Extending the "Development Pipeline" Towards Continuous Deployment and Continuous Experimentation: A Case Study in the B2B Domain

Olli Rissanen

Master's thesis
University of Helsinki
Department of Computer Science

Helsinki, September 11, 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 Pipeline" Towards Continuous Deployment and Continuous Experimentation: A Case Study in the B2B Domain

Oppiaine — Läroämne — Subject
Computer Science

Työn laji — Arbetets art — Level	Aika — Datum — Month and year	Sivumäärä — Sidoantal — Number of pages
Master's thesis	September 11, 2014	42

Tiivistelmä — Referat — Abstract

Delivering more value to the customer is the goal of every software company. To increase the value of a software product, a deep understanding of the customers behavior has to be achieved. Producing, testing and delivering software in real-time allows faster reaction and shorter feedback loops. Delivering value in real-time requires a company to utizile real-time delivery, data-driven decisions and empirical evaluation of new products and features. This thesis investigates practices known as continuous delivery and continuous experimentation to tackle real-time delivery and data-driven decisions. Continuous delivery is a design practice where the software functionality is deployed continuously to customer environment. This process includes both automated builds and automated testing, but also automated deployment. Continuous experimentation is a design practice where the entire R&D process is guided by controlled experiments and 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. This paper conducts a deductive case study in a medium-sized software company to examine the development process of two different software products. The scope of this study is in the B2B domain, where the customer is another company instead of an individual. As a result, challenges and best practices regarding the transition towards real-time value delivery in the B2B domain are identified. TODO: something about cross-case analysis with the two different products, TODO: exploratory case study, TODO: critical realism view

ACM Computing Classification System (CCS):
General and reference → Experimentation
TODO: add rest

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

Säilytyspaikka — Förvaringsställe — Where deposited

Muita tietoja — Övriga uppgifter — Additional information

Contents

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

Currently more and more software companies are moving to lean practices, which include short delivery cycles and thus shorter feedback loops.

The requirement of developing value fast have lead to the evolution of a new software development model [6]. 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.

-Delivering value in real-time -requires: real-time delivery -requires: empirically evaluating new products and services

-Deep customer insights -data-informed solutions -a deep understanding of customers and products by gathering data continuously from the live use of the products.

Lean software development, which was inspired by the Toyota Production System [], is a development approach attempting to eliminate unnecessary work and to create value for the customer. The approach consists of seven principles: eliminate waste, build quality in, deliver fast, optimise the whole, create knowledge, defer commitment and respect people.

TODO: Connect this thesis with the Lean Startup Build-Measure-Learn cycle

-Interesting aspects: -Products with a history cannot be developed in a MVP manner, more like in a MVF (feature) manner.

IEEE divides the quality of a software component into two components: how well the software meets software requirements specifications, and how useful the software is to the end user.

CMMI 5th stage

A way to shorten the delivery cycles and to automate the delivery process is continuous deployment. It is an extension to continuous integration, where the delivery process is often entirely automated, and software functionality is deployed frequently to customer environment. While continuous integration defines a process where the work is automatically built, tested and frequently integrated to mainline [15], often multiple times a day, continuous deployment adds automated acceptance testing and deployment. Continuous deployment therefore attempts to deliver an idea to users as fast as possible by automating the deployment process.

Define continuous experimentation principles here

In the context of this thesis, Continuous Experimentation refers to a development model following these principles: TODO: improve this

- Data driven decisions

- Link product development and business aspect
- React to customers present needs
- Turn hypotheses into facts
- Steer development process

Bosch et al. introduce an innovation experiment system, where the development process consists of frequently deploying new versions, using customers and customer usaged data in the development process and finally by focusing on innovation and testing ideas with customers to drive customer satisfaction and revenue growth [6]. "First, it frequently deploys new versions focusing on continuously evolving the embedded software in short cycles of 2-4 weeks" [13].

Continuous experimentation attempts to validate that an idea 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 [34]. The process in continuous experimentation is to first form a hypothesis based on a business goals and customer "pains" [6]. 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 attempt to validate the hypothesis.

