Transitioning towards Continuous Delivery in the B2B Domain: A Case Study

Olli Rissanen[1,2], Jürgen Münch[1]

[1] Department of Computer Science, University of Helsinki, P.O. Box 68, FI-00014 University of Helsinki, Finland
[2] Steeri Oy, Tammasaarenkatu 5, 00180 Helsinki, Finland
olli.rissanen@aalto.fi,juergen.muench@cs.helsinki.fi

Abstract. Delivering value to customers in real-time requires companies to utilize real-time deployment of software to expose features to users faster, and to shorten the feedback loop. This allows for faster reaction times, ensuring the development is focused on features providing real value. Continuous delivery is a development practice where the software functionality is deployed continuously to customer environment. Although this practice has been established in some domains such as B2C mobile software, the B2B domain imposes specific challenges. This article presents a case study that is conducted in a medium-sized software company in the B2B domain. The results suggest that technical challenges are only one part of the challenges a company encounters in this transition. The company must also address challenges related to the customer and procedures. The core challenges are caused by having multiple customers with diverse environments and unique properties, whose business depends on the software product. Some customers also require to perform manual acceptance testing, which slows down production deployments. The benefits found from continuous delivery support the case company in solving many of its business problems. The company can expose the software functionality to the customers from an earlier stage, and better utilize customer feedback.

Key words: Continuous Delivery, Continuous Deployment, Development Process, B2B, Case Study

1 Introduction

To deliver value fast and to cope with the increasingly active business environment, companies have to find solutions that improve efficiency and speed. Agile practices [**?**] have increased the ability of software companies to cope with changing customer requirements and changing market needs [**?**]. To even further increase the efficiency, shortening the feedback cycle enables faster customer feedback.

Continuous delivery is a design practice aiming to shorten the delivery cycles and by automating the deployment process. It is an extension to continuous integration, where the software functionality is deployed frequently to customer

environment. In continuous delivery, the software is developed in a way that it is always ready for release. While continuous integration defines a process where the work is automatically built, tested and frequently integrated to mainline [**?**], continuous delivery adds automated acceptance testing and automated deployment.

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

This study is organized as follows. The first chapter is the introduction at hand. The second chapter summarizes 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 third chapter presents the research design: objectives, context and applied methods. The findings are then presented in the fourth chapter, organized according to the research questions. The fifth chapter interprets the main results. Finally, the sixth chapter summarizes the results of study and answers to the research question, discusses the limitations of the study and introduces further research avenues.

1.1 Background and Related Work

Continuous delivery is an extension to continuous integration, where the software functionality is deployed frequently at customer environment. While continuous integration defines a process where the work is automatically built, tested and frequently integrated to mainline [**?**], often multiple times a day, continuous delivery adds automated acceptance testing and deployment to a staging environment. The purpose of continuous delivery is that as the deployment process is automated, it reduces human error, documents required for the build and increases confidence that the build works [**?**]. Continuous delivery aims to solve the problem of how to deliver an idea to users as quickly as possible.

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

Continuous delivery differs from continuous deployment. Both continuous delivery and continuous deployment include automatic deployment to a stag-

ing environment. Continuous deployment includes deployment to a production environment, while in continuous delivery the deployment to a production environment is done manually. The purpose of continuous delivery is to prove that every build is deployable [**?**]. 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.

Olsson et al. investigate the transition phase from continuous integration to continuous delivery, and identify barriers that companies need to overcome to achieve the transition [**?**]. One such barrier is the custom configuration at customer sites. Maintaining customized solutions and local configurations alongside 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.

The practice of deploying versions continuously allows for faster customer feedback, ability to learn from customer usage data and to focus on features that produce real value for the customer [**?**]. 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 [**?**]. They also note that finding a pro-active lead customer is essential to explore and form a new engagement model.

