavi et al. present three different ways for collecting raw data. The first solution is to simply use an existing data collection tool, such as Webmetrics. However, most data collection systems aren't designed for statistical analyses, and the data might have to be manually extracted to an analysis environment. A different approach is local data collection, in which a website records data in a local database or log files. The problem with local data collection is that each additional source of data, such as the back-end, increases the complexity of the data recording infrastructure. The last model is a service-based collection, in which service calls to a logging service are placed in multiple places. This centralizes all observation data, and makes it easy to combine both back-end and front-end logging.

Feedback collection can be divided into two groups, active and passive feedback [7]. Active feedback includes collection methods where the customer is aware that he or she is providing feedback. Such collection tools are for example surveys. Passive feedback is collected while the customer is using the system, without the customer being aware of the collection. Examples of such data collection includes tracking the time a user spends using a feature, the frequency of feature selections, or the path through the product the user takes to perform a certain task. Bosch states that what makes the major difference between Innovation experiment systems and tradititional software systems is the low cost and ease of data collection in the former.

An important aspect of collecting data is to pick the correct metrics. Metrics are numerical summaries used to compare variants of an experiment [29]. Metrics range from simple aggregates, such as total page views, to complex measures such as customer satisfaction. One of the common pitfalls for experiments is to choose the wrong overall evaluation criteria [9]. This is especially important in the planning phase, where the wanted improved metrics are initially chosen. However, if the data collection layer is complicated, attention has to be paid that the metrics transition correctly from the planning phase to the implementation of the collection phase.

3.1.5 Data analysis

To analyze the raw data, it must first be converted into metrics which can then be compared between the variants in a given experiment. An arbitrary amount of statistical tests can then be run on the data with analytics tools in order to determine statistical significance [29].

In the analysis phase, attention has to be paid to other variables that

might cause deviation of the OEC. As an example, Amazon showed a 1% sales decrease for an additional 100msec of performance [28]. In certain cases time could cause deviations of the OEC and complicate experiment results. Kohavi et al. suggest conducting single-factor experiments for gaining insights and making incremential changes that can be decoupled. Another suggestion is to try very different designs, or in backend applications for example completely different algorithms. Data mining can then be used to isolate areas where the new algorithm is significantly better. Finally, using factorial design when factors are suspected to interact strongly is recommended.

3.1.6 Roles in Continuous Experimentation

Fagerholm et al. define five different roles in continuous experimentation: Business Analyst, Product Owner, Data Analyst, Software Developer and Quality Assurance [16]. The business analyst along with the product owner are responsible for creating and updating the strategic roadmap, which is the basis of the hypotheses. The basis for decisions are the existing experimental plans and results stored in a database. A data analyst is used to analyze the existing experiments and results and to create assumptions from the roadmap. The analyst is also responsible for the design and execution of experiments. The data analyst is in tight collaboration with the software developer and quality assurance, who are reponsible for the development of MVPs and MVFs. The software designers create the necessary instrumentations used to collect the data required by the analyst.

Another approach for the roles is introduced by Adams et al., who conducted a case study on the implementation of Adobe's Pipeline. Adobe's pipeline is process that is based on the continuous experimentation approach [5, 3]. The innovation pipeline introduced in the study includes the following roles: designer, developer, researcher, product manager [3].

3.2 Continuous Experimentation in embedded software

The purpose of analysing experimentation in embedded software in the context of this thesis is that B2B applications and embedded applications share similarities in the release cycle. The innovation ideas for embedded software are prioritized as a part of the yearly release cycle. This resembles the release model found in the case company, where some customers prefer to have new versions shipped every three months. By analyzing how experiments are implemented and the data collected in embedded software, information on how long and short release cycles differ can be gained.

Eklund and Bosch have applied the innovation experiment system to embedded software [15]. In the study an embedded architecture for realising an innovation experiment system is presented. The study concentrates on what to evaluate in experiments in embedded software. The infrastucture

for enabling the innovation experiment system is shown in 6.

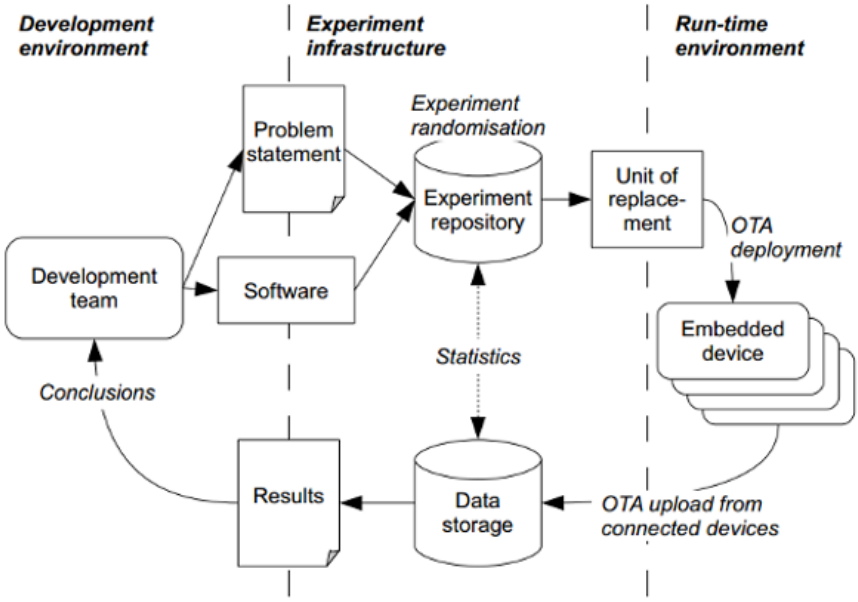

Figure 7: Continuous experimentation architecture for embedded software [15].

The major difference between embedded and traditional software in terms of experimentation is the data collection phase. With embedded software, the data has to be either transferred over-the-air or on-board. With on-board measurements and analysis, the experiment manager autonomously controls when to run which experiments.

The study also lists possible experiment scenarios supported by the architecture [15]. Possible experiment cases with the architecture are as follows:

- How long does it take to . .
- Which of . . . is most often used/accessed/. . .
- Identify behaviour that is not intended, e.g. menu selection followed by "back" indicates that the user made a mistake.
- Are there any features that are not used?
- Be able to evaluate competing designs based on the answers above, i.e. A/B testing (AKA. split testing).

3.3 Challenges regarding Continuous Experimentation

It is commonly found in the studies that the practiced experimentation is limited to optimizing narrow aspects of the front-end through A/B testing [7]. Additionally, experimentation is not often systematically applied [7]. Typically the imbalance between number of ideas that exist in the organization and the number of concepts that are tested with customers is enormous [7].

Some companies use traditional metrics such as the Net Promoter Score [39] to gather feedback. These metrics however fail to guide the development process by providing timely feedback, and instead are backward looking and focusing on the whole product [7].

One of the biggest challenges is also the architectural requirements set by the product to allow experimentation. According to Jan Bosch: "Collecting customer or performance data early in the innovation process requires changes to the R&D processes, as customers need to be involved much earlier and deeper in the process. This also requires architectural changes of the product and platform to determine customer preference and interest." [7].

Crook et al. investigate the common pitfalls encountered when running controlled experiments on the web [9], which are listed in Table II. The Overall Evaluation Criteria (OEC) used in the first pitfall is a quantitative measure of the experiments objective. In experimental design, Control is a term used for the existing feature or a process, while a Treatment is used to describe a modified feature or a process. As a motivator for the first pitfall, Crook et al. introduce an experiment where the OEC was set to the time spent on a page which contains articles. The OEC increased in the experiment implementation, satisfying the objective. However, it was soon realized that the longer time spent on a page might have been caused by confusion of the users, as a newly introduced widget was used less often than a previous version of it. Aside from picking a correct OEC, common pitfalls deal with the correct use of statistical analysis, robot users such as search engine crawlers, and the importance of audits and control.

3.4 Motivation

The software industry is growing due to increasing digitalisation, and it is especially important for new and old companies alike to identify the problems relevant to customers and to stay flexible. Identifying the customers problems allows the companies purely focus on creating value, while staying flexible means quickly diverting development efforts towards currently desired features. The importance of continuous innovation is increasingly recognized in the literature [46]. Companies are also increasingly interested in utilizing real-time customer usage of the software to shorthen feedback loops [35].

Pitfall 1	Picking an OEC for which it is easy to beat the control by doing something clearly “wrong” from a business perspective
Pitfall 2	Incorrectly computing confidence intervals for percent change and for OECs that involve a nonlinear combination of metrics
Pitfall 3	Using standard statistical formulas for computations of variance and power
Pitfall 4	Combining metrics over periods where the proportions assigned to Control and Treatment vary, or over subpopulations sampled at different rates
Pitfall 5	Neglecting to filter robots
Pitfall 6	Failing to validate each step of the analysis pipeline and the OEC components
Pitfall 7	Forgetting to control for all differences, and assuming that humans can keep the variants in sync

Table 1: Pitfalls to avoid when running controlled experiments on the web [9].

The process of guiding product development with continuous experiments diminishes guesswork and instead provides a systematic approach. Avinash Kaushik states in his article Experimentation and Testing primer [1] that "80% of the time you/we are wrong about what a customer wants". Mike Moran also found similar results in his book Do It Wrong Quickly, stating that Netflix considers 90% of what they try to be wrong [33]. A way to tackle this issue is to adopt a process of continuous experimentation. Ron Kohavi at Microsoft wrote "Features are built because teams believe they are useful, yet in many domains most ideas fail to improve key metrics" [27]. Additionally, Regis Hadiaris from Quicken Loans wrote that "in the five years I’ve been running tests, I’m only about as correct in guessing the results as a major league baseball player is in hitting the ball. That’s right - I’ve been doing this for 5 years, and I can only "guess" the outcome of a test about 33% of the time!" [32].

3.5 Existing case studies

Existing case studies regarding continuous experimentation have been conducted with varying scopes of investigation. Olsson et al. conducted a multiple-case study to explore companies moving towards continuous deployment and continuous experimentation [35]. In the transition phase from continuous integration stage to continuous deployment, the study investigates a telecommunications company with a highly distributed organization. The study considers transition from continuous delivery to continuous experimentation as a future research interest.

Fagerholm et al. analytically derive a continuous experimentation model

and apply it in practice in a startup company [16]. The study introduces an initial continuous experimentation model, examines the preconditions for setting up an experimental system and describes the building blocks required for such a system. The case company in this study is a relatively new startup company, and the company operates in the B2C domain. The study states that it is yet not clear how to determine the suitability of an experimental approach in specific situations. This is in line with the goal of this thesis: analyze the continuous experimentation approach in the B2B domain with two different software products.

Steiber et al. [46] have studied continuous innovation in Google. The study defines continuous innovation as "an experimental iterative process that operates successively to solve problems in markets characterized by turbulence, uncertainty and complex interactions". The study explores the organizational characteristics for continuous innovation in rapidly changing industries. The study finds that the entire organisation of Google is involved in continuous innovation. The study also states that there is a need for a more comprehensive analytical framework for continuous innovation, and also a need to study how to organize continuous innovation and continuous improvements.

Adams et al. conducted a case study on the implementation of Adobe's Pipeline, a process that is based on the continuous experimentation approach [5, 3]. The innovation pipeline introduced in the study consists of the following steps:

- Collect user problems
- Identify problems worth solving
- Prototype possible solutions
- Live user testing + public releases
- Refine, pivot or kill the idea
- Build products that understand the user

The pipeline team consists of the following roles: designer, developer, researcher and product manager. The general principles found in the study are as follows:

- The idea will change
- We're only right when the market tells us so
- Create fast. improve constantly
- Find the simplest thing that could possibly work

- Define "success" before we build
- Fail forward

The study is carried out in the company with over 10,000 employees. The study states that "there is a wealth of understanding about how small companies can innovate in a low-cost way in order to make small products viable".

Kohavi et al. have widely studied online experimentation at microsoft [30]. The experiments have been mostly performed in Microsoft's web sites with A/B testing, and the efforts have been proven succesful. This success further motivates the goal of this thesis of applying experiments in a different domain and different software products. However, as the study only consists of A/B testing, this thesis also inspects other kind of experiments performed in a continuous manner.

4 Case study as a research method

4.1 Case study

Case study is a way to collect data through observation and to test theories in an unmodified setting [48]. The case in a case study is the situation, individual, group or organization that the researchers are interested in [41]. Kitchenham et al. found case studies important for industrial evaluation of software engineering methods and tools [25]. Runeson and Höst define it as a suitable research method for software engineering, since it studies contemporary phenomena in its natural context [42]. It might not be possible to generate causal relationships from case studies as compared to controlled experiments. However, they provide a deeper understanding of the unit under study [42]. An opposite type of research would be a formal experiment, with a narrow focus and control on variables.

Case studies can serve different purposes. Robson defines four purposes as Exploratory, Descriptive, Explanatory and Improving [41]. In an exploratory case study, the purpose is to seek new insights, find out what is happening and generate ideas and hypotheses for new research. Descriptive case study attempts to portray a situation or a phenomenom. Explanatory case study attempts to seek and explanation to a problem or situation, mostly in a causal relationship. An improving case study tries to improve an aspect of the studied phenomenom.

Different approaches to the order of data collection, analysis and generalization can be categorized into inductive, deductive and abductive approaches [11]. In an inductive case study the research is conducted by first collecting the data, then looking for patterns and forming theorie that explain those patterns. Therefore an inductive approach moves from data to theory. A deductive approach, on the other hand, is in reversed order; the approach

moves from general level to a more specific level. A deductive approach begins with studying what others have done, and then testing hypotheses that emerge from those teories. An abductive approach starts with the consideration of facts, or particular observations. These observations are then used to form hypotheses that relate them to a fact or a rule that accounts for them. The facts are therefore correlated into a more general description, that relates them to a wider context.

Runeson and Höst define the case study structure in five steps [42]. First the case study is designed: objectives are defined and the study is planned. Then procedures and protocols for data collection are defined. Then the evidence is collected by executing the data collection on the studied case. The collected data is then analysed, and finally the results are reported.

Triangulation is a way to increase the reliability and validity of the findings. Triangulation means using different data collection methods or angles, and providing a wider picture of the case. Triangulation is especially important for qualitative data, but can also be used for quantitative data to compensate for measurement or modeling errors [42]. Stake defines four ways to apply triangulation: Data triangulation, Observer triangulation, Methodological triangulation and Theory triangulation [45]. In data triangulation, multiple data sources are used, or the data is collected at different occasions. In observer triangulation more than one observer is used in the study. In methodological triangulation different types of data collection methods are combined, such as qualitative and quantitative methods. In theory triangulation alternative theories or viewpoints are used.

Data collection in case studies tend to lean towards qualitative data that provides a richer and deeper view as compared to quantitative data [42]. As a data collection method, semi-structured interviews are common in case studies [42]. However, generalizing the results of a case study is often a subject of internal validity [26]. It is especially important in a case study to address reliability and validity of the findings. Even with good faith and intention, biased and selective accounts can emerge [41].