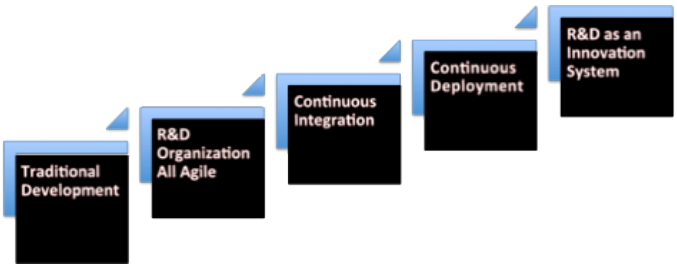

Figure 1: Organization evolution path [30].

The study is an exploratory deductive case study, which aims to explore how continuous deployment and continuous experimentation can be integrated to the development process. The study specifically aims to identify the main requirements, problems and key success factors with regards to these approaches. 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 deployment and continuous experimentaton also requires understanding the requirements of continuous deployment and continuous experimentation, and restrictions caused by the developed software product.

In this research, the units under the study are two teams responsible for developing two different software products. By focusing on different products, a broader view on the application 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, CDM, is a Master Data Management [27] solution running as a background application. The objective of this thesis can be therefore summared 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.
RQ 1: What are the B2B specific challenges of continuous delivery and continuous experimentation?
In B2B domain the connection with end users might not be established properly, and the customer feedback might only be received from certain key members. Running experiments on customer environments also requires a new deployment of the software product. The purpose of this research question is to identify the differences in the software development process and the product in B2B and B2C environments, and it is answered based on both literature review and interview. Question: contractual issues? Question: selecting the correct OEC in b2b is harder. for example revenue. question: in B2C people develop their own product, but in B2B the feature ideas might come only from customer.

RQ 2: 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 question is answered by comparing problems and benefits of literature review to the results of the interview.

One idea is also to aid marketing by selling the MVPs to customers. This way we can pivot the product with customers, they can test it in practice and then decide whether to pay for the product or not.

RQ 3: 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 adopted in practice. This requires an answer to questions what, how, who and when. This question is answered by applying Fagerholm et al. model [14] to two different teams and software products within the case company.
-Where do the feature ideas come from? -What roles are required? -What kind of deviations are there in the model?

1.3 Scope and limitations

This study focuses on the

The background theory is drawn from literature focusing on continuous delivery, continuous experimentation, lean startup and experiment innovation 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 they do 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.

2 Literature: Continuous Delivery

Agile software development emerged from the iterative software development methods, with early implementations being development in the 1990's [11]. 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.

Many agile models, such as Scrum [37], Extreme Programming [5] and Feature-Driven Development [32] further defined 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 all code.

Lean manufacturing, derived from the Toyota Production System [31], 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 [33] has been developed based on the same idea. The purpose of lean software development is to eliminating waste, or anything that doesn't bring value to the customer, and to pursuit 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, build quality in and see the whole.

Lean startup [34] is a business and product development method with focus on business-hypothesis-driven experimentation and iterative product releases. A primary purpose of lean startup is to assess the specific demands of consumers and to fulfill those demands with the least possible resources. This is achieveved by using a Build-Measure-Learn-loop, which 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.

TODO: -Stairway to heaven -Innovation experiment system

2.1 Development pipeline

2.2 Continuous deployment

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 is looking for an deployment approach that grants the following benefits.

- Fast feedback on changes
- Automation of repetitive actions
- Validating that deployed code has passed automatised 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 filling the aforementioned benefits is called Continuous deployment.

Continuous deployment 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 [15], often multiple times a day, continuous deployment adds automated acceptance testing and deployment. The purpose of continuous deployment is that as the deployment process is completely automated, it reduces human error, documents required for the build and increases confidence that the build works [18].

In an agile process software release is done in periodic intervals [7]. Compared to waterfall model it introduces multiple releases throughout the development. Continuous deployment, on the other hand, attemps to keep the software ready for release at all times during development process [18]. 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 deployment 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-

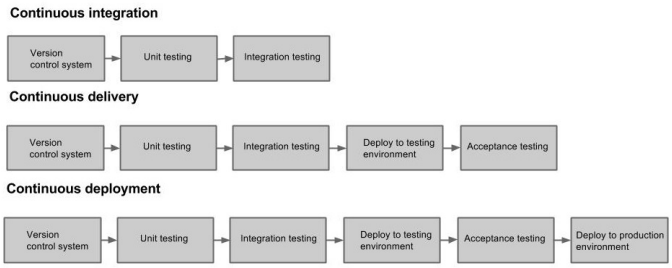

Figure 2: Continuous integration, delivery and deployment.

ment. Refer to Fig. 2 for a visual representation of differences in continuous integration, delivery and deployment. Both 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 proven deployable [18]. 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 [30]. One of 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. 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. Olsson et al. also note that finding a pro-active lead customer is also essential to explore and form aa new engagement model.

