Exploring the Encounter of Continuous Deployment and the Financial Industry

FELIX FRIE
GUSTAV HAMMARLUND

Master of Science Thesis
Stockholm, Sweden 2016
Mötet mellan Continuous Deployment och finansbranschen

FELIX FRIE
GUSTAV HAMMARLUND

Examensarbete
Stockholm, Sverige 2016
Exploring the Encounter of Continuous Deployment and the Financial Industry

Felix Frie
Gustav Hammarlund

Master of Science Thesis INDEK 2016:82
KTH Industrial Engineering and Management
Industrial Management
SE-100 44 STOCKHOLM
Mötet mellan Continuous Deployment och finansbranschen

av

Felix Frie
Gustav Hammarlund
Abstract
The digitisation of the financial markets has led to IT becoming a vital part of financial institutions. The principles and practices of Continuous Deployment (CD) are utilised to increase innovation through flexibility and swiftness at many IT companies. This thesis explores the encounter of CD and the financial industry through participant observations and semi-structured interviews with developers.

We find in our study that practitioners in the financial industry use practices that are part of a CD process. The specialisation of the systems that is evident in the industry could be considered a barrier for the adoption of a CD process. However, improved transparency that may come as a result of CD is well aligned with the demands that are evident in the industry. Furthermore, the requirement for code reviews might impact the ability to attain a continuous process, as it must be a manual.

Key-words: Continuous Deployment, Continuous Delivery, FinTech, Code Review, Innovation, Regulated markets
Sammanfattning
Digitaliseringen av de finansiella marknaderna har lett till att IT fått en allt större roll i den operativa verksamheten hos finansiella institutioner. Flera mjukvarubolag nyttjar den nya metodiken Continuous Deployment (CD) för att erhålla en ökad flexibilitet och snabbhet i sin utveckling. Denna studie undersöker mötet mellan CD och finansbranschen genom deltagande observationer och semi-strukturerade intervjuer.

Vi noterar i vår studie en utbredd användning utav verktyg och principer som utgör delar av CD hos finansiella institut. Den specialisering av system som är synlig på de finansiella marknaderna kan göra att automatisering, som är en del av CD, försvåras. En ökad transparens i utvecklingen, vilken CD kan ge, torde vara önskvärd hos finansiella institut då den kan bidra till att uppfylla de krav på ansvar och spårbarhet som de står inför. Dock går kravet på kodgranskning mot ett införande av CD, då det introducerar ett manuellt steg i utvecklingen.

Nyckelord: Continuous Deployment, Continuous Delivery, FinTech, Code Review, Innovation, Reglerade marknader
Acknowledgment

First and foremost we wish to thank all the persons we have come across on our journey in writing this thesis. Primarily the participants of our interviews but also all the people we have interacted with during our observations. We have been welcomed with open arms in whatever setting, something that have been encouraging and helpful for us in our research. We would also like to aim our gratitude to our parents without whom this thesis would never have been written. And our friends. Who in times of stress, found time to read our words with care and (hopefully) found all the mispelings and mistake’s we have introduced to the world.

Last, and far from least, we thank our supervisor Charlotte Holgersson. Her precise guidance and deep knowledge of academia has been invaluable. She has helped us with a cheer spirit and in the process made us aware of important questions, not only about academia, but also about society. We will never forget her nor what she has done for us.
Contents

List of Figures vii

List of Tables ix

1 Introduction 1

1.1 Background 2

1.2 Problematisation 5

1.3 Purpose 5

1.4 Research Questions 5

1.5 Delimitation 6

2 Theoretical Framework 7

2.1 Origins of Continuous Deployment 8

2.1.1 Agile Methodologies 8

2.1.2 XP 9

2.1.3 Critique against Agile and XP 11
2.2 Continuous Integration ........................................ 12
2.3 Continuous Deployment ...................................... 14
   2.3.1 Continuously Deploying Software ....................... 15
   2.3.2 Organising Continuous Deployment ..................... 18
   2.3.3 Between Benefits and Challenges of CD ............... 19