4.2 Qualitative research

Qualitative research attempts to answer to questions "Why?"" and "How?" instead of "What?", "Where?" and "When?" as compared to quantitative research. The most common way to collect qualitative data is via an interview. Seaman defines interviews as a way to collect historical data, opinions and impressions about something [44]. The interview can be either structured, unstructured or semi-structured. In a structured interview the interviewer asks all of the questions, and the objectives are very specificed. The focus of a structured interview is to find relations between constructs, and the objective is descriptive and explanatory [42]. In an unstructured interview the topic is broadly defined, and questions are asked also by the interviewee. In an

unstructured interview open-ended questions are typical, and unforeseen types of information can be gained. The focus of an unstructured interview is on how the interviewee qualitatively experience the phenomenom, and the objective is exploratory [42]. A semi-structure interview is a mixture of both open-ended and specific questions. Semi-structured interviews focus on how individuals qualitatively and quantitatively experience a phenomenom, and the objective is both descriptive and explanatory [42].

Seaman has explored qualitative research in software engineering context, and he defines different ways to analyse qualitative data [44]. A typical way of analyziz is coding, which means extracting values for quantitative variables from qualitative data. Coding can also be used to organize the data based on themes and views. Seaman states that qualitative data is a lot richer than quantitative data, and using qualitative methods increases amount of information, diversity of information and confidence in results.

The basic objective of data analysis is to form conclusions and theories from the data based on a chain of evidence. Qualitative data analysis can be divided into two different parts, hypothesis generating techniques and hypothesis confirmation techniques [42]. Both of these techniques can be exploratory and explanatory case studies [42]. In hypothesis generating techniques, a statement or proposition is extracted from the field notes that is supported in multiple ways by the data, and then used as a hypotheses [44]. The data can also be divided into cases using cross-case analysis, where two groups based on some attribute are examined for similarities and differences. In the confirmation of theory, weight of evidence is built using triangulation of different data sources.

Robson introduces common features of qualitative data analysis in a sequential list [41]. The first step is to code the initial set of data. Then, comments and reflections (referred to as 'memos') are added. After this the data is processed, similar phrases, patterns, themes, relationships and differences between sub-groups are identified. These identified patterns are then used to focus the next data collection phase. Gradually a set of generalizations are formed, that cover the consistencies discerned in the data. These generalizations are then linked to a formalized body of knowledge in the form of constructs or theories.

In the sequential list, hypotheses are identified after the data has been coded. The hypotheses are then used to guide the following data collection process in an iterative approach. During the iterations, some generalizations can be formed, which eventually form a formalized body of knowledge as a final result.

Hypothesis generating techniques are for example "constant comparison method" and "cross-case analysis" [44]. In the constant-comparison method the field notes are coded, text pieces are grouped into patterns and propositions that are strongly supported by the data are made. The propositions are then validated against any new data. In the cross-case analysis method

data is divided into cases, such as two groups based on the same attribute. Pairs of cases are then compared to determine validations and similarities.

The data analysis can be conducted with multiple different approaches, with each having a varying degree of formalism. Robson lists four such approaches, from least formal to most formal: Immersion approaches, Editing approaches, Template approaches and Quasi-statistical approaches [41]. Immersion approaches are the least format, relying mostly on the interpretive skills of the researcher and having a low level of structure. Editing approaches include the usage of some codes which the researcher decides based on findings during the analysis. Template approaches, also known as template analysis, use priori codes to develop a coding template, organising the data based on themes. Quasi-statistical approach is the most formal approach, including quantitative calculations, such as word frequencies. Runeson and Höst state that editing approaches and template approaches are most suitable for software engineering case studies, as a clear chain of evidence is hard to obtain in immersion approaches, and word frequencies are hard to interpret [42].

4.2.1 Template analysis

Template analysis doesn't describe a clearly defined method, but rather refers to a related group of techniques used to thematically organise and analyse textual data [24]. Template analysis is a relatively structured process to analyse textual data, and has mostly been used to analyse data from individual interviews [23]. The key component in template analysis is the generation of a coding template, which is initially formed on the basis of a subset of the data and then applied to further data.

King defines code as a label attached to a section of text to index it as a relating to a theme or issue in the data [24]. As an example, codes can be defined to identify points in the text where an interviewee mentions particular categories of presenting problems. These kind of codes can be descriptive, and require no further analysis by the researcher. However, also codes requiring more interpretation can be defined. In template analysis, a key feature is to form a hierarchical organization of codes [24]. This way groups of similar codes are clustered together to form higher-order codes. This way the researcher can analyse the data at different levels of specifity, with higher-order codes giving a broad view and lower-order codes a more specific view. Parallel coding, where the same text segment is categorised with two different codes, is also allowed.

The process of template analysis is to first create an initial template based on some pre-defined codes [24]. Then the template is revised by working through the data systematically while identifying and coding sections of interest in the text. Based on these findings, the initial template is modified, and finally develops to its final form. Modifications can be either insertion,

deletion, changing scope or changing higher-order classification [24]. The final template is such that no sections of text related to research questions remains uncoded.

The template is then used in interpretation to provide an account. King lists various guidelines that can be used in the interpretation of coded data [24]. The first guideline is to compile a list of all codes with some indication of frequency. The distribution of codes can then help to draw attention to the aspects of data with either multiple codes or missing codes. For example, if one interview in a set of interviews is missing a certain theme, analysis for possible reasons behind that missing code can be made. However, King suggests that while patterns can be used to draw attention to certain parts of the textual data, frequencies alone cannot be to gain any meaningful information.

Another guideline is selectivity. King states that every code cannot be interpreted to equal degree of depth, and themes most relevant to understanding the phenomena under study must be identified. Prior assumptions of the researched shouldn't limit the analysis. Yet another guideline is openness, which King describes as taking themes judged as marginal into account. Themes that aren't of direct relevance to the initial research questions shouldn't be disregarded, as they can play a useful role in adding to the background of the study. Also themes lying outside the scope of the study should be included in the analysis, if they're considered to cast light on the interpretation of central themes in the study. Finally, King explains that purely linear relationships between codes, such as hierarchical relationships with subsidiary codes next to parent codes, "may not reflect the kinds of relationships a researcher may want to depict in his or her analysis". Instead, maps, matrices and other diagrams can be used to explore and display findings, and analysis doesn't have to stop when a full linear template is produced.

Finally, an account of the interpreted data has to be presented. As with interpretation, King provides three common approaches to presentation [24]. First of the approaches is a set of individual case studies, followed by discussion of differences and similarities between cases. This approach gives a lot of perspective of individuals, but can be confusing if the amount of participants is large. The second approach is an account structured around the main themes identified, drawing illustrative examples from each transcript as required. King states that this is the approach that most readily provides a clear thematic discussion, but the dangers is in drifting towards generalizations and losing sights of individual experiences. The third approach is a thematic presentation of the findings, using a different individual case-study to illustrate each of the main themes. The main problem here lies in selecting the cases which represent the themes in the data as a whole. King also adds that no matter which approach is used, direct quotes from participants is essential.

5 Research design

Research design depicts the strategic decisions that guide how research is carried out [10]. This includes the studied phenomena, method selection, data-gathering and analysis. Different perspectives towards the researched subjects are usually categorized into two paradigms: positivist and constructivist [18]. Positivism contents that an objective reality, which can be studied, captured and understood, exists. Postpositivism argues that reality can never be fully apprehended, but only approximated. Postpositivism therefore relies on multiple methods to capture as much of reality as possible. Constructivism assumes that multiple realities exist. A critical paradigm lies somewhere between positivism and constructivism, assuming that our ability to know of this reality is imperfect. To understand as much of reality as possible, it must be subject to wide critical examination.

This thesis utilizes the critical realist perspective. Critical realism studies people's perceptions to gain understanding of the reality that exists beyond these perceptions [19]. Critical realism is especially suitable for case study research if the process involves thoughtful in depth research with the object of understanding why things are as they are [14].

During this research, the author was an employee at Steeri Oy. Consequently, this case study is based on the participant-observer research method [47]. Data triangulation is ensured by combining the observer results and interview results. The logical starting point of this thesis was the need by the case company to find a way to validate whether a feature is succesful or not.

5.1 Objective

The purpose of this study is to seek ways to improve the product development process of two products within the case company, using practices known as continuous delivery and continuous experimentation. Existing documented implementations of experimentational systems are primarily executed in the B2C domain, often with a Software as a Service (SaaS) product. An example of this is the Microsoft EXP platform [4]. The focus of this study is in the B2B domain, with two software products that are not used as SaaS.

Research objective To analyze the software development process transition towards shorter feedback cycle and data-driven decisions from the perspective of the case company, with respect to continuous delivery and continuous experimentation practices.

The study is an exploratory deductive case study, which aims to explore how continuous delivery and continuous experimentation can be implemented in the case company. The study specifically aims to identify the main requirements, problems and key success factors with regards to these approaches

in the context of the case company. Integrating these approaches to the development process requires a deep analysis of the current development process, seeking the current problems and strengths. Adopting both continuous delivery and continuous experimentaton also requires understanding the requirements of continuous delivery and continuous experimentation. To narrow the scope of the thesis, the focus of this thesis is in the development process of two different software products. The two different products were chosen so that cross-case analysis and therefore more generalizations can be made.

The research process starts by reviewing literature on both continuous delivery and continuous experimentation. In the literature review, main goals are to identify existing requirements and success factors in these approaches. The interview and documentation of the development process is then used to analyze and identify the pain points the company currently has in its development process. Then continuous delivery and continuous experimentation processes are viewed from the viewpoint of the case company, comparing the previously found benefits to the pain points in the case company's development process. As a result of viewing the practices from case company's viewpoint, necessary restrictions and requirements encountered in the B2B domain are obtained. Finally, the continuous experimentation model [16] is applied to the case company, and deviations listed.

The research questions and research methods are summarize in Table 2.

In this thesis along with confirming existing theories, we do aim to generate new concepts and theories. The theory and case analysis are continuously matched to gain insights coherent with both theory and empirical observations.

5.2 Case descriptions

Experimental context needs the three elements: background information, discussion of research hypotheses, and information about related research [26]. The two former will be discussed here, and the latter is introduced in the second and third chapters.

The company in question is Steeri Oy, which is a medium-sized company specializing in managing, analyzing and improving the usage of customer data. In this research, the units under the study are two teams responsible for developing two different software products. The first team is CDM, which is focusing on the respectively named software application. The other team is Dialog, also working for a respectively named software application. Both of the teams are managed by a similar structure, consisfting a team leader and a product owner, quality assurance and commercialization experts.

Knowledge gap	Research question	The focus of analysis	Data
B2B specific challenges in the practices	RQ 1: What are the B2B specific challenges of continuous delivery and continuous experimentation?	Template analysis of the interview results	12 interviews, documentations of the development process, literature
Benefits of the practices after reviewing the challenges	RQ 2: How do continuous delivery and continuous experimentation benefit the case company?	Template analysis of the interview results	12 interviews, literature
What deviations are there to existing models	RQ 3: How can continuous experimentation be systematically organised in the B2B domain	Conceptual analysis of continuous experimentation model by Fagerholm et al. [16], and application of the model to case company.	12 interviews, documentations of the development process, author observations

Table 2: Research questions and research methods.

Steeri Oy has a functional structure, with each business area forming independent teams based on the products and projects. The business areas of the whole company range from IT consulting to CRM implementations and development of own software products. To scope down the thesis, only teams doing development of two own software products are in the scope.

The unit of analysis is the development process of the two teams. The whole development process consists of the development framework used, but also of the interaction with customers, tools used in the current development process and the roles of individuals in the teams. The unit of analysis is studied by focusing on interviewing individual members at different positions in the teams. The purpose of the interview is to identify the pain points and ways of working in the current development process, which are then viewed against the benefits of continuous delivery and continuous experimentation found from existing literature. Analysis of implementing a continuous experimentation model is then done based on the interview results.

The development process of the team is elaborated in detail in the chapter "State of the practice". In the following sections the general characteristicts of company, the two products and applied development processes are identified.

The general charasteristics of the two teams under scope are then compared. The general characteristics are as follows.

- Product description
- Product environment
- Product architecture
- Usage
- Team composition
- Development practice
- Business model
- Programming languages
- Development tools
- Version releasing
- Unit testing
- Acceptance testing
- Feedback collection
- Usage data collection
- Feedback and usage data usage

The first three characteristics give us information on how the teams develop their software products. Usage then defines how and by whom the product is used in practice. The team composition and development practice provide important information on how the teams are composed and how they operate. Business model clarifies how the product is marketed, and how the product development is navigated in order to maximize revenue. Programming languages and development tools provide insight on the technical aspect of the development. Version releasing, unit testing and acceptance testing are taken into consideration when analyzing the transition towards continuous delivery, which eventually will automatize these phases. Feedback collection, usage data collection and data usage are analyzed to provide basis for the current feedback process operated by the case company, in order to gain insight on restrictions for continuous experimentation.

5.2.1 Dialog

The Dialog subteam focuses on developing an online marketing automation, iSteer Dialog. It is a software product designed to allow effective marketing on multiple channels, such as e-mail and websites, and to automate repetitive tasks. The product can be integrated to existing customer relationship management (CRM) and data warehouse solutions, and the data stored in CRM can then be effectively used for marketing purposes.

Product description. Dialog is a multi-channel online marketing automation tool. The purpose of the product is to allow effective marketing on channels such as e-mail and websites, and to automate repetitive tasks. The product can be integrated to existing CRM and data warehouse solutions, and the data stored in the CRM can then be effectively used for marketing purposes. The product is primarily used through a

comprehensive user interface by marketing professionals of customer companies.

Product environment. The product can be run on either servers provided by the case company or on the customers internal servers. Most of the customers use the product on the case companys own servers. Each customer using a provided server has an individual server and configurations, due to different integrations to the customers own systems. Customers running the product on their own servers form a minority.

Product architecture. The product consists of a single maven-packaged project. Along with the actual product, several additional third-party or custom components are required. Such components are for example Typo3, which is an open source web content management framework, and a custom component used for SMS integration. Most of the components reside in the version control system, and the main product additionally resides in the CI system. Custom integrations to customers data sources, often based on a third party ESB tool, are individual components as well.

Usage. Dialog is used by marketing professionals to automatize repetitive tasks. The user amount varies, but it is generally less than five human users per customer company.

Team composition. The team consists of 3 software developers, a manager for commercialization, a product owner and a quality assurance engineer. The software developers don't have designated roles, and each developer is involved in every aspect of the product development process.

Development practice. Development is based around a prioritized kanban board. Features are split into tasks by the product owner or team leader, and inserted into the kanban board. The developers then pick a single task from the sprint backlog column, complete it, and move it to the review column. From there the task undergoes quality assurance and delivery to the customer environment. The detailed development practice is introduced later in this chapter. The team is in the state of continuous integration, but the quality of automated tests isn't yet at the required level to lay confidence in builds working.

Business model. The product is still in the development phase, and is not sold as a ready package. The productization process is however ongoing, and the ideal situation is that the product can be sold as a ready package with only minor configurations per customer. The product is currently sold as a custom project, and requires installment to the customer environment and integration to the customers systems.