2.2.1 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 [18]. A deployment pipeline can be loosely defined as a consecutively executed set of validations that a software has to pass such 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 [18]. Refer to Fig. 3 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 [19]. 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 element

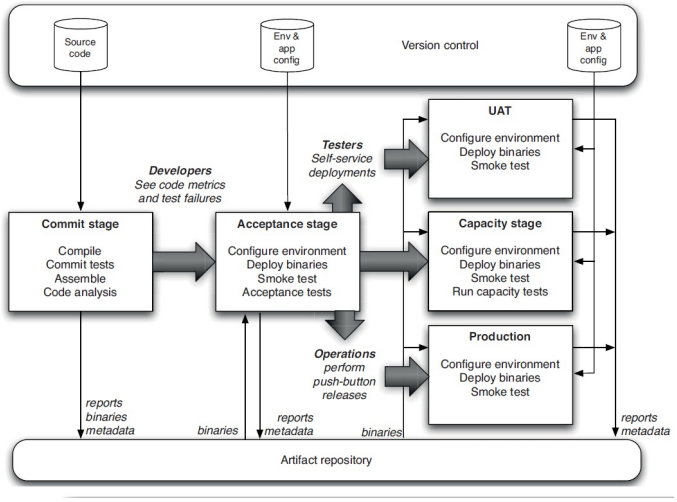

Figure 3: A basic deployment pipeline [18].

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.

2.2.2 Impact on development model

A picture of the typical development process in continuous deployment is shown in Fig. 4. After the team pushes a change to the version control system, the project is automatically built and tests are triggered stage by stage. 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.

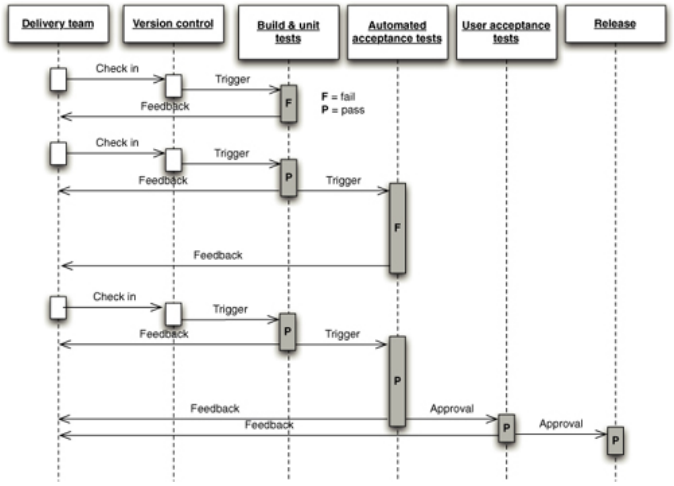

Figure 4: Components of the development process [18].

2.3 Challenges regarding continuous delivery

"With continuous flow, Sales and Marketing were not quite sure when anything would be released." "We had to provide mechanisms where they could track the status of work in real time, so they can plan accordingly." [29]

-Technical implementation -Education on the subject -Active participation of employees

3 Literature: Continuous Experimentation

3.1 Continuous experimentation

-for a method: what, how, when, who

Continuous experimentation is a term created by Dane McKinley, a Principal Engineer at Etsy [1]. 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 will be found, and planning to be wrong should be considered [1].

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 [6]. 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 [14]. The Build-Measure-Loop of Lean Startup is similar in the sense that controlled experiments drive the development.

In Lean startup methodology [34] 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 [30]. 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 deployment to innovation system wasn't explored, Olsson et al. anticipate that the key actions required are the automatical data collection components and capability to effectively use the collected data.

TODO: Maybe something about continuous innovation? Brown, 1997, The art of continuous change: Linking complexity theory and time-paced evolution in relentlessly shifting organizations

-Connect CE, IES, Microsoft stuff, Lean startup stuff

"A detailed framework for conducting systematic, experiment-based software development has not been elaborated" [14].