3 Empirical Setting: The Financial Sector 23
   3.1 Technology in Finance ..................................... 24
   3.2 Subject companies ........................................... 25

4 Method .......................................................... 27
   4.1 Research Design ............................................. 28
   4.2 Participant Observation ..................................... 29
      4.2.1 Going Native ............................................. 30
   4.3 Semi-structured Interviews ................................. 31
   4.4 Sampling .................................................... 33
   4.5 Anonymity .................................................... 34
   4.6 Validity & Reliability ..................................... 34

5 Results ............................................................ 37
   5.1 Automation .................................................... 38
   5.2 Fault Tolerance .............................................. 39
List of Figures

2.1 The CI process, development and integration are done continuously and iteratively .................. 12

2.2 Illustration of traditional, CI and CD processes. ........... 15

2.3 The CD process. It provides developers with feedback that serves as guidance in what features to develop next. ....... 16

6.1 The contradicting demands on financial systems. A high performing and stable system tend to lack functionality. A high performing system with high functionality lacks stability and a stable system with high functionality lacks performance to be competitive. ............................................. 51
List of Tables

5.1 Key aspects of the results. .................. 46

B.1 An enumeration of the participants .......... 82
Chapter 1

Introduction

This first chapter will explain the setting of technology in the financial industry. It will also briefly describe the evolution of traditional software development into more agile and continuous practices. The problem investigated in this report is pinpointed in the problematisation section followed by the purpose of our study. Subsequently, we present our research questions which serve to operationalise our purpose. Lastly, we specify the delimitations of this thesis.
1.1 Background

This thesis explores the encounter of the novel software development process Continuous Deployment (CD) in the regulated financial industry. The industry has undergone considerable change as a result of the advancement in Information Technology (IT) (Merton, 1995). The digitisation, which also can be recognised throughout other industries such as the music industry and retail, has been ongoing in the financial sector for the last decades. As a result of this digitisation the financial markets have become a complex network of digital systems (Kauffman, Liu, and Ma, 2015; Diaz-Rainey, Ibikunle, and Mention, 2015). The previous, rather manual, stock market where traders interacted on a trading floor, communicating and negotiating on stock prices, has shifted to a marketplace characterised by automation, where the major part of orders are executed by digital systems. This digitisation started in the equity market but has now found its way into an increased number of asset classes (Kauffman, Liu, and Ma, 2015; Diaz-Rainey, Ibikunle, and Mention, 2015).

Due to the digitalisation the size of the technology divisions at financial institutions has increased. These divisions work at the technological frontier in a multitude of computational technologies to develop competitive systems (Kauffman, Liu, and Ma, 2015). Developing systems with superior performance is crucial in order to gain competitive advantage in the financial industry, as it grants the firms abilities not available for other markets actors.

The financial institutions encounter competition in regards to technology development from smaller, third-party, software firms within the financial technology (FinTech) industry whom are specialised in developing technologies and that have organisations and processes that are designed to promote software development. These firms are very different from traditional financial institutions since they originated with software development as their core competence. Contrary to FinTech firms the financial institutions have
problems to foster software development that is cost effective and successful (Kauffman, Liu, and Ma, 2015; Diaz-Rainey, Ibikunle, and Mention, 2015).

Other large IT companies have a similar need for innovative software development to gain competitive advantage. An example is Facebook whom have adopted CD practices with fortunate outcome (J. Bird, 2015). The development processes at financial intuitions, in contrast to the processes at other IT firms, are monitored by external regulatory assessors which limits their liberty in utilisation of tools and practices (Feitelson, Frachtenberg, and K. L. Beck, 2013).