The customer has also be trained to use the product, and after the product goes live the cooperation is continued with a support contract.

Programming languages. The product is written mainly in Java, with the front-end built using JavaScript.

Development tools. The team uses GitHub as the version control system, and Jenkins as the integration server. Trello is used as the kanban board to manage projects, and Bugzilla is used as the bugtracker.

Version releasing. A new version is released when a customer requires a certain feature, and that feature has been completed. Updates for other customers are then released every two to three months on average. The release cycle length for customers in the development phase ranges from 2 to 4 weeks.

Unit testing. Unit testing is done by the developers whenever a feature is written.

Acceptance testing. Acceptance testing is performed by the product owner and quality assurance engineer of the team whenever a version is deployed to the customer environment.

Feedback collection. During development, acceptance testing is the main source of feedback. Some of the features are also developed for pilot customers. Feedback is also collected from customers in retrospective meetings after the projects, and received directly from customers via e-mail and meetings. Feedback isn't automatically collected in any way. The team has plans for in-product surveys and usage data analysis.

Usage data collection. Usage data isn't collected at all.

Feedback and usage data usage. The received feedback is discussed within the team to improve the development process or certain parts of the product. The feedback isn't systematically used,.

The software is configured and integrated to the customer environment as project work. This deployment doesn't require additional code as per customer, but only different configuration files.

5.2.2 CDM

The CDM subteam focuses on building a Master Data Management [31] solution, which integrates multiple customers data sources such as CRM, ERP and billing system to create a single point of reference.

Product description. CDM is a Master Data Management [31] solution, which is used to integrate multiple customer data sources to create a simple point of reference. These data sources can include for example CRM, ERP and a billing system. The product also manages the data by removing duplicate records, matching same records and cleaning and validating the data with the help of external data such as the resident registration database. The product has an user interface that can be used for testing, but the product is primarily a background application without users.

Product environment. The product is installed to a customer environment, and certain custom specific configurations and rules have to be implemented for each customer. Such configurations are for example the integrations to customers systems and external identification services used.

Product architecture. The product consists of several maven-packaged projects, which in the build stage are added as a dependency under a single main project. The whole product can thus be packaged into a single web archive file.

Usage. CDM is an integrated application, which is primarily only used by other applications. Therefore humans are only using CDM for debugging purposes.

Team composition. The team consists of 5 software developers and a team leader. The software developers don't have designated roles, and each developer is involved in every aspect of the product development process.

Development practice. Development is based around a prioritized kanban board. Features are split into tasks by the product owner or team leader, and inserted into the kanban board. The developers then pick a single task from the sprint backlog column, complete it, and move it to the review column. From there the task undergoes quality assurance and delivery to the customer environment. The detailed development practice is introduced later in this chapter. The team is in the state of continuous integration, but the quality of automated tests isn't yet at the required level to lay confidence in builds working.

Business model. The nature of the product requires customer integrations to customers systems, which are currently mostly done as manual work. In the ideal situation the product contains automated integration, and could be sold as a ready product with minor configuration work. Currently the product is sold as a custom project, and requires installment to the customer environment and integration to the customers systems.

After the product goes live the cooperation is continued with a support contract.

Programming languages. The product is written in Java.

Development tools. The team uses GitHub as the version control system, and Jenkins as the integration server. Trello is used as the kanban board to manage projects.

Version releasing. A new version is released when a customer requires a certain feature, and that feature has been completed. Updates for other customers are then released every two to three months on average. The release cycle length for customers in the development phase ranges from 2 to 4 weeks.

Unit testing. Unit tests are required for new features, and are written by the developers during of after the feature implementation.

Acceptance testing. Acceptance testing is performed by either a developer or the team leader whenever the version is deployed to a customer environment.

Feedback collection. During development, acceptance testing is the main source of feedback. Feedback is also collected from customers in retrospective meetings after the projects, and received directly from customers via e-mail and meetings. Feedback isn't automatically collected in any way.

Usage data collection. Usage data isn't collected at all.

Feedback and usage data usage. The received feedback is discussed within the team to improve the development process or certain parts of the product. The feedback isn't systematically used,.

5.3 Context

The previous chapter introduced the two teams and products under the scope of this thesis. The more general context of the whole company under study is defined by the following properties.

Company size. Steeri has approximately 80 employees at the writing of this thesis. The employees are located in two cities in Finland, and many of the employees are assigned to projects at other organizations.

Business area. The company in question focuses in managing, analysing and improving the usage of customer data. This includes various CRM-systems, such as Oracle Siebel, but the company also has its

own marketing automation and master data management products. The company also does data analytics within the Business Intelligence team.

Teams under study. As Steeri covers a wide business area, the scope for the thesis is narrowed down to two teams. The other team focuses on product development of a marketing automation product, while the other focuses on a master data management product. Both of the teams consist of developers, managers and Q&A dealing with the same products.

Team setup. The development team consists of a team leader and developers, while the management team consists of two product owners, commersialization manager and a requirement engineer.

Development practices. Currently the team in question uses continuous integration, with feature branches merged into mainline as soon as the feature is finished. This could differentiate from a couple of hours up to multiple days.

Development model. The development model is essentially a Lean model, the core being a prioritized Kanban board.

Tools used. The teams under study use Git as a version control system and GitHub code review tool, Trello as the Kanban board for task management, Jenkins as the continuous integration server and Bugzilla for issue tracking. Various other tools, such as the hour logging system, CRM system and various other ticket logging tools exist as well.

5.3.1 Development process

The development process of the case company has evolved from traditional, waterfall-style development through an agile phase to the current Lean approach based on a kanban board. In the very first phases, when the company was young, tasks were simply put into excel. Eventually, Agilefant backlog product management tool was introduced and tasks were moved into it. Scrum and agile practices were also adapted at the same time. Agilefant allowed sprint management, but caused too much overhead according to the developers. However, during the time Agilefant was used as the product and sprint backlogs, a part of Dialog development was offshored. Agilefant allowed easy tracking and managing the offshore development efforts.

After Agilefant was determined to cause too much overhead, Scrum whiteboard was used for a while. The physical whiteboard however caused issues with offshore participants. After the Development and Integration team was split further into two sub-teams focusing on each project, Scrum meetings were also seen as overhead. This was caused by only half of the

team being interested and attached to tasks considering a single product. After it was decided that the team is split into two sub-teams, practicing Scrum was again seen as an overhead due to the low amount of participants. Eventually the whiteboard was replaced with an online kanban board, Trello. By the same time the Scrum meetings were abandoned for a more streamlined process, that is able to respond to changing requirements faster.

Work items are first added to customer backlogs in iSteer Contact. In this phase they might be just initial skeletons and do not need to contain all needed information. The work items are added by product or business owners or project managers. At this phase the created backlog items are not yet visible in team backlogs in iSteer Contact. The idea is to list them in the customer backlog as early as possible and start refining the requirements collaboratively both offline and online with the help of tools such as Chatter, a platform for discussion. Chatter can be used to discuss a single story or the complete backlog.

When the backlog item is ready for the development team to start working on it, it should contain a descriptive name, specifications that explain both the technical side and the business side and an agilefant link for hour reporting.

Team backlog is a view to all stories assigned to one selected team. It is a tool to collect and organize stories from all customer/project backlogs without cloning details into multiple places. It presents a view of all team backlog stories that are not yet completed and are assigned to the selected team. Team backlog items are prioritized and estimated once a week, and new stories will be moved to Trello when they are accepted by the development team. Stories can be moved to Trello by project managers or product owners, but they must be checked by team leaders who then move them forward into the sprint backlog or prioritized list columns.

Overview. Trello is a project management application that uses Kanban to control the production chain from development to release. Kanban attempts to limit the work currently in progress by establishing an upper limit of tasks in the backlog, thus avoiding overloading of the team. Trello consists of multiple boards, each representing a project or a development team. A board consists of a list of columns, and each column consists of cards. Columns each contain a list of tasks, and cards progress from one column to the next when each task has been completed. A card is essentially a task, which is added by the Backlog Owner, and can be checked out by a developer. In Steeri, the columns used are Sprint Backlog, In Progress, Review, Ready, Verified and Done.

Sprint backlog. The Backlog Owner moves the cards that have the highest priority into the sprint backlog. The sprint backlog should always contain enough cards so that whenever a developer completes a task something new is available in the backlog.

In progress. The actual development work is done in this stage. The actor here is the developer. The checklist to move a card into review stage is:

- All tasks are complete
- Changes are pushed to a feature-specific branch
- Unit tests are written and pass in the CI server
- Feature is well-written and does not need refactoring
- Pull request is created and a link is added to the comment field
- Feature has been documented as needed

If the feature needs refactoring a task list must be created and the card moved back to In Progress column.

Review. Here other developers review the new code and deploy it to a development environment. Checklist for moving the card into ready stage is:

- Pull request is reviewed by at least two (2) persons
- Pull request is merged to the development branch
- Feature is deployed to a development environment
- Source code quality has to be good enough!

After a developer has reviewed the pull request, he leaves himself as an assignee. The second person who reviews the pull request is responsible for cleaning all the assignees and moving the feature to Ready column. If there is a major problem in the pull request the feature should be moved back to Sprint Backlog and the yellow "Boomerang" tag added. Person who created the pull request is responsible for implementing the necessary remarks. If only small fixes are needed, they should be implemented within the Github pull request workflow. The second person who has reviewed and accepted the pull request is responsible for deploying the feature to a development environment.

Ready. Here the product owner verifies the new functionality in the development environment. The card can be moved to Verified if the feature has been verified by the Product Owner in the development environment. Product owner is responsible for moving the feature to the Verified column. If verification for the feature fails, the Product Owner should move the feature back to Sprint Backlog column with the highest priority. In addition the Product Owner should add a yellow "Boomerang" label with a comment describing the results in the feature.

Verified. Here the backlog owner collects the timestamps and trello flow data. The timestamps depicts the duration it took from a card to process through the whole chain. The data is then used to analyze which columns the card spent the longest time in, and to identify the pain spots.

Done. This column simply states that the task has been completed, and should eventually be archived. There's currently no general validation required from the customer, as the customer projects each have a different schedule and process for builds.

Prioritized lists. The backlog owner adds tasks to prioritized lists from team backlog as soon as the tasks meet the required criterias, contain the required information and are inspected by both the stakeholders and the backlog owner.

5.4 Methods

The primary source of information in this research are semi-structured interviews [42] performed within the CDM and Dialog teams. The interview consists of pre-defined themes as follows: (1) current development process, (2) current deployment process, (3) current interaction with customers, (4) software product and (5) future ways with continuous delivery and continuous experimentation. Data is also collected through the product description documents and development process documents to verifying and supplementing the interview data. Data triangulation is implemented by interviewing multiple individuals in different positions of the teams. Methodological triangulation is implemented by collecting documentary data regarding the development process from internal documents, and by including observations made by the author in the analysis.

The interview was chosen as a data collection method because of the nature of the research questions. Since the study focuses on applying two development methods to the development process of a team, individual perceptions of members are required. As the case study doesn't include a technical implementation, quantitative measurements to properly measure the effects before and after implementation isn't an option. The research questions also cannot be properly supported with quantitative data, as they are exploratory in nature. There is also an uncertainty about how much information the interviewees are able to provide, and thus the questions are mostly open-ended. The nature of interviewees opinions on the research questions are also not known in advance, and quantifying it isn't simple.

In order to answer the research questions, information regarding the development process, customer interaction and feedback, deployment process and technical product details are required. The interview is divided into themes to address these aspects. The interview questions address, among

other things, the specific situations and action sequences of the interviewee rather than general opinions.

5.4.1 Data collection

The interview is a semi-structured interview with a standardized set of open-ended questions, which allows deep exploration of studied objects [42]. The interviews are performed once with every interviewee. There are a total of 12 interviewees: 6 in each team. The interviewees in the CDM team consist of 5 software designers and one team leader. In Dialog team, the interviewees consist of 3 software designers, a quality assurance engineer, a manager for commercialization and a team leader. Leading questions are avoided on purpose, and different probing techniques such as "What?"-questions are used. The interviews are performed in the native language of the interviewee if possible, otherwise in English, and are recorded in audio format. The audio files are then transcribed into text.

The interview begins with a set of background questions, used in coding the interview data per subject. After the introductory questions, the main interview questions make up most of the interview. The interview session is structured based on areas under study rather than based on a specific model. The interview roughly consists of five different areas: development process, deployment process, customer interaction, feedback and usage data, and experimentation.

The development process questions aim to identify the current ways of working, including variances between different roles. The questions also identify the perceived problems and strengths in the development process, which are then used in analysing the potential process for continuous experimentation.

The deployment process is examined from multiple point of views. This includes the customer point and view, developer point of view and management point of view. Emphasis is based on B2B specific issues and practices, which are often caused by differences in the software environment as compared to B2C.

The customer interaction section analyses customer involvement in the development and deployment process. The section identifies where the customer's presence is high, what is the role of the customer in the operational level and how the role of the customer affects the transition towards continuous delivery and continuous experimentation.

Feedback and usage data collection section analyses the procedure of collecting and utilising the data. The section aims to identify the manners in which feedback is received, collected and systematically used. The section also analyses how different roles interact with the customer.

The experimentation section analyses the possibilities of conducting experiments to guide the development process in the case company. This includes

technical details, such as instrumenting the software components with data collection mechanisms and external usage data collection components, and storing the data in customer environment. The section also includes more specific questions, such as possible challenges faced in A/B testing and issues in quickly deploying experiments to the customer environment.

Interview data is primarily sought from the developers of the development teams and the managers of the team. As the focus of the thesis is on the development process of two different teams, all participants of the team and its management team are interviewed. Process data is sought from the process documents made by the team. Quantitative data, such as the Trello ticket flow time, is sought from Trello. Data regarding the development process is also collected from the internal documents.

During the interviews, it was soon detected that the interviewees are able to give accurate answers to certain contexts based on their positions. This eventually contributed to the template analysis, where the data was better organised based on roles. Developers are most aware of the product and project context, including technical details and project details. Team leaders are mostly aware of the process and project context. They are the persons who are most able to explain the development process in detail. The team leaders are also most proficient in feedback collection. The management is most aware of the company context, including the status of the organization and available resources in the company.

To summarize, interviews were chosen as the main data collection method based on three criteria. Firstly, the nature of the research questions required insight on the company settings and opinions of the employees. Secondly, quantitative data cannot be used to answer to the research questions, as first initial understanding of the phenomena has to be formed. Thirdly, the analysis is divided into themes based on the software products, and qualitative data provided by interviews can be readily analysed according to themes.

5.4.2 Data analysis

In this case study the data analysis is based on template analysis, which is a way of thematically analysing qualitative data [22]. The initial template was first formed by exploring the qualitative data for three themes: development process, deployment of software and experimentation. Through multiple iterations of the data, multiple subthemes were then added to the three existing themes by further coding the data. Attention was paid to different roles of the interviewees.

Prior to the template analysis the data was inserted into a spreadsheet, where the rows represent codes of interest and columns represent interview subjects. This eased the analysis of data. The final template was formed after three iterations, which each increased the depth of the template by one

or two levels, and the amount of codes by several.