A model of continuous experimentation has been created by Fagerholm et al. [14].

An experiment is essentially a procedure to confirm the validity of a hy-

pothesis. 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 listened to. 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" [6]. Most organizations have many ideas, but the return-on-investment for many may be unclear and the evaluation itself may be expensive [25].

Continuous deployment 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" [6]. 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 attempt to validate the hypothesis.

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" [6]. 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.

[13] Applied the continuous experimentation research to embedded software. "Improvement occurs in individual software parts, but the underlying design concept remain mostly unchanged [3]" "The experiment infrastructure allows developers to deploy new software and collect data how it behaves in areal-world settings being used by actual users. The infrastructure support deployment of software experiments and collection of data over-the-air on a scale comparable to the entire customer base". "The infrastructure supports with automated randomisation and factorial designs [6] sufficient to draw statistical conclusions from the experimental scenarios." "The experiment manager architecture, seen in Figure 3, supports the deployment of multiple experimental software parts to the same device and autonomously controls when to run which experiment, even allowing for local A/B-testing. Measurements and analysis is done on-board in real-time. The experiment scenario to be answered is implemented on the embedded device (i.e. how long does it take to . . .)" TODO: picture here

3.1.1 Experimentation planning

3.1.2 Experimentation stages and scopes

Fig. 6 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 [6]. 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 5: Scopes for experimentation [6].

3.1.3 Components in continuous experimentation

Kohavi et al. investigate the practical implementations of controlled experiments on the web [25], 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.

3.1.4 Data collection

"Active customer feedback is concerned with surveys and other mechanisms where the customer is aware that he or she is providing feedback. Passive feedback and usage data is collected while the customer is using the system. Examples include the amount of time a user spends using a feature, the relative frequency of feature selections, the path that the user takes through the product functionality, etc. The low cost and ease of data collection leads to the next major difference between IES-based and traditional software." [6]

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 [25]. The data should also be annotated to an identifier, such 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.

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.

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 [14]. 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.

TODO: adobe: Pipeline team: designer, developer, researcher, product manager [2]

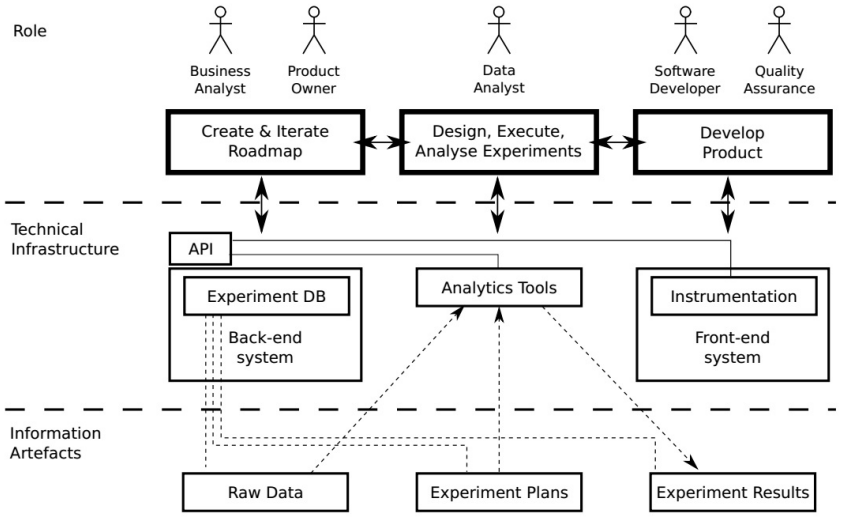

Figure 6: Continuous experimentation infrastructure [14].

3.2 Continuous deployment and continuous experimentation collaboration

As the experiments are run in a regular fashion, integrating experiments to the deployment pipeline should be considered. This requires changing the development process in such fashion that functionality is developed based on some actual data. The components required to support continuous experimentation include tools to assign users to treatment and control groups, tools for data logging and storing, and analytics tool for conducting statistical analyses.