As software is integrated into more parts of everyday life, academic and industrial efforts have gone into increasing the effectiveness of Software Development Life-Cycles (SDLC) (Highsmith and Cockburn, 2001; Davis, Bersoff, and Comer, 1988; Boehm, 2006). The increased rate of change in the software industry, allowing less time for planning, specifying requirements and documentation (Boehm, 2006; Dybå and Dingsøyr, 2008), has led to the emergence of agile software development methods (Boehm, 2006; Dybå and Dingsøyr, 2008; Highsmith and Cockburn, 2001). The term agile does not imply a specific set of processes, but instead that the processes utilised are flexible and quick (Dybå and Dingsøyr, 2008; Highsmith and Cockburn, 2001).

One of the most commonly practiced and researched agile software development methodology is Extreme Programming (XP) (Dybå and Dingsøyr, 2008). XP popularised the concept of Continuous Integration (CI) (Claps, Svensson, and Aurum, 2015; Dybå and Dingsøyr, 2008; Fowler and Foemmel, 2006) to facilitate a high rate of innovation and to minimise the risk of introducing code that would cause bugs, unwanted errors, in the program or system. Today, many tech companies use CI along with the concept of CD (Hüttermann, 2012). These concepts will briefly be handled here and more thoroughly in chapter 2.

CI originated from large software development projects which required many developers to, at some point, integrate their code into the main code base, often called the mainline (Fowler and Foemmel, 2006). CI implicates
that tests are conducted both locally on the developers’ machines and continuously on an integration server, to ensure that new additions of code builds together with the mainline (Fowler and Foemmel, 2006). If the build breaks, it will be easier to fix as the differences would be small and developers can fix the build locally. Thus always keeping the code clean and in a buildable state. To prevent incidents a practice of running a number of tests prior to integration is employed. This requires the developer to synchronise with mainline and pass the tests before integrating (Duvall, Matyas, and Glover, 2007; Fowler and Foemmel, 2006). Some bugs are however difficult to identify with tests and might not be visible until the code reaches production environment.

CD builds on the practices of CI. In addition to continuously integrating changes, the software is continuously shipped to live production systems where the code is exposed to users (Claps, Svensson, and Aurum, 2015; Olsson, Alahyari, and Bosch, 2012). This process aims to make the entire SDLC process continuous, from code being written to it being deployed in production (Leppanen et al., 2015). This allows for shorter feedback loops for customer feedback and also shorter lead times for bug fixing and feature releases (J. Bird, 2015; Claps, Svensson, and Aurum, 2015; Leppanen et al., 2015).

CD practices was originally mostly utilised by start-ups, but is today used by big successful IT-companies such as Google, Netflix and Facebook (J. Bird, 2015). The aforementioned practices allow these companies to develop software swiftly, further accentuating that software is tested continuously during development. They are not hesitant to ship updates early and flush out bugs in a production environment, as the practice is focused on minimising the fear of failure and viewing failure as something positive (J. Bird, 2015). Failure is a learning opportunity and capital for allowing a high paced innovation process.
1.2 Problematisation

It is often argued that large financial institutions have to increase their agility in software development to remain competitive in the increasingly technical business environment. The principles and practices of CD are utilised to increase innovation through flexibility and swiftness of software development at many successful IT companies. The requirements put on software development life-cycles in the financial industry differs greatly from requirements in other industries as they are exposed to regulatory assessment.

1.3 Purpose

This thesis will investigate the encounter of Continuous Deployment practices and the development of software in the financial industry.

1.4 Research Questions

- **Main RQ**: How does Continuous Deployment practices comply with the development of technology used in the financial industry?

- **Sub-RQ1**: What barriers and facilitators to a Continuous Deployment process exists in the financial industry?