Template analysis was chosen as an analysis approach because it makes it easy to organise and analyse the data according to themes. It is also particularly effective when comparing the perspectives of different participant groups [24]. Since the thesis focuses on multiple themes and the interview on multiple groups, template analysis was seen as a suitable approach.

The results are formed from the final template, and structured according to themes of the research questions to answer the research questions as comphrehensive as possible. The results are then validated from the original interview data, so that none of the sections left out of the template are in conflict with the findings.

6 Findings

This section summarises key findings of the case study. It is structured according to the research questions, allowing the progress from a higher-level perspective to more detailed view. First, the identified challenges in continuous delivery and continuous experimentation are listed and explained. Then, the benefits of applying continuous delivery and continuous experimentation are analyzed based on the current state of the case company. Finally, a continuous experimentation model by Fagerholm et al. [16] is mapped to the case company context, and deviations identified.

6.1 Continuous Delivery: B2B challenges

Software development practices and product characteristics vary based on the domain and delivery model. Typical B2C applications are hosted as SaaS applications, and accessed by users via a web browser. In the B2B domain, applications installed to customer environments are very common. This section analyzes the problems faced with applications that are installed and operated in customer environments when transitioning towards continuous delivery.

The challenges regarding continuous delivery will be analyzed in three areas: technical, procedural and customer. Technical aspect includes the environmental challenges, configural challenges and other challenges related to the software product and its usage. Procedural aspect includes the challenges regarding the development process of software. Customer aspect consists of the customer interaction and customer commitment.

6.1.1 Technical challenges

The technical challenges for continuous delivery are derived from the interview. A part of the interview focused on the current deployment process and

customer interaction of the case company, and this process is then compared to the continuous delivery practice from literature review.

Specific problem
Downtime is critical for certain customers
Automated testing has to be built on top of a matured software product
Software is often integrated to multiple third party applications
Software is often accompanied by multiple external components
There exists multiple different configurations due to having multiple customers with different specifications
Transferring the software product to diverse customer-owned environments requires different deployment configurations

Table 3: Technical challenges in continuous delivery.

Downtime of the case company's products can be fatal. According to the Dialog product owner, downtime causes end-users being unable to perform their job. Downtime can also interrupt ongoing customer tasks, possibly losing critical data in the progress. With CDM, downtime can cause other integrated systems to shut down due to too many failing requests. Currently the deployment time for both projects is negotiated with the customer to prevent these cases, and the version deployments are done when the system can be closed for a short period of time. In the continuous delivery process the deployment pipeline has to be designed to address the issue of downtime.

The developers perceive automated testing and test environments to be the largest technical task. Since both of the case company's software products have been in development for a long time, building a viable test automation has many challenges with architectural decisions and available workload. The developers state that building a sufficient test automation is a very laborious process due to the maturity of the software, and are concerned with the maintainability of the test suite. The management is not sure what to test with automatic acceptance testing to validate that a version is valid. Due to the current amount of time spent on manual testing, having an automated test suite is seen as very beneficial.

Both of the case company's software products are integrated to various third party applications and APIs. As CDM is a component that integrates various data sources together, changes to the APIs communicating with these applications must be planned and discussed in advance. Based on the interview results, automatically updating the integrations requires an unduly amount of work considering the results.

It is also common for B2B applications to have external components that have to be configured when the software is installed or the APIs to

these components changed. Therefore the configurations for these external components either have to be manually updated or automated as well. In the case company, both Dialog and CDM are integrated to multiple external components. Currently upon version change these integrations are manually updated when necessary. In continuous delivery, changes that modify the API and therefore break these integrations might have to be manually updated.

Both of the case company's products are used in multiple different customer environments. This introduces a problem of managing different configurations per customer environment and software instance. Multiple possible solutions exist for the configurations, however, such as creating custom profiles per customer with different configuration for each profile. In the case company the challenge is how to automatically determine which customer profile should be used for each environment.

One of the main differences between B2B and B2C is the production environment. While a company focusing on B2C often owns the product environment, in B2B domain it is common to install the software on customers environment. Along with the varying customer specific configurations, there exists varying customer environment specific configurations. The customer owned environment introduces multiple challenges: accessing the environment, transferring the software product to the environment and integrating the software to other components within the environment. These challenges related to transferring the software include firewall configurations, required certifications and user rights. The customers also have to be informed on actions performed in their servers. For certain environments, a VPN connections is required. A VPN connection often has a user limit, which can prevent automatic version deployment.

6.1.2 Procedural challenges

The procedural challenges are analyzed based on the development process documentations of the company and the interview, in which a section was dedicated to the current development process of the case company.

The basic deployment pipeline in the case company first includes a deploy to a user acceptance testing server, which is then tested manually by either the team or the customer. Only after the version has been acceptance tested and validated to work properly, can the production version be released. Based on the interview results, continuous deployment to production is seen very risky due to the applications playing a major role in running the customers business, and continuous delivery with deployments to a staging environment is seen as very beneficial. Therefore the deployment procedure has to be designed to allow testing the software in a user acceptance testing

Specific problem
User acceptance testing environment is a requisite for production release
The development process drifts towards small feature branches from long-lived feature branches
Triggering the compilation and deployment of a modular project to maintain integrity is hard
The software has to be deployed to multiple customers
Versioning is affected by having different customer profiles of the product
Responsibility of deploying moves towards developers
Management and sales loses track of versions

Table 4: Procedural challenges in continuous delivery.

environment, and making it possible to manually trigger the deployment to production.

Both of the case company's products are developed with a branching model, where feature branches are first thoroughly developed and then integrated to the master branch. Only the master branch is ever released to the customer. However, with continuous delivery the long-lived feature branches should be changed to short-lived and relatively small feature branches to enable faster deployment of the software. The benefit of smaller branches is that while there may still be bugs, they're introduced in a fewer number of commits, and entire large features don't need to be rollbacked. Smaller branches also enable faster feedback. While the small feature branches might be common for companies with a relatively new software products, companies that have been developing products for a long time might be more devoted to the practice of feature branches. However, it is still possible to keep the software releasable with long-lived feature branches by hiding new functionality or by constructing it as a series of releasable small changes [20].

The software applications in B2B often are large and modular applications, as is the case in the case company. CDM is a software component that is compiled from multiple projects. As the deployment process is currently manually triggered by first releasing a version, a suitable time can be chosen each time. With continuous delivery the deployment process is triggered automatically, and with a modular project each module has to be in the correct state in order to produce a coherent version. The point when a deployment is triggered has to be designed to maintain the integrity of the application.

> "Pull requests aren't always working as is, due to the nature of CDM. There are dependencies between projects, and occasionally merges are done without the will to deploy. However, if pull requests are to be deployed when merged, a better picture of this process is needed for the developers." (CDM Developer)

In the case company, merging a pull request is not a valid trigger of a deployment, since pull requests are project specific. Some features require changes to multiple projects, and all of the pull requests have to be merged for the feature to work as intended. However, a good CI tool will handle the dependency management [20].

Both of the case company's products are used by multiple customers, each having their own environment. As the deployments are currently done manually, the customers receiving each deployment can be manually chosen.However, with a continuous delivery process whenever a feature or a new release is ready to be delivered, it can either be deployed to a single customer or to every customer. Attention has to be paid when configuring the workflow to allow deploying to an arbitrary customer. If only certain customers accept updates on a continuous basis, it has to be possible to prohibit deploying to every customer at once. This is the case especially in the early stages, where all of the customers projects aren't yet in the state of continuous delivery.

Multiple customer environments affects versioning of the software product. In the case company, each customer has a unique configuration of the product, with possibly different versions of certain components. According to Jan Bosch, in an IES environment only a single version exists: the currently deployed one. Other versions are retired and play no role [7]. However, with multiple environments, multiple different versions of the software are necessary at least in the early phase. Eventually the product may be developed in such manner that a single version is sufficient. The build script therefore has to be configured to consider the customer-specific components when compiling the builds for each customer.

Continuous delivery also drifts response towards the developer, and the developers decide what is ready to be released. Currently in the case company the product owners and team leaders are responsible for negotiating the deployment date with the customer, and they also inform the developers that a new version is required. If the developer can single-handedly deploy a feature, the management can quickly lose track on the features available to customers. This also requires the developers to deeply understand the details of the version control system and automated testing.

Due to increased developer responsibility and varying interval of version updates continuous delivery causes, a team leader express concern that the delivery process complicates tracking when deployments are performed, and when features are finished. This also concerns other parties working in the customer interface, such as sales. While this is a concern for version deployments to the UAT environment, especially if developers perform these deployments at will, can production releases still be negotiated with the customer. Neely and Stolt have also found similar results [34], which they solved by providing mechanisms by which the parties if need of knowledge regarding release times could track the work in real time.

6.1.3 Customer challenges

The customer challenges are analyzed based on two sections of the interview: customer interaction and the deployment process.

Specific problem
Some customers are reluctant towards new versions
Customers are trained to use a certain version, and modifications confuse the users
Changelogs are especially important, since as versions are released faster the customers become less aware on what has changed
Pilot customer is required for developing the continuous delivery process
Acceptance testing is performed by both the company and the customers, and requires a lot of resources from the customers
Production deployment schedule has to be negotiated with the customer
Ongoing critical tasks by users cannot be interrupted by downtime

Table 5: Customer challenges in continuous delivery.

One of the main issues found in the deployment of software is that some customers are reluctant towards new releases. One of the reasons for this reluctancy is that new releases occasionally contain new bugs. In the case company, customers using Dialog have been trained to perform certain tasks with a certain user interface. The customer might perform these tasks daily, once every two weeks or even less frequently. If the UI changes often, the customers feel lost and initially take more time to perform the tasks. This causes frustration in the users, and visible changes generally increases the reluctancy customers have towrads new versions, unless the changes are significantly improving the user experience.

> "The user interface should be easy to use. Now it's relatively hard to learn. If customers have just learned to perform a task, and we change the UI, the feedback is terrible."
>
> (Dialog product owner)

While the user experience could be improved in the long run, the customers might feel frustrated when they have to learn to work with the modifications. The interviewees have ideas on how the reluctancy can be reduced. If modifications were made more often, it leads to smaller and less notable modifications which might reduce the customer reluctancy towards new versions. When customers become accustomed to the model of continuous version updates, the reluctant attitude might change. Customers also need to understand the concept of continuous delivery, and that the features

are eventually implemented to address customer needs and to improve the user experience in the long term.

> "If something changes, communication with the customer is hard. The customers feel that even if the version included very good changes, if something visible changes it gets alot of negative feedback".
>
> (Dialog product owner)

The customers are unaware of the problems related to deployment issues unless the deployment has errors. However, if the version contains many UI changes the customer might feel frustrated by being unable to perform certain actions in the old way:

> "Customers have been doing this for their living - they're getting old, they have their own working histories and they're in the working mode. These persons don't approve alterations in the software, and they're not ready for the constant changes."
>
> (Dialog product owner)

Listing the changed features in changelog entries is especially important when releases are made more often. While the changes become smaller the faster versions are released, the customers become less aware of when the version will be updated and when features have changed. Currently the version deployments are negotiated with the customers, and necessary actions are taken to explain the changed features to the customers in meetings or in e-mails. These actions can include changelogs, but also training sessions or meetings. When the deployments are made more often, discussions regarding version releases may be reduced or even ceased.

A way to identify the best practices in continuous delivery is to develop the continuous delivery process with a pilot customer. Pilot customer is a company willing to help the company to quickly learn what works and what needs to be improved. The interviewees expressed a desire to first test the continuous delivery process with a single customer that is willing to receive updates in a continuous manner, since the engagement model inevitably differs from the current model. Therefore the company can train customer interaction with continuous delivery, find the actual reasons for version reluctancy, see how the change in deployment process affects customer interaction and even test if the UAT environment is necessary.

The acceptance testing is performed in varying ways. For Dialog, some customers require to perform manual acceptance testing in a user acceptance testing environment before the product can be deployed into production. Other customers trust the developers to perform the acceptance testing, and the deployment to production can be negotiated without it necessarily requiring to pass the user acceptance testing phase. In CDM, all of the customers perform manual acceptance testing in the user acceptance testing

environment. The technical implementation therefore should make it possible to continuously deploy versions to the user acceptance testing environment, and by the push of a button to the production environment. However, if the versions are deployed to user acceptance testing environment very often, customers might feel encumbered by the amount of required testing. The customers also have to be informed whenever a new version is available to the user acceptance testing environment.

Customers might be using the software when a new version is deployed, and the deployment process shouldn't interfere with ongoing usage. Therefore the most viable solution is to initially deploy the versions to the UAT environment. To prevent downtime and obstructing the usage of the software, the production deployment date should be negotiated with the customer. Production deployment might cause downtime, and cannot be made during times when critical tasks are performed. The test environments can be continuously updated, and once the customer is satisfied with the software and feels a need for a new version, deployment to production can be triggered.

In the case company, the production deployment date is often negotiated with and specified by the customer, while the product can be deployed to UAT at will. UAT is also used by the customer to validate the functionality of the software, since the customer has ordered the product to perform certain tasks and fill certain criterias. However, the customers don't validate the versions at the UAT environment unless they are informed that new versions are available, since the UAT is not used for any other purpose than testing.

6.2 Continuous Experimentation: B2B challenges

The case company's two software products differ on how they are used by the customers. In this section, continuous experimentation challenges are analyzed for general cases encountered in both products, and special cases encountered in a single product due to a specific product characteristic.

The challenges regarding continuous experimentation are analyzed in four areas: technical, organizational, customer and A/B-testing. A/B testing was chosen as an examined subject of its own, since it is a commonly known experiment, and the difference between the case company's products affect especially A/B testing.

Technical aspect includes the data collection, measurable metrics and other challenges related to the software product and its usage. The organizational aspect includes challenges related to the organizational culture, financing and ways of working. The customer aspect includes the challenges regarding the experiments in the customer interaction process and from customers point of view. A/B testing section analyses the viability of guiding the development process of both products with A/B tests.

6.2.1 Technical challenges

The technical challenges are analyzed based on both the product description documents and sections of the interview focusing on experimentation and feedback.

Specific problem
Not having a user inteface limits the scope of experiments
Low end-user volume limits experiments that rely on statistical significance
Product that is only usable via API calls has a limited scope of experiments
Measurable metrics of the software products that provide value for the customer are not extensively identified
Experimentation infrastructure has to be implemented on top of a matured project
Data has to be collected and stored in customer environments, and transferred from the environment for analysis

Table 6: Technical challenges in continuous experimentation.

The case company's products represent two different styles of software products. While Dialog is used by users via a UI, CDM is used by other systems via a REST API. As CDM is a software component which is primarily used by other systems via API's and certain calls, tracking user behavior doesn't provide as much input as with a user interface. CDM does not have a UI, which limits the possibilities for A/B testing. While it is possible to experiment with a software component that has a UI and human users by employing A/B testing, other experiments have to be conducted on software without direct user access. Such experiments include feature alpha, usage metrics, usage behavior tracking and participatory design, among others [7]. The CDM developers considered usage metrics to be a valid experiment, and were concerned that usage behavior tracking might not provide appealing results with no human users.