3.3 Challenges regarding continuous experimentation

"Experimentation in practice is often limited to optimizing narrow aspecs of the front-end through A/B testing, and is not systematically applied" [6]

"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." [6]

"A final aspect is the avoidance of versioning of software. Traditionally, for many companies, every customer would ultimately have a unique configuration of the product with different versions of components making up the system. This adds a whole new layer of complexity to the already costly process of deploying new versions. In an IES environment, there is only one version: the currently deployed one. All other versions have been retired and play no role." [6]

"Innovation in large organization is often characterized by an enormous imbalance between the number of ideas that, informally or formally, exist in the organization and the number of concepts that are in fact tested with customers." [6]

"Traditional metrics such as the Net Promoter Score[Reichheld, F.F.: The One Number You Need to Grow] have been used for the last decade or more, but often fail to provide timely feedback during the development process as these are backward looking and focus on the entire product." [6]

"Most organizations have many ideas, but the return-on-investment (ROI) for many may be unclear and the evaluation itself may be expensive." [25]

TODO: possible challenges in Lean Startup: [28]

-Randomization algorithm and assignment method, data collection, storage, analysis [25]

Pitfalls:

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 [8].

-Roles -Environment -Deployment

3.4 (TODO:move&rename)Why is this interesting?

"Companies are increasingly transitioning their traditional research and product development functions towards continuous experiment systems" [30]. TODO check this

"The literature is increasingly recognizing the importance of continuous innovation" [40]

3.5 (TODO:move&rename)Existing case studies

Existing case studies regarding continuous experimentation have been conducted with varying aspects under concentration. Olsson et al. conducted a multiple-case study to explore companies moving towards continuous deployment and continuous experimentation [30]. 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.

Case study: stairway to heaven [30] -multiple-case study exploring companies moving towards continuous deployment -evolution phases, no continous experimentation transition defined -Continuous delivery transition with a highly distributed team -R&D as an experiment system

Fagerholm et al. analytically derive a continuous experimentation model and apply it in practice in a startup company [14]. The study introduces an initial continuous experimentation model,

Case study: building blocks [14] -analysis of a continuous experimentation model in the case company -design science research that derives a model and applies it in practice -initial continuous experimentation model -startup company

Case study [40] -continuous innovation "an experimental iterative process that operates successively to solve problems in markets characterized by turbulence, uncertainty and complex interactions" -continuous innovation, "the ability to renew the organization and to develop new products and business models, is central for most companies" -specifies in organizational characteristics -innovational culture aspects -Steiber [23] report on a study of the model of continuous experimentation followed by Google, analysing a success story of this approach.

Case study [4, 2] - Adams [1] present a case study on the implementation of Adobe's Pipeline, a process that is based on the continuous experimentation approach. - Innovation pipeline: 1 collect user problems 2 ID problems

worth solving 3 prototype possible solutions 4 live user testing + public releases 5 refine, pivot or kill the idea 6 build products that understand the user

Pipeline team: designer, developer, researcher, product manager

General principles: 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 -accept failure as part of our process not the end of it -fail often, but make mistakes on a small scale -always learn from what went wrong -don't fall in love. remain open to the pivot -don't let them ruin good with perfect

(Case: online experimentation at microsoft [26])

Motivation: -clearly defined in the B2B context -

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 [41]. The case in a case study is the situation, individual, group or organization that the researchers are interested in [35]. Kitchenham et al. found case studies important for industrial evaluation of software engineering methods and tools [23]. Runeson and Höst define it as a suitable research method for software engineering, since it studies contemporary phenomena in its natural context [36]. 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 [36]. 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 [35]. 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 [10]. 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 [36]. 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 [36]. Stake defines four ways to apply triangulation: Data triangulation, Observer triangulation, Methodological triangulation and Theory triangulation [39]. 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 [36]. As a data collection method, semi-structured interviews are common in case studies [36]. However, generalizing the results of a case study is often a subject of internal validity [24]. 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 [35].

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 [38]. 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 [36]. 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 [36]. 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 [36].

Qualitative data analysis [35]

Seaman has explored qualitative research in software engineering context, and he defines different ways to analyse qualitative data. [38] -Coding, extracting values for quantitative variables from qualitative data -Coding subjective data, select reference points for views, evaluate relability -Generation of Theory (GoT), extract from a set of field notes a statement or proposition supported in multiple ways by the data - – for use as hypotheses -GoT cross-case analysis: data divided into "cases": two groups based on some attribute and examine similarities and differences (devs / management?) -Confirmation of Theory: target: building up "weight of evidence"; triangulation; anomalies in the data -qualitative data is richer than quantitative data. use of qualitative methods increases – amount of information – diversity of information – 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 [36]. Both of these techniques can be exploratory and explanatory case studies [36].