2 Case study

2.1 Objective

RQ1: What are the B2B specific challenges of continuous delivery?
Software development practices and product characteristics vary based on the domain and delivery model. Typical B2C applications are hosted as Software as a Service (SaaS) applications, and accessed by users via a web browser. In the B2B domain, applications installed to customer environments are very common. The purpose of this research question is to identify the challenges faced in applying continuous delivery in the B2B environment. The research question is answered by conducting interviews to discover the current development process and its challenges in the case company, and using these findings and the available literature on continuous delivery to map the initial set of challenges these approaches will encounter in the case company. The available literature is used to provide a thorough understanding of continuous delivery as a whole, so that challenges can be identified in all aspects of the practice.

RQ2: How does continuous delivery benefit the case company?
To rationalize the decision to adopt continuous delivery in a company, the actual benefits to the business have to be identified. This research question aims to identify clear objectives for what is achieved by adopting continuous delivery. Sections of the interview aim to identify the current perceived problems of the

case company related to deployment, product development, collecting feedback and guiding the development process. These problems are then compared to the benefits of the approach found from the literature.

2.2 Research design

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

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

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

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

3 Results

The challenges regarding continuous delivery are 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 process and customer commitment.

3.1 Technical challenges

The technical challenges for continuous delivery are derived from the interviews. A part of the interview focuses on the current deployment process and customer interaction of the case company. The current deployment process, customer interaction and challenges related to them are then used as a basis for analysing challenges that influence continuous delivery.

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

Table 1. Technical challenges in continuous delivery.

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

The developers perceive automated testing and test environments to be the largest technical task. Since both of the case company's software products have been in development for a long time, building a viable test automation has many challenges with architectural decisions and available workload. The developers

state that building a sufficient test automation is a very laborious process especially due to the maturity of the software, and are concerned with the maintainability of the test suite. The management is not sure what to test with automatic acceptance testing to validate that a version is valid. Due to the current amount of time spent on manual testing, having an automated test suite is seen as very beneficial.

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

It is also common for B2B applications to have external components that have to be configured when the software is installed or the APIs to these components changed. The configurations for these external components either have to be manually updated, or automated as well. In the case company, both Dialog and CDM are integrated to multiple external components. Currently upon version change these integrations are manually updated when necessary. In continuous delivery, changes that modify the API and therefore break these integrations might have to be manually updated.

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

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

3.2 Procedural challenges

The procedural challenges are analyzed based on the development process documentations of the company and the interviews. In the interviews, a section is

dedicated to the current development process of the case company. The development process and its challenges are then used to analyse and identify challenges that influence continuous delivery.

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

Table 2. Procedural challenges in continuous delivery.

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

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

The software applications in B2B often are large and modular applications, as is the case in the case company. CDM is a software component that is compiled from multiple projects. As the deployment process is currently manually

triggered by first releasing a version, a suitable time can be chosen each time. With continuous delivery the deployment process is triggered automatically, and with a modular project each module has to be in the correct state in order to produce a coherent version. The point when a deployment is triggered has to be designed to maintain the integrity of the application.

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

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

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

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

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

Due to increased developer responsibility and varying interval of version updates continuous delivery causes, a team leader express concern that the delivery

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

3.3 Customer challenges

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

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

Table 3. Customer challenges in continuous delivery.

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

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

While user experience could be improved in the long run, customers might feel frustrated when they have to learn to work with modifications. The interviewees have ideas on how the reluctancy can be reduced. If modifications were made more often, it leads to smaller and less notable modifications which might reduce the customer reluctancy towards new versions. When customers become

accustomed to the model of continuous version updates, the reluctant attitude might change. Customers also need to understand the concept of continuous delivery, and that the features are eventually implemented to address customer needs and to improve the user experience in the long term.

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

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

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

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

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

The acceptance testing is performed in varying ways. In Dialog context, some customers require to perform manual acceptance testing in a user acceptance testing environment before the product can be deployed into production. Other customers trust the developers to perform the acceptance testing, and deployment to production can be negotiated without a build necessarily requiring to pass the user acceptance testing phase. In CDM context all of the customers perform manual acceptance testing in the user acceptance testing environment. The technical implementation therefore should make it possible to continuously deploy versions to the user acceptance testing environment, and by the push of

a button to the production environment. However, if the versions are deployed to user acceptance testing environment very often, customers might feel encumbered by the amount of required testing. The customers also have to be informed whenever a new version is available to the user acceptance testing environment.