The lower user amount also affects the types of suitable experiments. Dialog has only approximately five users per customer, while CDM has no human users at all. Experiments relying on statistical significance, such as A/B testing and multi-variate testing, lose most of their value with a low user volume. To be able to confirm validations with such experiments require a varying amount of users depending on the conversion rate of the experiment. The company then either has to work with a lower confidence interval, or have a very high conversion rate to reduce the amount of subjects needed

for statistical importance. However, many experiments rely on tracking and improving the usage behavior and not relying purely on statistics. A product owner views experiments relying on tracking the user actions suitable for developing Dialog:

> "The amount of statistical data isn't much (due to the user amount), but the users actions can be tracked, and they continuously perform repetitive tasks. This customer would then provide a lot of usage data. If the time spent by this user is halved, could the customer make more money. Experimentation doesn't therefore necessarily require a large user amount."
>
> (Dialog product owner)

The business model for the products affects the desired improved metrics. Since the purpose of the software components is not generate revenue per se, knowledge of how the customers benefit from the product is required before the metrics can be chosen. According to the interview results, the interviewees were only able to name a few metrics that they perceived as desired improved metrics. In B2B the software development is especially focused on adding value for the customer, but eventually the features must also add value for the software product itself. In the case company the developers are not actively participating in customer interaction, and are therefore not as aware of the customers challenges are the team leaders and product owners.

The products also have custom solutions for different customers, which means a measurable property might only apply to a single customer. This makes it somewhat harder to pick the correct measurement variables. The most popular metric in the interviews is the customer opinion of the product:

> "We don't aim for increase in clicks or in purchases. Instead, if we wanted to measure how Dialog could be improved, the most useful metric would be the customer opinion. A measure could also be how fast certain work tasks can be performed, but it's hard to measure it. The customer doesn't necessarily perform the task as fast as possible."
>
> (Dialog developer)

A developer also presented the possibility to tune the product in cooperation with the customer, using their internal measures as metrics:

> "If the customers had their own metrics that they have been tracking, and we could affect this trend, we could figure out whether this is caused by changes in the software. Many customers monitor their employers and measure them."
>
> (Dialog developer)

This approach is quite similar to the participatory design and collaborative innovation [7] practices. A CDM developer also has an idea of surveying the customers daily routine as a basis for building experimental features:

> "We should always think what the customers status is now, what kind of features the customer is using and what is the system that CDM replaces. We can develop the experimental features against these things. In the old system the customer might have done things that they they felt was inconvenient. We could conduct the experiments customer wise by surveying the customers daily routine and build experimental features based on this." (CDM developer)

It is common in B2C applications to fluctuate between different Overall Evaluation Criterias (OEC), conducting multiple experiments with different metrics. In B2B domain the possibility to co-operate with the customer in order to increase or identify the possible metrics should be considered. Eventually, the OEC has to be well thought. As an example, pure profit is not a good OEC, since raising price can increase a short-term profit but have a negative impact in the long run [27]. A clear difference exists between metrics that are not measurable and not-economical to measure [27]. The former is a bad way to run a business, since without being able to articulate what the purpose of the experiment is, it is impossible by the organization to determine whether the experiment is succesful [27]. Also, a common pitfall in conducting experiments is to pick an OEC "for which it is easy to beat the control by doing something clearly “wrong” from a business perspective" [9].

The case company's software products have been developed for over three years each, and are becoming quite large and mature. To be able to collect customer or performance data, and to determine customer preference and interest, architectural changes of the product are required. One of the most important components is the instrumentation layer that is used to collect the data of the experiments, and it can be implemented in multiple ways. Architectural changes are however more costly for matured software, which already have a structure and many architectural dependencies. However, the interviewees view the current product architecture as very modular, with the possibility of easily adding new modules for data collection.

The case company's products are mostly installed to customer environments, with customer-owned servers. The data then has to be effectively stored and acquired from an external environment, if the analysis is to be done at the company's own servers. However, the case company's applications deal with millions of records of data, and the developers don't see storing additional experimentation-related data as a challenge:

> "The data would be on the customer companies servers, where we have our own access and right to install software. If the data is collected on the server where we fetch it from, it's not a challenge. The data shouldn't encumber the system." (CDM developer)

6.2.2 Organizational challenges

The organizational challenges are analyzed based on the data collected from the experimentation section of the interview.

Specific problem
Requirements by customers bring in revenue, and are prioritized over own product development
Competence in experimentation is not high, and requires education
Transitioning towards experimentation culture affects other teams, including sales
Creating a culture of innovation within the organization

Table 7: Organizational challenges in continuous experimentation

According to the interviews, transitioning towards experimental R&D is perceived as desirable but expensive. Product development has been in recession with the accelerating customer amount and increasing customer requirements. The teams under inspection have no prior experience with experimenting or data analysis, and require training in order to gain domain knowledge of experimentation.

> "Currently, the product is developed based on what the customers want. We would need more resources [to guide the development with experiments]."
>
> (Dialog quality assurance engineer)

The interviewees are also concerned about organizational divergence, especially if continuous experimentation is only adapted in two teams. Both continuous delivery and continuous experimentation affect the customer interaction process, and other teams interacting with the case teams need to understand the experimentation concept.

The teams under study have been working with agile and lean practices for a long time, with minimal focus on innovation. Adopting continuous experimentation shifts the mindset more towards innovation and emphasizes the importance of deeply learning the customers needs. Systematically applying experiments also changes the way software products are developed. New features and product ideas are developed with keeping the possibility of confirming validations with experimentations in mind from the very early stage. This results in more measurement, tuning small details and especially making decisions based on data. The interviewees expressed an interest to shift other parts of the organization towards innovational mindset, especially the product management and sales teams, since they are working in the interface to the customer.

6.2.3 Customer challenges

The customer challenges are analyzed based on three sections of the interview: customer interaction, feedback collection and experimentation.

Specific problem
Large part of developed features are requirements from customers
Customers perform routine tasks that cannot be interrupted
Customers have to be informed of major changes
Technical personnel aren't involved in customer interaction
The software has to be acceptance tested by the customers before production release
The end-users might be customers' customers, complicating feedback collection
A pro-active lead customer is required to develop the experimentation process
Proper legal agreements need to be made with the customers to collect usage data and user behavior

Table 8: Customer challenges in continuous experimentation.

In the case company, features have been primarily created for actual customer requirements. This somewhat limits the direction of product development. In the case company, recently almost 100% of the completed features have been determined by the customer. A product owner views the current situation as temporary:

> "The goal ratio [of own product development against the development of custom features] would be 50-50. Half would be our vision of how to perform business, the rest requirements from customers. In the current situation the ratio is 100-0, and for over a year every feature has been the result of a customers needs regarding Dialog."
>
> (Dialog product owner)

The product roadmap has advanced by being led by the customer projects, since the revenue comes from the customers. The actual customer requirements are a large part of the product development stage, and customer projects often consists of only features required by the customer. As the product development is in the ideal case guided by both the customer needs and own ideas, the proportion of data-driven decisions is smaller than in B2C. However, the interviewees state that as the software is productized, the amont of customer specific work decreases, eventually increasing the importance of data-driven decisions.

Another issue in the customer projects is downtime. Since the customers are using the software during normal work hours, critical tasks cannot be interrupted by version changes. Currently the time slow for version deployment is negotiated with the customer, and the downtime is positioned during times that the software isn't used. A simple solution for this is to have the test environment continuous updated, while the production environment update is negotiated with the customer. However, the test environment isn't always actively used. This is what a developer has to say on the issue:

> "Usually the customers test environment isn't in heavy use, and it might take for the customer to react to changes. In our own test server we get a lot faster feedback. The customer has also be informed if we perform major changes, since the customer needs to know how to use the software." (Dialog developer)

Therefore continuously updating the test environment either has to alarm customers that new features are to be tested, or production environments has to be fixed in such manner that continuous updates don't cause downtime.

The customers have been trained to perform certain tasks with the software, and if the UI constantly undergoes major changes the customers might get lost. There is already some reluctancy towards now versions, since new versions possibly introduce new bugs and certain UI elements might have their places changed. Therefore customers have to be informed for major changes, so that possible new training sessions can be arranged. For minor updates, release notes and required guides are a must. While certain types of experiments can be performed without the consent of the customer, experiments that degrade the user experience might need to be informed to the customer. As the customers acknowledge that the product they have bought is being developed to better suit their needs, approval of the experiments is more likely. However, publicizing the experiments can affect the experiments outcome, and should be handled with care.

The case company's customer interaction is mainly done by the team leaders and product owners. This limits the developers knowledge of the customers needs, and the parties with enough knowledge to design experiments are the personnel continuously interacting with the customer. In the case of a B2B company, the customers problems can be complex and require deep knowledge of the customer domain. Eventually, when developers have participated in experimentation, the knowledge is distributed to all participants.

The case company's products are mostly installed to customer environments, with customer-owned servers. The typical deployment flow includes a customer testing phase in the user acceptance testing environment before a production version can be deployed. The UAT environment is not actively used, and the customers perform UAT only when they are informed that a new version is available. The production environment is seen critical enough

that no major bugs are allowed, and thus the UAT testing cycle is seen favorable by both the customer and the company. Performing experiments only in the UAT environment reduces the user amount to very low, and prevents actual end-users being subjected to the experiments. Entirely skipping the UAT environment is seen as very risky by the product owners and developers. However, the benefit of the UAT environment is that the experiments can also be tested. The cost of the UAT environment is that the experiments cannot be tested as quickly. If the experiments are publicized, it can bias the users and affect or even spoil the experiments outcome.

The customer feedback is currently continuously received but not actively collected. Product owners are in constant connection with the customers, and customers can send comments and requests via them. The customers can also insert tickets into a bug tracker. Most of the received feedback is about some functionaly not working or a bug. Weekly status meetings with the customer are also arranged. Feedback is received via e-mail as well. In the end of the project, a retrospective meet is hel with the customer, where the whole project is discussed.

The feedback collection in B2B has multiple challenges. In the case of customer companies, feedback doesn't always come straight from the end user. Feedback relies on what the other companies representatives say, and this often includes their own opinions. Companies want the software for their own use, and they have requirements for property rights, security and others. In Dialog, the customer company's representatives perform marketing operations with the software. Also their customers can be considered as users, since they are the subjects of the marketing operations, and interact with the software by filling web forms. This adds to the legal obstacles, and customer's consent for user behavior is required to collect data from their users to improve the customers business. Additionally, in case of Dialog, the user is not the payer of the software. The user has to use the software even if it was bad, which makes it hard to get a neutral point of view.

The interviewees feel that having a pro-active lead customer is required to both design the experimentation process and to practice it with the customer. The interviewees are unsure how experimenting affects the engagement model with the customer, and what is required from the customer. The interviewees also wish to research if the customers have their own metrics related to the use of the software, and whether these can be improved by experiments. More information related to the customer usage of the software and challenges face with the software is also required from the customer, and a customer willing to co-operate is seen as very beneficial.

According to the interview results, the product owners state that to collect usage data and user behavior from customers, proper agreements might need to be made. Some of the customer companies have strict rules for which data can be collected. Consent from the customers is therefore required to collect data related to the user actions.

> "We need a permission from the customer to run experiments. The customers conduct mostly routine actions. The software has to be stable, so that the customers know where everything is."
>
> (Dialog developer)

However, the interviewees are positive that the customers are willing to allow the data collection, since the product has already been bought. The interviewees express that as the purpose of data collection is to develop the software in a way that adds value for the customer, it is easier to justify the need to collect data.

> "The customers are motivated that we are trying to improve their actions. Our customers probably let us track their actions, because they have already bought the product."
>
> (Dialog product owner)

6.2.4 A/B testing

The A/B testing is analyzed based on the product description documents and two sections of the interview: feedback collection and experimentation.

Specific problem
Low end-user volume limits the scopes of experiments
CDM is used by integrated systems instead of human users
CDM has user no interface to tune
Measurable metrics for A/B testing are not extensively identified
Users have been educated to use certain features, and changing these confuses the users

Table 9: A/B testing challenges.

Both of the case company's products have a low user amount. Dialog has from approximately five users per customer, while CDM has zero human users. The lower user amount complicates statistical analysis. Kohavi et al. [30] state that a sufficient user amount is a requirement for controlled experiments on the web: "Very small sites or products with no customer base cannot use experimentation, but such sites typically have a key idea to implement and they need quick feedback after going live. Because new sites are aiming for big improvements, the number of users needed to detect the desired effects can be relatively small (e.g., thousands of users). Large sites, which are typically better optimized, benefit even from small improvements and therefore need many customers to increase their experiment's sensitivity level". The lower a sample size, harder it is to draw meaningful conclusions

from it. For a company operating in business with low user amounts, due to either new market or the type of the product, attention has to be paid on when statistical significance is reached. The company either has to work with a lower confidence interval, or majorly increase the conversion rate to decrease the amount of subjects needed for statistical importance. If the user base increases in the future, A/B testing becomes a possibility especially for the wide UI of Dialog. The interviewees state that A/B testing could be conducted by having different companies act as control and treatment. However, different companies might have different use cases.

The main difference between the case company's software products is the intended use of the software. CDM, being an integrated system, doesn't have a UI unlike Dialog. The types of A/B tests that are viable to CDM are mostly related to performance, which is seen as the only viable metric by the developers. Along with low user amount, the issue with Dialog is that the customers have been taught to use the UI to perform certain tasks. The UI of Dialog is quite complicated, and some customers are very reluctant to changes to the UI. A possible solution for this issue is a detailed use guide with each new version, where the changes are easily readable.

In A/B testing, the measurable metrics greatly vary between the case company's products. In CDM, the team finds testing the query machine beneath the API to be the most beneficial. These kind of A/B tests would be performed in the back-end, dividing the users to different queries or databases. Performance metric could then be optimized based on the result. Another possibility found from the interview results is offering multiple query features, and then tracking which one is used more by the customer. However, as the customers clients are hard-coded, the users would also have to modify their systems accordingly if new queries were added. Other essential metrics were not identified, and identifying these metrics requires extensive planning from the team.

In Dialog, the need for data-driven decisions is greater than that of CDM due to user experience playing a larger role. As Dialog doesn't generate profit per-se, the metrics that the team members perceive as important are more concerned with usability and user experience. While the interviewees haven't identified many metrics related to the software products that should be improved, they were able to name a few. These are time spent on performing certain tasks, amount of errors encountered in certain features and amount of back-selections indicating mistakes made by the user. Two of these metrics are related to the user interface of Dialog. By optimizing the time spent on performing tasks, the users are able to gain more benefit from Dialog by being able to perform more sales and marketing actions in a shorter time window. When the company starts conducting experiments to guide the product development, the real value for the customer and metrics related to the value must be identified.

Some features in Dialog are specified to work exactly as is, and the customers have been educated to use them in training sessions. Changing these features can cause the customers to get confused and frustrated due to not initially knowing how to use the features. By keeping the changes small enough, the frustration can be reduced. Teaching both the control and treatment to use different versions can cause the get publicized and thus bias the experiments outcome.