Robson introduces common features of qualitative data analysis in a sequential list [35]. 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" [38]. 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 [35]. 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 [36].

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 [22]. Template analysis is a relatively structured process to analyse textual data, and has mostly been used to analyse data from individual interviews [21]. 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 [22]. 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 [22]. 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 [22]. 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 [22]. 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 [22]. 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 [22]. 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 [9]. 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 [16]. 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 is based on the critical realist perspective. Critical realism studies people's perceptions to gain understanding of the reality that exists beyond these perceptions [17]. 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 [12].

TODO: triangulation?

The logical starting point of this thesis was

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 applications of continuous experimentation are primarily executed in the B2C domain, often with a SaaS product. Examples are the Microsoft EXP platform [3] and Etsy []. The focus of this study is in the B2B domain, with two applications that are not used as SaaS.

The study is an exploratory deductive case study, which aims to explore how continuous deployment and continuous experimentation can be integrated to the development process. The study specifically aims to identify the main requirements, problems and key success factors with regards to these approaches. 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 deployment and continuous experimentaton also requires understanding the requirements of continuous deployment 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 design 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. Then interview is used to analyze and identify the pain points the company currently has in its development process. Then continuous deployment and continuous experimentation processes are viewed from the viewpoint of the case company, and possible pains identified. As a result, necessary restrictions and requirements encountered in the B2B domain are obtained.

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

Knowledge gap	Research question	The focus of analysis
a	RQ 1: What are the B2B specific challenges of continuous delivery and continuous experimentation?	Empirical case supported by conceptual analysis
a	RQ 2: How does continuous experimentation benefit the case company?	Empirical case supported by conceptual analysis
a	RQ 3: How can continuous experimentation be systematically organised in the B2B domain	Empirical case supported by conceptual analysis

Table 2: Research questions and research methods.

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 [24]. The two former will be discussed here, and the latter in the section "Frame of reference".

-configuration management: no employer designed and no formal education

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 the Development & Integration, which is further split into two sub-teams: Dialog and CDM. Both of the subteams develop a different product. The team is managed by a Concepting & Management team, which consists of the Product Owners of both software products, quality assurance and commercialisation experts.

The organisation of Steeri Oy is of a divisional type, with each business area forming independent teams based on the products and projects. The subteams of the first team under study have a common team leader, but different product owners and middle management. TODO: describe organisation structure

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 process. The unit of analysis is studied by focusing on interviewing individual members at different positions in the organisation. The purpose of the interview is to identify the pain points in the development process regarding continuous deployment and continuous experimentation.

Continuous deployment and continuous experimentation haven't been previously used by the case company.

In the beginning of the study, feedback was only collected in the form of bug reports from the customers. The bugs were then prioritized according to the importance, and added into the Trello workflow.

Under analysis is also the company's ways to interact with the customer.

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 the two products and development processes used are identified and compared. The general characteristics are as follows.

- Product description
- Product environment
- Product architecture
- Users of the software component
- Team composition
- Development practice
- Business model TODO: validate if this is confidental
- 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. Testing is used to.. TODO: explain the general characteristics

5.2.1 Dialog

The Dialog subteam focuses on developing a multi-channel 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 CRM 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 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.

Users of the software component. 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. 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. 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 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.

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

Acceptance testing. Acceptance testing is performed by the Concepting and Management team whenever a version is deployed to the customer environment.

Feedback collection. Feedback is collected from customers in retrospective meetings after the projects. Feedback is also received directly from customers via e-mail and meetings, but isn't collectively gathered in other ways. Feedback isn't automatically collected in any way.

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

Feedback and usage data usage. The feedback is used to improve the software product.

-java, javascript front? -frontend -binaries are built -jenkins deployed -which env decides -testing (written by coder, tested by uat?) -management

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 [27] 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 [27] 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.

Users of the software component. 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 4 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.

Business model.

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.

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

Acceptance testing.