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

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

3.4 Benefits to the case company

This section addresses the RQ2 (i.e. How do continuous delivery and continuous experimentation benefit the case company?). A major problem found in the interviews is that currently the reliability towards new versions is low. The low reliability both increases customers reluctancy towards version updates, and increases the amount of user access testing that is performed after version release. This makes the duration of building a new version relatively long and unstable, since the bug amounts differ upon each version deployment. The version releases are also often performed in a hurry, which also increases the error rate. The low reliability is caused by multiple reasons: features are occasionally forgotten and not tested, manual builds introduce human error factor, the knowledge of manual steps required for a succesful build is not shared and compiling modular products is hard.

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

The manual build process involves human error factor, which in the case company is the primary cause for the majority of bugs in new versions. One

example of such an error is using the wrong commands to compile and package the project, which causes wrong binaries getting into the new version. Combined with the lack of comprehensive automated testing, each deployment is a new error-prone experiment. Important factors increasing the rate of human error are partially missing documentation and the complexity of the build process. According to a developer:

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

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

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

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

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

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

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

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

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

4 Discussion

The study investigates extending the development process towards continuous delivery to advance the company towards real-time value delivery. The research questions address the challenges faced in this transition, and the benefits found from this practice.

The results suggest that the challenges faced in continuous delivery in the B2B context are multidimensional, and related to the technical, procedural and customer aspects. The major difference a company operating in the B2B domain

faces in the transition as compared to the B2C domain is that there are plenty of customers with unique properties, whose business relies on the software. The primary issues causing these challenges are therefore the diverse customer owned environments and the importance of the software product for the customer.