6.3 Continuous Delivery's benefits for the case company

A major problem found in the interviews was that currently the reliability towards new versions is low. The low reliability both increases customers reluctancy towards version updates, and increases the amount of user access testing that is performed after version release. This makes the duration of building a new version relatively long and unstable, since the bug amounts differ upon each version deployment. The version releases are also often performed in a hurry, which also increases the error rate. The low reliability is caused by multiple reasons: features are occasionally forgotten and not tested, manual builds introduce human error factor, the knowledge of manual steps required for a succesful build is not shared and compiling modular products is hard.

Versions are occasionally forgotten from the UAT phase due to the lack of comprehensive automated testing and the broad scale of features in both software products. These features can then remain broken or contain bugs when the users start using the new version. Even though these features get fixed relatively fast, the introduced bugs both increase the customer reluctancy towards new versions, and decrease the reliability of both the team and the customer. This is fundamentally caused by the lack of quality assurance before the release. Adopting a test automation solves this issue, as long as tests are written for every feature.

The manual build process involves human error factor, which in the case company is the primary cause for the majority of bugs in new versions. One example of such an error is using the wrong commands to compile and package the project, which causes wrong binaries getting into the new version. Combined with the lack of comprehensive automated testing, each deployment is a new error-prone experiment. Important factors increasing the rate of human error are partially missing documentation and the complexity of the build process. According to a developer:

> "There are many components that have to be configured manually. The documents for how to execute these manual actions exist, but many things still rely on memory."
>
> (CDM developer)

While some steps of the deployment process are documented, there are

still parts that are memorized by developers. Since some of the manual steps are only memorized by certain developers, the deployment requires assistance from these persons. With continuous delivery only a handful of developers might have knowledge of the entire build deployment configuration, but everyone are able to trigger the deployment process. Automating the entire deployment process also reduces the human error factor and necessity for documentation.

The interviewees perceive compiling and building CDM a tricky task. CDM is a modular product compiled out of many individual packages, which each have a unique version. Whenever a developer updates a version, attention has to be paid that the newest version of the package is present in the compiled application. By manually triggering the build progress, different flags can be used to define if a package is loaded from the artifactory or local environment. If the versions of these two environments differ, the product might get built with an incoherent composition of packages. With continuous delivery a good CI tool will handle the dependency management [20].

With the current deployment model, release cycle length ranges from two to four weeks. The feedback cycle is therefore quite long, and non-critical bugs reported by customers might take until the next release to fix. By adopting continuous delivery, feedback can be received a lot faster from the customers. Implementing the automated tests also adds a layer of feedback. The developers state that they would be more motivated if code could be deployed faster. Deploying the code makes the developers feel like they have achieved something concrete. However, developing the software so that it is always ready for release also improves the developers confidence towards the created code and product.

Automating the deployment also reduces the time required for deploying a version. The product owners state that building and deploying a version can take up to a day, which is seen as way too long. The primary reason taking up the time is caused by having to manually compile, test and bugfix the versions. With continuous delivery, the amount of manual work can be diminished to couple of button clicks defining which customer profile to deploy. Continuous delivery therefore allows the release of changes in time spans of hours, also allowing the most important features and fixes to be delivered faster. With faster problem fixing due to increases in bug discover and fixing rate, the software eventually ends up with fewer bugs.

Along with the deployment, environment installations take relatively long with manual installation of each component. With continuous delivery, starting the application in a new environment is ideally just generating configuration information describing the environment's unique properties, and starting the automated deployment process to both prepare the new environment and to deploy the correct version [20]. This allows for much more flexibility of deployments, where versions can be deployed when it is convenient to do so.

The prolonged deployment time also causes more downtime. Downtime is not currently perceived as an issue in itself, as it is negotiated with the customer and happens during the hours where downtime is acceptable. With continuous delivery, the company gains more incentive to reduce or eliminate downtime. In the case of CDM, downtime can be eliminated by utilizing for example parallel environments or canary releases [20]. With canary releases, a new version is rolled to a subset of the production server, and a part of the users are routed to the new version. If the new version is then deemed to work as intended, all of the users are routed to the new version.

One of the issues is that customers are having problems tracking the status of the development process in projects. This is caused by not deploying features that are not complete, and the relatively long length of the software delivery cycles. The customer is better kept on track of the development with continuous delivery, as the process is more transparent. In projects with a set duration it is much easier for the customer to understand the status of features when they are deployed in constant intervals.

To sum up the findings, continuous delivery provides an answer to many of the challenges faced in the current deployment process and product development in general. The management considers improving the deployment process to be one of the most important improvements. According to the findings, continuous delivery mainly increases the speed, quality, and capacity of the development. Speed and capacity are ensured by faster delivery, while quality is increased by the automated testing and faster feedback. Smaller problems can be quickly fixed without spending unnecessary time on manuall deploying a new version to the customer, and bigger changes only take as long as the implementation requires. After the initial investion, the practice will eventually allow the company to spend less money on management and operations, because both meeting costs and unnecessary repetitive work can be eliminated.

6.4 Continuous Experimentation's benefits for the case company

In the case company, selection of the development ideas is based almost exclusively on the opinions of management of the organization. The ideas for new features are often discussed in the steering group where only management is present. Studies report similar results in other companies as well [28]. Studies show that data-driven decision can be used to better guide the development by choosing features where the benefits can be measured [28, 33, 1]. Consequently, the business value of the software product can be increased by focusing the development on features that have the largest impact. The case company exposes features to customers only as soon as they are complete. By utilizing minimum viable features and minimum viable products, the decision to develop to feature or a product can be made from

an early stage. This enables conserving resources if the feature or product is found to be incompetent, or increasing the resources if the feature is seen as beneficial.

Applying continuous experimentation reduces the amount of guesswork and instead provides a systematic approach towards product development. However, Avinash Kaushik states in his article Experimentation and Testing primer that "80% of the time you/we are wrong about what a customer wants" [1]. The company Netflix has also found similar results, considering 90% of what they try to be wrong [33]. A way to tackle this issue is to adopt a process of continuous experimentation.

Data driven decisions are a way to gain competitive advantage by utilizing usage data or feedback in product development. Linking product development and business aspects verifies that the product is moving towards a direction with a higher business value. The consequence of utilizing data for product development is better knowledge of customer behavior, which can be used to better react to customers needs. Conducting experiments also allows companies to quantify business hypotheses and analytically derive answers, effectively allowing the use of hypotheses to steer development process.

The development time of features in the case company lasts from a single day up to two weeks. With the software deployed in intervals of two to four weeks, the features are completed before they are deployed to the customer environment. If at this point the feature is found to provide no value for the customer, it is too late to cancel the development process. With continuous experimentation, the development is done with smaller and measurable changes, and the minimal versions of features and products are validated with customer. Then, when a feature is found to provide no value for the customer, the development resources can be directed to other features.

The case company does not yet capitalise measuring metrics as part of the decision making process. By constantly measuring the new features of the product, breaking the product can be avoided. Breaking the product essentially means that features diminish the value of the product are implemented. Such features could for example be UI changes that are thought to increase the user experience, but fails to do so resulting from incorrect presumptions.

Dialog is seen as a suitable candidate for experiments due to the extensive user interface and human users. Currently the product development is primarily guided by direct customer needs. Features planned for Dialog that are not direct customer needs are discussed in the product steering group by the managemetn. Especially A/B testing is useful to determine the best web-page version [7]. The metric for success differs based on the business goal. In Dialog the important metrics stated by the product owner and management include shortest path to complete a task, and amount of paths available to perform certain features. A path consists of clicks required to perform a single task, such as to create a marketing campaign for a new

customer group. However, the metrics are only based on the persons own visions and perspectives.

> "Developers can make improvement suggestions, and occasionally product owners invent feature ideas. The ideas are based on a guess that the customer might find this valuable."
>
> (Dialog developer)

While the development is therefore mainly guided by customer needs that directly bring in revenue, there is a need to generalize the product for general customer needs and for more productization. Utilizing experiments in guiding the product strategy is seen as a viable solution by both the developers and management. However, the low user amount, 3-5 users per customer, does not suit experiments based on statistical importance. Experiments based on the user actions and behavior are therefore seen more suitable.

> "The amount of available statistical data is not high much due to the user amount, but the users actions can be tracked, and they continuously perform repetitive tasks. This customer would then provide a lot of usage data. If the time spent by this user could be halved, the customer would make more money. It doesn't therefore necessarily require a large user amount."
>
> (Dialog product owner)

A possibility expressed by a member of the management is to aid marketing by delivering a minimum viable product implementation of the product to new customers. This makes it is possible to pivot the product with customers, who can then test it in practice and make a decision whether to pay for the product or not. This strategy is similar to the lab website concept, where companies offer early versions of products free of charge for the purpose of collecting feedback [7]. One of the benefits of continuous experimentation is the increase in innovation resulting from the possibility to quickly learn from the customer.

Another experimental feature that is seen suitable for Dialog is an in-product survey. Surveys are placed at strategic locations in the product to obtain active customer feedback when the customer is performing a certain task. It can provide remarkably valuable information as compared to purely quantitative passive data.

CDM is perceived as a more challenging target for experiments due to the hard-coded interfaces and the nature of the application. The fact that there are no human users or a UI removes the possibilities of front-end A/B testing, a very common and well researched experiment. However, the developers aknowledge that there is a need for better understanding the customers needs. The team faces similar problems to the Dialog team, where a large part of the features are developed based on customer needs, and there is a need for more productization.

> CDM team leader "Most of the work is strived to be done in a customer-oriented way. Even with a very good product owner, hardly ever does anyone know what the customer wants if it hasn't been asked."

Regardless of excluding most of A/B testing, continuous experimentation can be practiced with different types of experiments that can guide the development of CDM. Building alternative implementations of existing features with attached instrumentation measuring performance can be used to determine if the current implementation or alternative implementation is preferable. Instead of relying on opinions, the decision can then be made based on the feedback received. Most of the desired improvements are related to performance, and thus the important metrics are related to amount of queries performed in a given timeframe and the duration of a single query.

> "CDM is a background system, and the performance related to queries is the most important metric for us. Imagine a situation where we're at a point analyzing how a query should be implemented, and whether to use a relational database or a document database. We can conduct an experiment where we implement both, test it with a data set and choose the more efficient query." (CDM team leader)

One of the main benefits of continuous experimentation in the case of CDM is increasing the knowledge of how customers are using the system. With the current development, user feedback or usage data is not systematically collected and capitalised. While the customers do provide feedback upon version releases, these happen relatively infrequently and a lot of development resources have already been spent on the features. Also a possibility is to focus the development towards important errors based on the usage data.

> "The software is composed of APIs, through which different components send and receive data. We could track the types of messages that are received and if they succesfully pass. This data can then be used to develop the API further, and to for example track typical errors." (CDM developer)

The lab website concept is less applicable for connected and embedded systems [7]. It would require the installation and integration of whole CDM, which is a laborious process. The experiment scenarios listed for embedded systems by Eklund and Bosch [15], shown in chapter 3.4, contain cases that can be used for CDM development via experiments. Tracking the access amount of different queries in the API can be used to focus development towards most used calls. Identifying features that are not used at all is also a way to steer development into more benefical cases. However, due to different

customers using the product in different ways, cross-case analysis has to be made to prevent failing to distinguish differences between users.

Continuous experimentation benefits the decision making and product development of the case company, and addresses the needs of producing more features that bring real value to the customer. The teams have to address issues regarding the trustworthiness of certain types of experiments, especially those relying on statistical importance. However, with continuous experimentation the teams can reduce the feedback cycle by utilizing minimum viable features and products, making data-driven decisions to guide the product development and to better focus the resources on features that generate real value for the customer.

6.5 Implementing Continuous Experimentation as a development process

As requirements and benefits of continuous experimentation have been investigated in the previous sections, a plan to implement the practice in the case company is presented here. First, the steps required for the case company to transition towards a continuous experimentation system are identified and explained. These steps have been formed based on the findings of previous research questions. Then, the continuous experimentation process is implemented by applying the continuous experimentation model by Fagerholm et al. [16] to the context of the case company.

6.5.1 Applying the general concept

Based on the previous research questions, the following steps were identified as critical requirements in the case company's transition towards continuous experimentation.

- Finding a pro-active lead customer
- Allocating time and resources for product development
- Implementing an infrastructure for experimentation
- Identifying metrics in the software products that increase the value for customer
- Investigating required legal agreements associated with data collection
- Educating employees to increase the competence in experimentation

One of the key factors in applying the concept is finding a pro-active lead customer that is willing to explore the new engagement model. The lead customer can then be used to train customer interaction with the new

engagement model, identify possible new challenges and form best practices with experimentation. A lead customer is also necessary to design and further develop the experimentation system. Olsson et al. have also found similar results [35].

A large part of the features implemented by the teams are custom implementations requested by the customers as part of projects. To be able to gain more suitable targets for experiments, the company has to focus more of its time and resources towards product development. Often the features requested by the customers are features that must be implemented in a certain way, and experimenting with them is not necessary. An example of such a feature is the integration of a new system to the software product. To proceed with continuous experimentation, the company must want to make data-driven decisions.

The company has to implement the infrastructure to support experimentation. The required components for the infrastructure are the data collection instrumentation, analytics tools and an experimental database [28, 16]. The developers view a pluggable logging service as a viable solution for data collection due to the modular architecture of the software products. The infrastructure should also allow the company to manage experiments by providing functionality for quickly executing and terminating experiments. Automating the creation of experimentation plans is also an opportunity that should be considered in further research [16]. Fagerholm et al. define that a suitable experimentation system must be able to release minimum viable products or features with suitable instrumentation for data collection, design and manage experiment plans, link experiment results with a product roadmap and manage a flexible business strategy [16].

Identifying which aspects of the product are valuable to the customer and the metrics to measure these attributes is also required to lead the experimentation design. Both the management and the developers were able to name few metrics that are perceived as important for the customer, but no deep knowledge of actual customer needs has been achieved. Increasing the knowledge in this area first allows the teams to improve the product strategy, and consequently helps with creating assumptions, hypotheses and experiments.

The company needs to negotiate with the lead customer about the legal aspects of data collection. Some companies have strict policies regarding data collection, and the case company must identify what kind of data can be collected and whether some data is confidental. Along with the data collection, the lead customer should approve the occasional user experience alteration that is necessary for certain experiments.

As the competence in experimentation and data-driven product design is low, the company must educate key personnel in these matters. The product management must have an experimentational mindset, while also understanding the statistical and technical limitations of experiments. The product

management also needs to understand the procedural, organization and customer requirements of continuous experimentation. The developers need to understand the technical challenges in order to design and experiments.

6.5.2 Applying the experimentation model

Fagerholm et al. define that a suitable experimentation system must be able to release minimum viable products or features with suitable instrumentation for data collection. The experimentation system must also allow the design and management of experiment plans, linking experiment results with a product roadmap and managing a flexible business strategy [16]. The Build-Measure-Learn cycle starts with forming a hypothesis and building a minimum viable product (MVP) or a minimum viable feature (MVF) with tools for data collection. Once the MVP has been created, the data is analyzed and measured in order to validate the hypothesis. To persevere with a chosen strategy means that the experiment proved the hypothesis correct, and the full product or feature can is implemented. However, if the experiment proved the hypothesis wrong, the strategy is changed based on the implications of a false hypothesis.