Feedback collection.

Usage data collection.

Feedback and usage data usage.

5.3 Context

The general context of the 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 and a master data management product, while the other team consists of 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.

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. 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 at least the following information: A descriptive name Specifications that explain both the technical side and the business side Agilefant link four hour reporting

The backlog item is added to team backlog by selecting the “show in team backlog” -option.

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’s just a view of all “in team backlog” stories that are not yet completed and are assigned to the selected team.

Completed stories related to one team can be found from the report (link in team detail page) Team backlog items (stories) are prioritized and estimated once a week (Tuesday) and new stories will be moved to Trello when accepted by the development team.

Stories can be moved to Trello by project managers or product owners but they must be checked by team leader (Juha) who then moves 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 and 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 you have reviewed the pull request leave yourself 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: · 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 the 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.

The researcher is a part of the development team of the company, but the viewpoint ...

5.4 Methods

The primary source of information in this research are semi-structured interviews performed within the Development & Integration team and its management. TODO: business people? The interview consists of pre-defined themes as follows: (1) current development process, (2) current deployment process, (3) current interaction with customers, (4) problems and strengths in the current development process, (5) software product, (6) future ways with continuous deployment and continuous experimentation. Data triangulation will be implemented by interviewing multiple individuals. Methodological triangulation will be implemented by collecting documentary data regarding the development process.

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 cannot be properly supported with quantitative data. There is also a 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 interview rather than general opinions.

5.4.1 Data collection

The interview has a standardized set of open-ended questions TODO: validate. Leading questions are avoided on purpose, and different probing techniques such as "What?"-questions are used. Interviews were performed in the native language of the interviewee if possible, otherwise in English.

The interviews were performed once with every subject. Semi-structured interview with open questions allow deep exploration of studied objects [36]. 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 interviews are recorded in audio format, and then transcribed into text.

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 a single team split into two sub-teams, all participants of the team and its management team were 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. TODO: add more -Developers: Product and project context -technical details -Team leaders: process and project context -lean, agile, which practices are followed -Management: Company context -status of the organization, available resources in economy

5.4.2 Data analysis

In this case study the data analysis is performed with tabulation [36], where coded data is arranged into tables to get an overview of the data. The data is organized by ..TODO.

The data is analysed based on template analysis, which is a way of thematically analysing qualitative data [20].

Template analysis was chosen as an analysis approach because it organises and analyses the data according to themes. It is also particularly effective when comparing the perspectives of different participant groups [22].

Since the thesis focuses on multiple themes and the interview on multiple groups, template analysis was seen as a suitable approach.

TODO: Analysis is done [36] Coding Interview results are added to spreadsheet, with the participants grouped based on their teams. TODO: find analysis methods

"One example of a useful technique for analysis is tabulation, where the coded data is arranged in tables, which makes it possible to get an overview of the data. The data can, for example be organized in a table where the rows represent codes of interest and the columns represent interview subjects." [36] TODO: Tabulationia käytetään.

-roles (devs, business, PO's)

5.5 Limitations

-current state at steeri -short summary

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.

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 more common.

In B2B the customer is a company.

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,

6.1.1 Technical challenges

One of the main differences between B2B and B2C is the product 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. This introduces multiple challenges: accessing the environment, integrating the software to third party components and downtime.

In the case company, both of the products often run on the customers environment.

Specific customer environment, connection might require VPN -vpn user limit might be full

multiple customer environments with different configurations per customer -should the product be always deployed to every customer?

In the IT-departments, customers have to be informed on actions in their servers. There might be firewall configurations, required certifications and missing rights.

database needs to be versioned

the software might be connected to 3rd party applications or some other APIs, and if the API changes then these need to be configured as well

6.1.2 Procedural challenges

6.1.3 Customer challenges

Some customers of the case company hate new releases, since new releases occasionally contain new bugs.

Customers need to understand what continuous delivery is all about.

A customer might be doing a critical task with the software component, and therefore automatic version changes shouldn't cause downtime.

Some customers don't want UI changes, since they have been trained to perform certain tasks with a given UI. The customer might use the product once every two weeks and then get confused if the UI varies. Therefore the UI should have a stable consistency. -however, if modifications were made more often, it would lead to smaller modifications and it might reduce the customer hate towards new versions

UAT is often the norm and is required before anything is moved to production

The customer validates the functionality of the software.