- **Sub-RQ2**: How can the Continuous Deployment process be modified to cope with the requirements of the financial industry?
1.5 Delimitation

This thesis will focus on the requirements put on development in the financial industry. Furthermore, we have chosen to take the viewpoint of a third party financial technology provider, about to be exposed to such requirements, as it highlights the stress they put on its SDLC model. Although the understanding of how a CD model works in practice is highly relevant for us, we do not aspire to provide any practical guidelines. Instead we will highlight the opportunities and limitations that the regulated environment imposes on software development for large scale financial institutions.
Chapter 2

Theoretical Framework

This second chapter will present the existing body of knowledge on the field of CD. We investigate the origins of CD in the fields of agile software development and take the route through XP and its practice of CI which makes the foundation for the theories of CD. Lastly, the state of the art research regarding CD is presented.
2.1 Origins of Continuous Deployment

The roots of CD are found in agile software methodologies (Claps, Svensson, and Aurum, 2015; Rodriguez et al., 2016; Olsson, Bosch, and Alahyari, 2013; Olsson, Alahyari, and Bosch, 2012). To gain a better understanding of CD, we start with building a foundation in agile methodologies and establish a view of the research of the field.

2.1.1 Agile Methodologies

The concept of agile software development can be traced back to various methodologies used in the early 1990’s (Larman and Basili, 2003), its values were formalised in 2001 by various practitioners in what is known as the agile manifesto (Fowler and Highsmith, 2001):

"We are uncovering better ways of developing software by doing it and helping others do it. We value:

- Individuals and interactions over processes and tools.
- Working software over comprehensive documentation.
- Customer collaboration over contract negotiation.
- Responding to change over following a plan."

- *The Agile Manifesto* (Fowler and Highsmith, 2001)

The agile movement is a reaction to the notion that traditional software development methods are too static and should strive to be more agile (Boehm, 2002). Agility is about removing the slowness that is associated with traditional software development to achieve speed and flexibility and thus allow for adjustments to changing user behaviour and environments (Dybä and Dingsøyr, 2008).
Dybå and Dingsøyr, in their systematic literature review of empirical publications on agile methodologies (Dybå and Dingsøyr, 2008), describes the main agile development methods to be Crystal methodologies, Dynamic Software Development Method (DSDM), Feature-driven development, Lean software development, SCRUM and XP. In their study they found that of the methodologies described above, XP was by far the most researched. They also concluded that for agile software methodologies the current state of the research to be nascent, implying that there is not yet a clear theoretical framework for agile software development in general. What exists are lessons learned type studies that does not maintain a high degree of scientific quality. Furthermore they stated that specific practices such as pair programming, one of the practices advocated by XP among others, is mature as it has received much attention.

In the paper ‘A decade of agile methodologies: Towards explaining agile software development’, which Dingsøyr co-authored (Dingsøyr et al., 2012), they, again, argued the importance of providing a rigid theoretical framework for further scientific progression in the field of general agile software methodologies. We acknowledge the limitations in current agile literature. To gain a better understanding of how agile methodologies are put into practice we continue by examining the specific literature on the most frequently researched agile methodology, XP.

2.1.2 XP

Initially in an XP project the task in its whole is divided in separate tasks called stories which correlates to features desired by the customer (K. Beck, 2004). Stories are ranked for their time to complete and importance. Iteration begins with the customer selecting a minimal set of the most valuable stories that will be the initial focus. The stories are divided in separate tasks that programmers then sign up for (K. Beck, 2004). A story is considered finished when it is deployed in production. After completion of a story the next most important story is selected and iteration continues. Beck describes
this process as turning traditional software development sideways, doing the steps of planning, analysing, integrating, testing and deploying for each individual story. This can be done as the cost for changing software is low if done early (K. Beck, 2004; K. Beck, 1999).

In order to get an understanding of how an agile methodology is applied, we describe Beck’s summary of the main practices of XP (K. Beck, 1999), each practice is highlighted in italic below.