The infrastructure and Build-Measure-Learn model [16] introduce the general process of continuous experimentation. Here the model is mapped to the case company context, and elaborated based on the findings of earlier research questions. The analysis of the model is divided into three phases: design, execute and analyse.

In the design phase the roadmap is created and iterated by the business analyst and product owner, while experiments are designed, executed and analyzed by a data analyst. In the case company the teams lack a data analyst, and team leaders and management are the personnel in constant interaction with the customer. As both product owners and team leaders are in constant customer interaction, they are the personnel with most knowledge of the business strategy to form assumptions from the it. Developers are only minimally involved in the customer interaction, and lack deep knowledge of the customer domain. The teams also have to educate themselves on data analytics, hire a data-analyst or use other personnel in the company to assist in turning the hypotheses into experiments.

1. Form an assumption from the business model that is thought to increase the value of the product

2. Form a hypothesis based on the assumption that can be subject to experimental testing

3. Define the type of the experiment and the experimental problem so that it can be run with trustworthy results

4. If running a controlled experiment, define an Overall Evaluation Criteria that can be collected and used to provide an answer to the hypothesis
5. Implement the MVF or MVP
6. Implement the instrumentation to collect the metric

The process of designing an experiments starts by forming an assumption from the business strategy, which is perceived as increasing the value of the product for the customer. After the hypothesis is formed, the type of experiment to test the hypothesis needs to be chosen. The experiment must be able to provide worthful results, and for experiments relying on statistical importance, the user amount has to be sufficient. For experiments relying on user behavior, statistical importance might not be necessary to draw results.

Then, if running a controlled experiment, the overall evaluation criteria (OEC) is explicitly defined before the feature or MVP is implemented. The OEC is a metric that the company aims to improve with the experiment. One of the common pitfalls in running controlled experiments is to pick an OEC for which the control group is easy to beat by doing something "wrong" from a business perspective [9]. Picking the OEC also concretizes the experiment, making the purpose and goal of the experiment more obvious.

After the OEC has been formed and the experiment type decided, the minimum viable feature (MVF) or minimum viable product (MVP) is implemented, and the instrumentation for data collection is added to the product. The suitable data collection instrumentations for the case company's product according to the developers is a server-side service layer, which can be used to log data from multiple services. For this phase, the technical infrastructure has to be built that supports quickly designing, executing and analysing experiments.

The project management tools used by the teams suit the experimentational model. A kanban board is already used to transfer tasks from the management to the developers, and can be used by to develop experiments as well. The product owners and analysts must first design the experiment, and insert a new task into the backlog with detailed specifications. A developer then checkouts the task and implements the MVF or MVP with instrumentation.

1. The software is deployed to a UAT environment
2. The software is acceptance tested in the UAT environment, production deployment is negotiated
3. The software is deployed to production environment
4. The software is run for a period of time, while the data is collected into a database

In the execute phase the product is deployed to an environment with the experimental feature and instrumentation for data collection. For the case company, the deployment has to undergo testing in the UAT environment. The UAT environment is also suitable for testing the experiment itself and the impact of the experimental feature on the product. When the UAT phase is complete, the production deployment is negotiated and performed. With automated deployments, the production deployment should happen by a click of a button. The software is the run for a period of time, while the required data is collected into a database located in the environment.

1. The data is uploaded from the database and analysed through the infrastructure
2. The team draws conclusions from the data on whether to develop the new feature or product further, keep it as it is, or to cease it and revert back to the unmodified software.

In the analyze phase the results from the experiment are collected from the database and analysed through analytics software. The business strategy is then persevered or pivoted based on whether the hypothesis is valid and supports the assumption.

7 Discussion

The study investigates extending the development process towards continuous delivery and continuous experimentation to advance the company towards real-time value delivery. The research questions address the challenges faced in this transition, the benefits found from these practices and systematical application of continuous experimentation in the B2B domain.

The results suggest that the challenges faced in continuous delivery in the B2B context are multidimensional, and related to the technical, procedural and customer aspects. The major difference a company operating in the B2B domain faces in the transition as compared to the B2C domain is that there are plenty of customers with unique properties with their business relying on the software. The primary issues causing these challenges are the diverse customer owned environments and the importance of the software product for the customer.

Figure 8 visualizes the challenges the case company faces in transition towards continuous delivery. Multiple challenges are related to two or more aspects, and the problems affecting all aspects can be seen as the core challenges. Acceptance testing is related to all aspects since customers want to perform acceptance testing with new versions, automated acceptance testing has to be implemented and the user acceptance testing is required before a production release can be made. Another challenge related to all

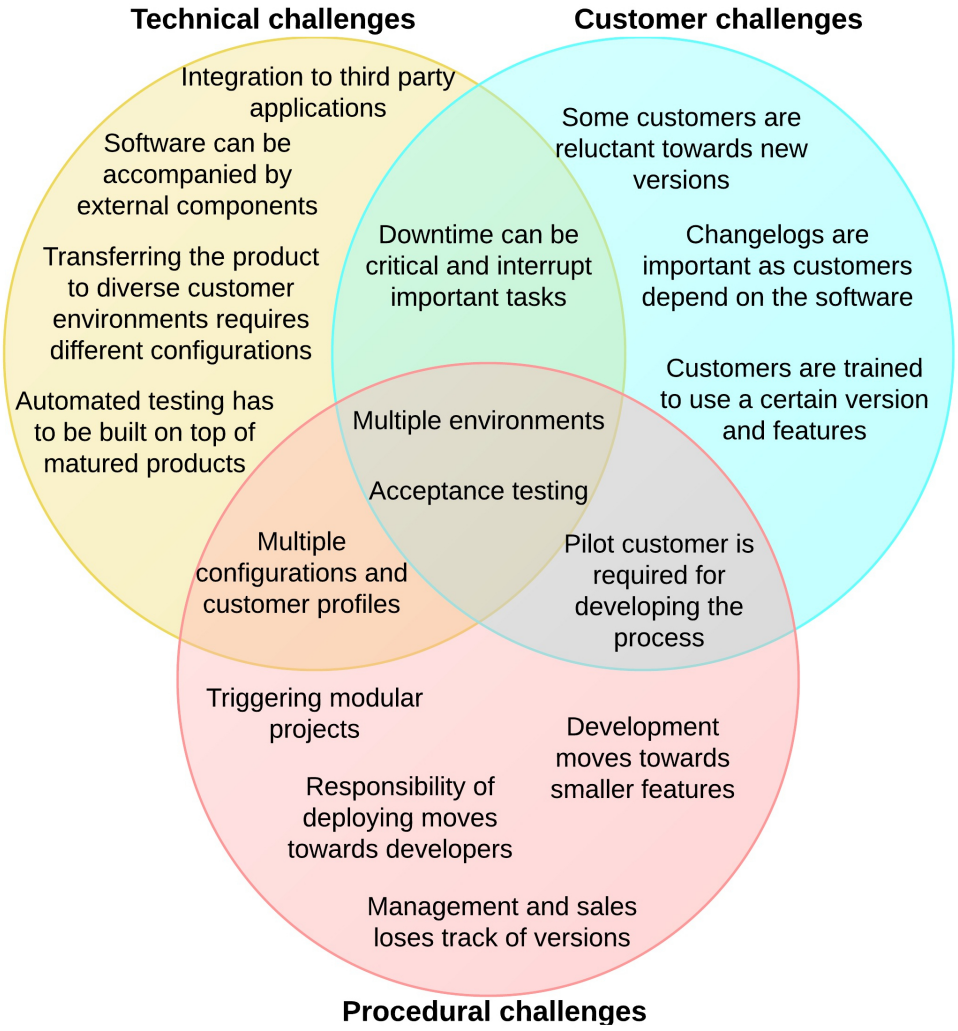

Figure 8: Case company's challenges in continuous delivery.

aspects is the diversity of customer environments. It affects the technical implementation, as software has to be transferred to diverse environments. The procedural challenge is that the software has to be deployed multiple customers, and it has to be decided whether each version is always released to every customer.

The findings are in align with and could be considered as extending some of the theoretical contributions by Olsson et al. [35], who researched the transition towards continuous delivery and an experimentational system. Identifying that transitioning towards continuous delivery requires a company to address issues in multiple aspects of the company also benefits companies in practice.

Challenges faced in continuous experimentation are divided into four aspects: technical, organizational, customer and A/B testing. Figure 9 visualizes the challenges in the first three aspects, and the main benefits found from continuous experimentation. Additionally, the challenges related to A/B testing are useful in highlighting the important product characteristics affecting experiments. The challenges are mostly related to the lack of experimental mindset, low user volume, technical properties of the software products and importance of the software product for the customer. The importance of the software product means that the business of the customers depends on the application, and changes should be done carefully enough not to unnecessarily complicate the usage.

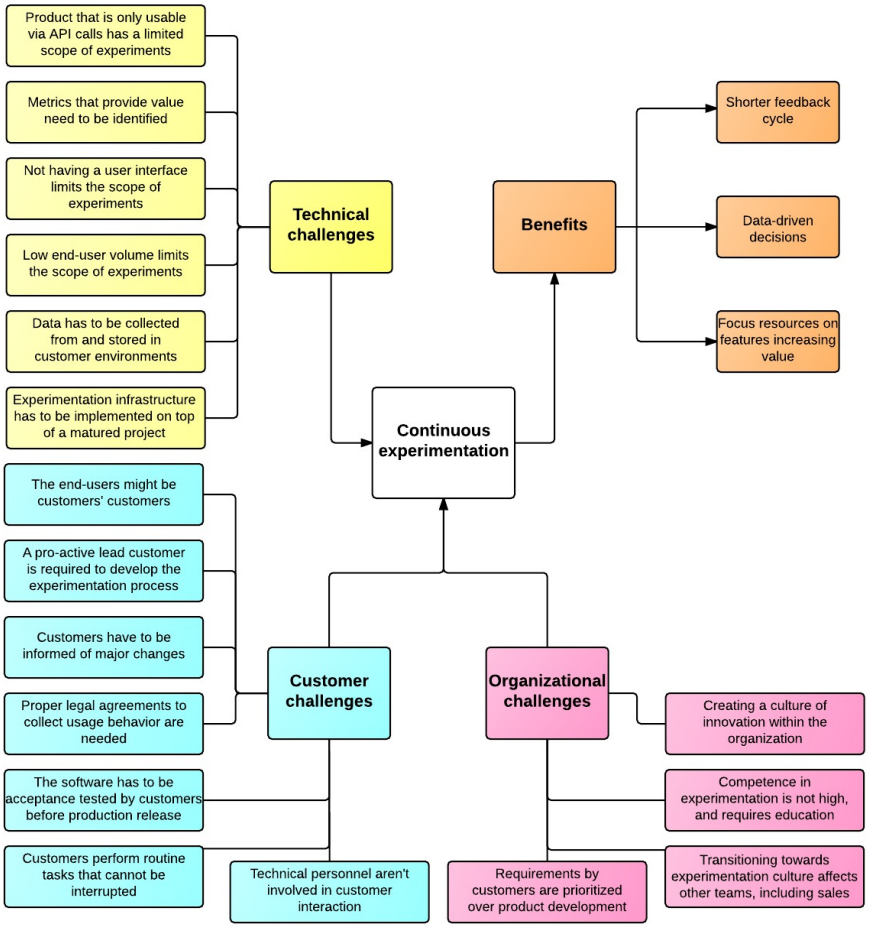

Figure 9: Challenges and benefits of continuous experimentation.

The results extend the theoretical contributions by Olsson et al. [35]

by providing a list of requirements the case company has to resolve when it transitions towards continuous experimentation. However, the challenges found in continuous experimentation are not as overlapped as in continuous delivery. This could implicate that the challenges are not yet identified on a fundamental enough level required to form these connections.

The study also suggests that continuous delivery corresponds to many of the case company's primary needs. The issues related to the deployment process are considered very important by both of the teams. The issues include low reliability in new versions, human error factors when performing version releases and deployments, and long feedback cycles. One of the main benefits of continuous delivery is that the software is kept in a state where it is always ready for deployment [20], even if the actual deployments are not made more often.

These issues into which the benefits were mapped were found by researching the current deployment process and challenges faced in the development process. An unexpected large part of the major issues stated by the interviewees were related to deploying the software, and it was identified as one of the major challenges in the current development. The benefits found from continuous delivery, which were sought from existing literature [20, 34], matched the challenges surprisingly well. By taking the time to map the actual encountered problems to solutions, the concept of continuous delivery can be better sold to the management of companies.

Continuous experimentations benefits for the case company are related to choosing which features to develop and which to cancel. In the current development model, these decisions are made only after the feature has been completed and deployed to the customer. With continuous experimentation, the development is done in shorter cycles which allow for faster feedback. For each feature, the decision to either persevere or pivot with the product strategy is based on collected quantitative or qualitative data instead of opinion. Figure 9 visalizes the three main benefits on a very general level. Shorter feedback cycle is achieved by utilizing minimum viable features and products in the development. Data-driven decisions means that the business strategy and software product is guided by relying on measured quantitative or qualitative data from experiments. Finally, the result is that the development can better and earlier be guided towards features providing actual value.

Performing continuous experimentation also differs based on the nature of the software product. The product characteristics affecting the experiments are most distinctly the user interface and human users. The two products under inspection in this thesis provided two different viewpoints to consider when planning suitable experiments. For Dialog, a system with a wide UI and high importance of user experience, front-end A/B testing and usage behavior tracking were found as viable experiments. However, with A/B testing the user amount limits the statistical significance, and the user base

has to be increased to draw trustworthy conclusions. For the intagrated background application CDM that has no UI and human users, the types of suitable experiments are those not relying on user behavior. Hoever, tracking usage metrics can be used to measure the distribution in API calls. Another suitable experiment is feature alpha, where a new feature or product is tested with a small proportion of customers.

The experimentation infrastructure and Build-Measure-Learn model [16] suit the case company well. As the Build-Measure-Learn model introduced is very general, certain steps required elaboration. Such elaborations included refining the three phases of the Build-Measure-Learn model: design, execute and analyse. For design phase, defining Overall Evaluation Criterias for controlled experiments was added as a step, since a common pitfall is to incorrectly choose the OEC [9]. For the execute phase, a deployment to the UAT environment was added as the customers want to perform manual acceptance testing before each production release. The contributions to the experimentational model are investigating the challenges faced by the model in the B2B domain.

The findings supported the research questions well. Many of the challenges in the current development process of the case company are in line with the benefits of continuous delivery and continuous experimentation. Extending the development process towards continuous delivery and continuous experimentation is an expensive but benefical transition. Implementing the technical infrastructure requires the investment of plenty of resources, while some of the other challenges are simply a matter of decision. Another challenge requiring more resources is educating personnel, which is required especially for stakeholders designing and analyzing experiments.