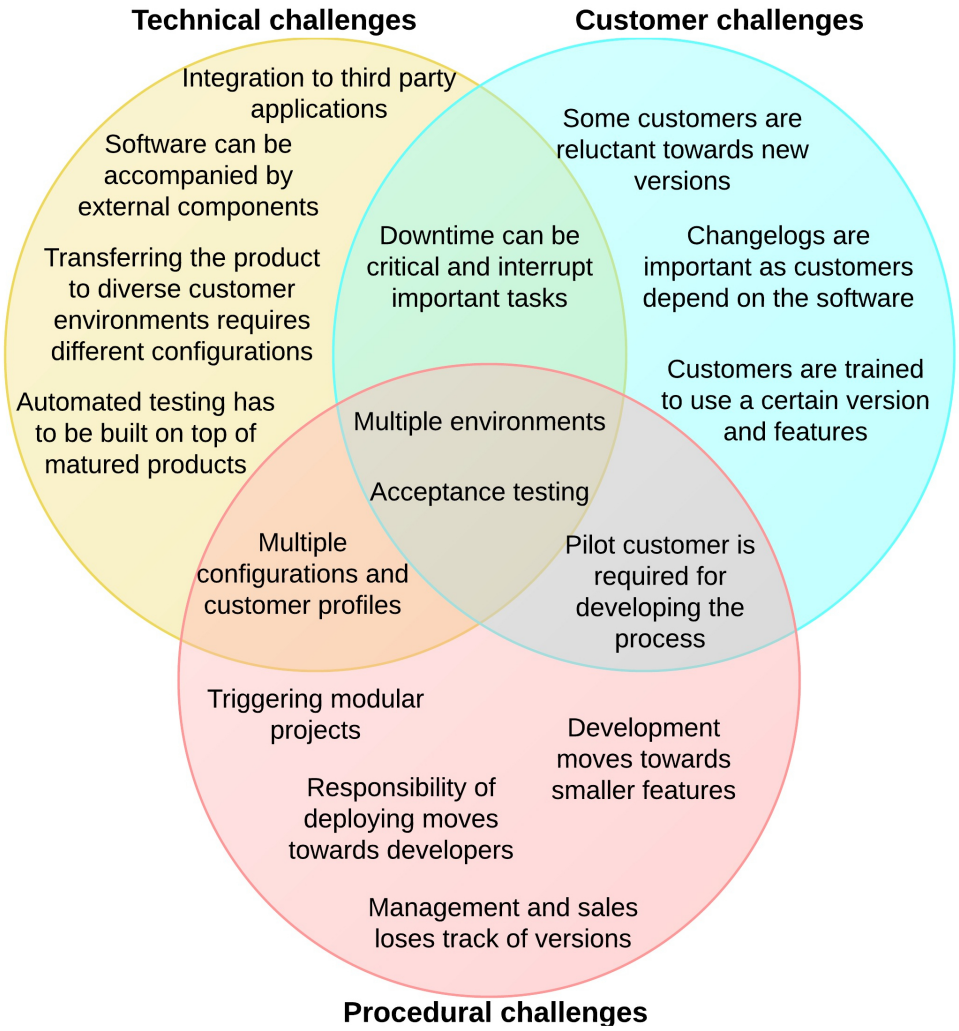

Fig. 1. Case company's challenges in continuous delivery.

Fig. 1 visualizes the challenges the case company faces in transition towards continuous delivery. Multiple challenges are related to two or more aspects, and the problems affecting all aspects can be seen as the core challenges. Acceptance testing is related to all aspects since customers want to perform acceptance test-

ing with new versions, automated acceptance testing has to be implemented and the user acceptance testing is required before a production release can be made. Another challenge related to all aspects is the diversity of customer environments. It affects the technical implementation, as software has to be transferred to diverse environments. The procedural challenge is that the software has to be deployed multiple customers, and it has to be decided whether each version is always released to every customer.

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

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

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

5 Summary

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

The focus of this study was not solely on the technical aspects of these development processes, as research already exists on these subjects [**?**, **?**, **?**]. Rather, the study focused on identifying the challenges in the transition towards these processes as a whole. The analysis of continuous delivery and continuous experimentation proceeded from higher-level considerations of requirements and benefits towards more detailed focus of applying continuous experimentation

in B2B context. The findings provide insights into the challenges a company faces in the transition in this domain. In addition, the findings also compare the development models from the viewpoint of two different software products, providing further analysis on software product characteristics that affect these models.

This study has identified the main requirements a company operating in the B2B domain has to address when applying continuous delivery and continuous experimentation. In the case of continuous delivery, the challenges can be divided into technical challenges, procedural challenges and challenges related to the customer. These challenges are mostly caused by having multiple customers with diverse environments and unique properties, whose business depends on the software product. While continuously deploying versions to a user acceptance testing environment requires a company to address multiple challenges, continuously deploying to production is even more difficult. The reason for this is that some customers want to perform manual acceptance testing before production releases can be made.

The benefits of these practices matched to many problems found in the case company, and a company operating in similar domain with similar products can use them as a basis when considering applying these practices. The most important challenges in the case company were found to be related to the deployment process and guiding the development of the software products. By utilizing continuous delivery, the case company can solve problems such as long feedback cycles, low reliability in new versions and high amount of resources required for releases.

5.1 Limitations

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

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

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

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

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

References

1. Smith, T.F., Waterman, M.S.: Identification of Common Molecular Subsequences. J. Mol. Biol. 147, 195–197 (1981)
2. May, P., Ehrlich, H.C., Steinke, T.: ZIB Structure Prediction Pipeline: Composing a Complex Biological Workflow through Web Services. In: Nagel, W.E., Walter, W.V., Lehner, W. (eds.) Euro-Par 2006. LNCS, vol. 4128, pp. 1148–1158. Springer, Heidelberg (2006)
3. Foster, I., Kesselman, C.: The Grid: Blueprint for a New Computing Infrastructure. Morgan Kaufmann, San Francisco (1999)
4. Czajkowski, K., Fitzgerald, S., Foster, I., Kesselman, C.: Grid Information Services for Distributed Resource Sharing. In: 10th IEEE International Symposium on High Performance Distributed Computing, pp. 181–184. IEEE Press, New York (2001)
5. Foster, I., Kesselman, C., Nick, J., Tuecke, S.: The Physiology of the Grid: an Open Grid Services Architecture for Distributed Systems Integration. Technical report, Global Grid Forum (2002)
6. National Center for Biotechnology Information, http://www.ncbi.nlm.nih.gov