Customers need to understand the idea of continuous delivery

customer often specifies the deployment date

customer might be using the software when the version is updated

========== Delivery B2C:

"Most customers of most products hate new releases. That's a perfectly reasonable reaction, given that most releases of most products are bad news. It's likely that the new release will contain new bugs. Even worse, the sad state of product development generally means that the new features are as likely to be ones that make the product worse, not better. So asking customers if they'd like to receive new releases more often usually leads to a consistent answer: No, thank you. On the other hand, you'll get a very different reaction if you ask customers next time you report an urgent bug,

would you prefer to have it fixed immediately or to wait for a future arbitrary release milestone?" [**?**]

Mindsets:

"if a change is supposedly side effect free, release it immediately. "

"The second shift in mindset required is to separate the concept of a marketing release from the concept of an engineering release. Just because a feature is built, tested, integrated and deployed doesn't mean that any customers should necessarily see it."

"Plus, you want to get good at selectively hiding features from customers. That skill set is essential for gradual roll-outs and, most importantly, A/B split-testing. In traditional large batch deployment systems, split-testing a new feature seems like considerably more work than just throwing it over the wall. Continuous deployment changes that calculus, making split-tests nearly free. As a result, the amount of validated learning a continuous deployment team achieves per unit time is much higher. "

6.2 Continuous experimentation: B2B challenges

6.2.1 Technical challenges

6.2.2 Customer challenges

In the B2B domain the typical characteristics is lower user amount as compared to B2C,

6.2.3 A/B testing

========== A/B testing B2B:

Many customers - A/B testing could be split such that A and B both are different customers. However, different customers often have different use cases.

CDM: No users, except for other systems Measurable metrics -performance Idea: A/B test different databases and measure performance No user interface

Dialog: User amount is very low, so there's no room for quantitative analysis Customers are very concerned about UI modifications UI is quite complicated, needs detailed use guide "In B2B we need more volume from customers, and a whole different skillset"

========== Experimenting B2B:

To collect data from customers, proper agreements might need to be made. Too few users for statistical importance Customers have to be informed for major changes UI changes have to be informed to the customers, and release notes are a must

CDM: Customers don't want downtime Features have been primarily created for customer needs, and new features might be hard to validate

with customers measurable metrics -performance -query amount

Dialog: Users perform routine tasks, and the UI should be stable such that the customers know how to do them. Measurable metrics don't include clicks or sales events. -one possibility is to measure the duration it takes to complete a routine task, but it is very hard -customer satisfaction, how happy the customer is and how easy the product is to use Some customers might use the product only a few months a year While theres not much statistical data, users can be tracked and they perform the same routine multiple times. If this time could be reduced, that would be benefitable to the customer.

========== Experimenting B2C:

========== Feedback collection currently:

Customer feedback is mainly received, not collected Product owners in constant touch with the customers E-mails In our products (CDM), the feedback loop is short and feedback is received when something has been done. Retro after project

========== Feedback collection B2C:

6.3 Continuous deliverys benefit to case company

6.4 Continuous experimentations benefit to case company

What kind of OEC can be picked? Which attributes can be improved?

Data-driven decisions

6.5 Implementing continuous experimentation as a development process

Interesting interview questions regarding this issue:

daily work routine weekly work routine current development model where to the development ideas come from current deployment process

6.6 Cross-case analysis

The simplest experiment cycle: 1.oec 2. 3.impl 4.analysis 5.steer

7 Discussion

This thesis was motivated by

7.1 Contribution

This thesis contributes to..

The contributions to .. are

7.2 Further research

References

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

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

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

[4] 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.

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

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

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

[8] 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.

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

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

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

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

[13] 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.

[14] 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.

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

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

[17] 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.

[18] 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.

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

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

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

[22] 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.

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

[24] 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.

[25] 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.

[26] 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.

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

[28] May, Beverly: *Applying lean startup: An experience report–lean & lean ux by a ux veteran: Lessons learned in creating & launching a complex consumer app.* In *Agile Conference (AGILE), 2012*, pages 141–147. IEEE, 2012.

[29] 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.

[30] 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.

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

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

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

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

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

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

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

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

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

[40] 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.

[41] 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?