8 Conclusion

8.1 Summary

This thesis was motivated by the general increased interest towards innovative experimental systems [35], and the lack of related studies in the B2B domain. While especially continuous delivery and partly continuous experimentation have been utilized and researched in the B2C domain, the benefits and challenges are still vague for B2B companies. Understanding the central aspects of continuous delivery and continuous experimentation will be a must for software companies willing to stay ahead of competitors in the current rapidly moving industry.

The logical starting point of this thesis was an open issue by the case company: "how to validate whether a feature would be succesful or not?". This issue is fundamental for the case company, as knowing the answer allows the case company to focus on developing features that develop real value for the customer. Both practices investigated in this thesis provide solutions to

this issue. Shortening the feedback cycle by exposing features to customers as fast as possible allows the company to receive feedback faster. This is done by both automating the deployment process and utilizing minimum viable products and features. Based on the quick feedback, the product development can be steered towards the correct direction from an early stage. Additionally, measuring metrics from the features wtih experiments allows the company to make decisions based on qualitative or quantitative data instead of opinion.

The focus of this thesis was not solely on the technical aspects of these development processes, as research already exists on these subjects [28, 15, 20]. Rather, the thesis focused on identifying the challenges in the transition towards these processes as a whole. The analysis of continuous delivery and continuous experimentation proceeded from higher-level considerations of B2B requirements to more detailed focus of applying continuous experimentation in B2B context. The findings provide insights into the differences between B2C and B2B companies, the software products in these domains and the development models used. In addition to the difference in two domains, the findings also compare the development models from the viewpoint of two different software products.

This thesis has identified the main requirements a company operating in the B2B domain has to address when applying continuous delivery and continuous experimentation. For continuous delivery, the challenges can be divided into technical challenges, procedural challenges and challenges related to the customer. These challenges are mostly related to having multiple customers with diverse environments that rely on the software product. For continuous experimentation the challenges can be divided into technical challenges, organizational challenges and challenges related to the customer. As with continuous delivery, these challenges are related to having multiple customers that rely on the software. Additionally, moving to continuous experimentation requires an organizational culture shift towards an experimental mindset. A/B testing was examined from the viewpoint of two software products to survey challenges caused by product specific characters. Characters mostly affecting the suitability of experiments are the amount of human users and the convention by which the software is used.

The benefits of these practices matched to many problems found in the case company, and a company operating in similar domain with similar products can use them as a basis when considering applying these practices. The most important challenges in the case company were found to be related to the deployment process and guiding the development of the software products. By utilizing continuous delivery, the case company can solve problems such as long feedback cycles, low reliability in new versions and high amount of resources required for releases. Continuous experimentation on the other hand allows the case company to better guide it's product development, and rely on quantitative or qualitative data instead of opinions

when choosing which features or products to develop further.

The third research question identified the actual steps the company has to complete in the transition towards continuous experimentation. To apply the continuous experimentation model, the case company needs to implement the technical infrastructure, find a pro-active lead customer, allocate time and resources for product development, educate employees, investigate legal agreements related to data collection and identify metrics to determine what types of experiments can be performed. Additionally, a Build-Measure-Learn cycle [16] was analyzed and mapped to the context of the case company. The deviations included deployment of the software to the user acceptance environment and defining an overall evaluation criteria for controlled experiments.

8.2 Limitations

Since case studies only allow analytic generalisations instead of statistical generalisations, cannot the findings be directly generalised to other cases. This limitation applies especially to the first two research questions, which mostly relied on the findings found from interviews. Limiting direct generalisation also applies to applying continuous experimentation in practice, since software products and customers inevitably differ.

Two types of triangulation were used: data triangulation by including persons with different roles into the interviews, and methodological triangulation by collecting documentary data and observations by the author. However, the reliability of the results could have been increased by employing observer triangulation and theory triangulation.

Regardless of the limitations a case study has, the phenomena was deeply understood through gathering a large amount of qualitative data and systematically analysing it. Therefore the core findings should be applicable to similar research problems outside of the empirical context of this thesis. This means that the B2B challenges and benefits of continuous delivery and continuous experimentation can be considered as a starting point for further studies in other contexts where these development models take place.

The scope was limited to the development process of two software products within the case company, and direct generalisations to other domains and products should be avoided. However, the findings of this thesis are in align with and could be considered as extending some of the theoretical contributions by Olsson et al. [35], who researched the transition from agile development towards continuous deployment.

While the data collection was applied to the teams developing the software products, the upper management of the company was not interviewed. The variety of responses in other teams was not explored, or the multitude of other different possible contexts in software development in the B2B domain. The responses from the interviewees mostly differentiated only based on

the software product being developed, and were quite similar in terms of composition and experience.

8.3 Further research

An interesting research question for further research is how the core findings of the thesis can be transferred to other companies working in the B2B domain with different software products. This includes investigating deviations found in the challenges, benefits and the experimentational model stated in this thesis. The challenges found in the transition phases also have to be validated by adopting the respective development model in practice.

The individual aspects where challenges were found could be more deeply analyzed for both continuous delivery and continuous experimentation. Also, grouping the challenges into different groupings can yield interesting results.

An interesting question regarding the continuous experimentation model discussed in research question 3 is how to build the actual back-end infrastructure for experimentation. Additionally, further researching the challenges in performing experiments in the B2B domain is required. This thesis can be considered as a starting point for identifying the domain specific challenges.

References

[1] http://www.kaushik.net/avinash/experimentation-and-testing-a-primer/.

[2] march 2014. http://mcfunley.com/design-for-continuous-experimentation.

[3] august 2014. http://tv.adobe.com/watch/max-2013/the-innovation-pipeline-how-adobe-defines-the-next-big-thing-in-web-design.

[4] *Experimentation platform*, march 2014. http://www.exp-platform.com/.

[5] Adams, Rob J, Evans, Bradee, and Brandt, Joel: *Creating small products at a big company: adobe's pipeline innovation process.* In *CHI'13 Extended Abstracts on Human Factors in Computing Systems*, pages 2331–2332. ACM, 2013.

[6] Beck, Kent: *Extreme programming explained: embrace change.* Addison-Wesley Professional, 2000.

[7] Bosch, Jan: *Building products as innovation experiment systems.* In *Software Business*, pages 27–39. Springer, 2012.

[8] Cockburn, Alistair: *Agile software development*, volume 2006. Addison-Wesley Boston, 2002.

[9] Crook, Thomas, Frasca, Brian, Kohavi, Ron, and Longbotham, Roger: *Seven pitfalls to avoid when running controlled experiments on the web.* In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1105–1114. ACM, 2009.

[10] Denzin, Norman K and Lincoln, Yvonna S: *The discipline and practice of qualitative research.* Handbook of qualitative research, 2:1–28, 2000.

[11] Dubois, Anna and Gadde, Lars Erik: *Systematic combining: an abductive approach to case research.* Journal of business research, 55(7):553–560, 2002.

[12] Dybå, Tore and Dingsøyr, Torgeir: *Empirical studies of agile software development: A systematic review.* Information and software technology, 50(9):833–859, 2008.

[13] Dzamashvili Fogelström, Nina, Gorschek, Tony, Svahnberg, Mikael, and Olsson, Peo: *The impact of agile principles on market-driven software product development.* Journal of Software Maintenance and Evolution: Research and Practice, 22(1):53–80, 2010.

[14] Easton, Geoff: *Critical realism in case study research.* Industrial Marketing Management, 39(1):118–128, 2010.

[15] Eklund, Ulrik and Bosch, Jan: *Architecture for large-scale innovation experiment systems.* In *Software Architecture (WICSA) and European Conference on Software Architecture (ECSA), 2012 Joint Working IEEE/IFIP Conference on*, pages 244–248. IEEE, 2012.

[16] Fagerholm, Fabian, Guinea, Alejandro Sanchez, Mäenpää, Hanna, and Münch, Jürgen: *Building blocks for continuous experimentation.* In *Proceedings of the 1st International Workshop on Rapid Continuous Software Engineering (RCoSE 2014), Hyderabad, India*, 2014.

[17] Fowler, Martin and Foemmel, Matthew: *Continuous integration.* Thought-Works) http://www.thoughtworks.com/Continuous Integration. pdf, 2006.

[18] Gephart, Robert P: *Qualitative research and the academy of management journal.* Academy of Management Journal, 47(4):454–462, 2004.

[19] Healy, Marilyn and Perry, Chad: *Comprehensive criteria to judge validity and reliability of qualitative research within the realism paradigm.* Qualitative market research: An international journal, 3(3):118–126, 2000.

[20] Humble, Jez and Farley, David: *Continuous Delivery: Reliable Software Releases Through Build, Test, and Deployment Automation.* Addison-Wesley Professional, 1st edition, 2010, ISBN 0321601912, 9780321601919.

[21] Humble, Jez, Read, Chris, and North, Dan: *The deployment production line.* In *Agile Conference, 2006*, pages 6–pp. IEEE, 2006.

[22] King, Nigel: *Template analysis.* 1998.

[23] King, Nigel: *Doing template analysis.* Qualitative organizational research: Core methods and current challenges, pages 426–250, 2012.

[24] King, Nigel, Cassell, C, and Symon, G: *Using templates in the thematic analysis of texts.* Essential guide to qualitative methods in organizational research, pages 256–270, 2004.

[25] Kitchenham, Barbara, Pickard, Lesley, and Pfleeger, Shari Lawrence: *Case studies for method and tool evaluation.* IEEE software, 12(4):52–62, 1995.

[26] Kitchenham, Barbara A, Pfleeger, Shari Lawrence, Pickard, Lesley M, Jones, Peter W, Hoaglin, David C., El Emam, Khaled, and Rosenberg,

Jarrett: *Preliminary guidelines for empirical research in software engineering.* Software Engineering, IEEE Transactions on, 28(8):721–734, 2002.

[27] Kohavi, Ron, Deng, Alex, Frasca, Brian, Walker, Toby, Xu, Ya, and Pohlmann, Nils: *Online controlled experiments at large scale.* In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1168–1176. ACM, 2013.

[28] Kohavi, Ron, Henne, Randal M, and Sommerfield, Dan: *Practical guide to controlled experiments on the web: listen to your customers not to the hippo.* In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 959–967. ACM, 2007.

[29] Kohavi, Ron, Longbotham, Roger, Sommerfield, Dan, and Henne, Randal M: *Controlled experiments on the web: survey and practical guide.* Data Mining and Knowledge Discovery, 18(1):140–181, 2009.

[30] Kohavi, Ronny, Crook, Thomas, Longbotham, Roger, Frasca, Brian, Henne, Randy, Ferres, Juan Lavista, and Melamed, Tamir: *Online experimentation at microsoft.* Data Mining Case Studies, page 11, 2009.

[31] Loshin, David: *Master data management.* Morgan Kaufmann, 2010.

[32] Moran, Mike: *Multivariate testing in action: Quicken loan's regis hadiaris on multivariate testing.* Biznology Blog by Mike Moran.

[33] Moran, Mike: *Do it wrong quickly: how the web changes the old marketing rules.* IBM Press, 2007.

[34] Neely, Steve and Stolt, Steve: *Continuous delivery? easy! just change everything (well, maybe it is not that easy).* In *Agile Conference (AGILE), 2013*, pages 121–128. IEEE, 2013.

[35] Olsson, Helena Holmström, Alahyari, Hiva, and Bosch, Jan: *Climbing the" stairway to heaven"–a multiple-case study exploring barriers in the transition from agile development towards continuous deployment of software.* In *Software Engineering and Advanced Applications (SEAA), 2012 38th EUROMICRO Conference on*, pages 392–399. IEEE, 2012.

[36] Ōno, Taiichi: *Toyota production system: beyond large-scale production.* Productivity press, 1988.

[37] Palmer, Steve R and Felsing, Mac: *A practical guide to feature-driven development.* Pearson Education, 2001.

[38] Poppendieck, Mary and Poppendieck, Tom: *Lean software development: an agile toolkit.* Addison-Wesley Professional, 2003.

[39] Reichheld, Frederick F: *The one number you need to grow.* Harvard business review, 81(12):46–55, 2003.

[40] Ries, Eric: *The lean startup: How today's entrepreneurs use continuous innovation to create radically successful businesses.* Random House LLC, 2011.

[41] Robson, Colin: *Real word research.* Oxford: Blackwell, 2002.

[42] Runeson, Per and Höst, Martin: *Guidelines for conducting and reporting case study research in software engineering.* Empirical software engineering, 14(2):131–164, 2009.

[43] Schwaber, Ken and Beedle, Mike: *Agile software development with scrum.* 2002.

[44] Seaman, Carolyn B.: *Qualitative methods in empirical studies of software engineering.* Software Engineering, IEEE Transactions on, 25(4):557–572, 1999.

[45] Stake, Robert E: *The art of case study research.* Sage, 1995.

[46] Steiber, Annika and Alänge, Sverker: *A corporate system for continuous innovation: the case of google inc.* European Journal of Innovation Management, 16(2):243–264, 2013.

[47] Strauss, Anselm and Corbin, Juliet M: *Basics of qualitative research: Grounded theory procedures and techniques.* Sage Publications, Inc, 1990.

[48] Zelkowitz, Marvin V and Wallace, Dolores R.: *Experimental models for validating technology.* Computer, 31(5):23–31, 1998.

A Interview questions

ID	Question
1	Name of the interviewee
2	Team of the interviewee
3	Position of the interviewee
4	Years in the company
5	Years of experience in the industry
6	What is the software product you're working with?
7	Describe your personal daily work routine.
8	Describe a normal week with your team.
9	What is the current development model in your team?
10	How has the development model evolved during your stay in the company?
11	Where do the development ideas come from? Are they mostly requirements from customers?
12	Are the current development ideas based on evidence or guesswork?
13	What is the current deployment process like?
14	How is it decided when to deploy?
15	How is it decided what to deploy?
16	How often is new version released?
17	How often are the new versions deployed to customer?
18	How are the deployment dates chosen?
19	Which parts of the deployment process are manual?
20	Which parts of the deployment process are hard to measure automatically?
21	Is the customer involved in the deployment process?
22	Does the customer have to do something when a version is deployed?
23	What are the strengths in the current deployment process?
24	What are the problems in the current deployment process?

ID	Question
25	Describe the customer interaction process.
26	From your point of view, what are the challenges with the customer interaction?
27	From your point of view, what are the strengths with the customer interaction?
28	From your point of view, how could the interaction with the customer be improved?
29	Is it common for customers requirements to change?
30	Is the development team aware of the customers present requirements?
31	How is feedback collected from the customer?
32	What are the B2B specific challenges in feedback collection?
33	From your point of view, how could the feedback collection be improved?
34	How is the customer feedback used?
35	From your point of view, how could the feedback usage be improved?
36	How many end-users does the product have?
37	How is usage data collected?
38	How could usage data collection be improved?
39	What challenges would real-time deployment have?
40	What challenges would plugged-in data collection instruments have?
41	If data were to be collected in customer environment, what challenges would be faced storing it?
42	What are the strengths in the current development process?
43	What are the problems in the current development process?
44	Could your product be instrumented with a data collection service? If not, why?
45	In your product, could experiments be quickly deployed to the customer environment?
46	In your product, could the development process be guided by A/B testing?
47	Does your team have a data analyst?