In the *Planning game* it is the customers responsibility to decide what stories should be developed, based on estimates from the developers. *Small releases* is the practice of putting each iteration into production as early as possible. As each story is completed it is deployed long before the entire project is completed. A *Metaphor* for the system should be created to facilitate communication with customers. The system should also be of *Simple design* so that it is possible to easily test and communicate the code among developers. *Tests* should be designed frequently by the developers along with functional tests designed by the customers. The entire test suite should always run and pass. *Refactoring* should transform the design of the system while ensuring that all tests are running and are being passed. *Pair programming* is the concept that all production code should be written in pairs. *Collective ownership* of the code, meaning that every developer should be able to improve any part of the code if they are required. *40-hour weeks* should be the norm for every developer, overtime indicates underlying problems that must be addressed. *Open workspace* for better communication amongst the developers and encourage collaboration. *Continuous integration* is the practice that new code should be integrated with the code base at most within a few hours. All of this are *Just rules* and guidelines. Everyone should strive to follow the rules but be aware that circumstances might require breaking them (K. Beck, 1999).

By utilising XP for software development the value of software projects are enhanced through frequent feedback from the customer along with the emphasis on testing, simplicity and incremental changes (K. Beck, 2004). As will be evident, many of the practices described above are found in CD.
2.1.3 Critique against Agile and XP

Hikka, Tuure and Rossi empirically studies two cases in their paper (Hilkka, Tuure, and Rossi, 2005) and concludes that agile software development in general and XP in particular are not new and have been around since the 1960s. They argue that what XP does is essentially capture and formalise practices by talented individuals and teams (Hilkka, Tuure, and Rossi, 2005). As the agile field stems from lessons learned type studies and lacks empirical studies we believe that this critique is not XP specific but relevant for agile methodologies in general (Dybå and Dingsøyr, 2008).

Another critique of agile development and XP is that it misses out key engineering elements such as planning and analysing in their software development approaches (Boehm, 2002; Keefer, 2002; Paulk, 2001). With rapid planning there is a sense of solving problem as you go, which might lead to reduced quality in the final product as well as relying too heavily on talented individuals instead of being successful methods on its own (Paulk, 2001; Keefer, 2002). One can trace this back to the warrants for empirical studies and a more formally defined framework as described in the previous section (Dybå and Dingsøyr, 2008; Dingsøyr et al., 2012). Keefer identifies that the customer requirements often are a lot more complex than XP assumes (Keefer, 2002). This forces the practitioners to rely on implicit knowledge, which may lead to problems. Although Keefer’s paper heavily lacks scientific backing we recognise the required implicit knowledge as a hindrance many agile methodologies fail to solve.

The eligibility for agile methodologies in large scale software development projects has frequently been questioned and is handled in Dybå and Dingsøyr’s literature review (Dybå and Dingsøyr, 2008). They discuss that it is likely the case that XP is harder to implement in complex organisations although the adoption of XP in different organisational settings is possible (Dybå and Dingsøyr, 2008) and dependant on how interwoven software development is in the organisation.
2.2 Continuous Integration

When the number of developers concurrently working on a project are increased, the integration problems are amplified. As each developer finishes their task, they need to make their piece of code work together with all others’ code to create a functioning program. In more traditional, waterfall, projects this has been considered a separate step to be handled in the end of the project, when all developers have finished their contribution (Fowler and Foemmel, 2006). One drawback of this is that integrating all code at once can be problematic and potential problems are not discovered until late in the software development process (Fowler and Foemmel, 2006). CI strives to minimise the impact of this step by executing integration frequently throughout the development process.

Duvall, Matyas and Glover describes CI in their book “Continuous Integration - Improving Software Quality and Reducing Risk” (Duvall, Matyas, and Glover, 2007) which we have used as our main source for understanding the practices and principles of CI. Although the book is more intended for practitioners than for academia, we believe it to be integral in obtaining a deep understanding of how CI should be used. This view has been complement by turning to articles of more scientific nature to form a critical viewpoint of the subject.

![Figure 2.1: The CI process, development and integration are done continuously and iteratively](image)

The ambition of CI is to make integration a ”nonevent” as Fowler first described it (Fowler and Foemmel, 2006), a term also used by Duvall, Matyas and Glover (Duvall, Matyas, and Glover, 2007). Duvall, Matyas and Glover describe that this is achieved with CI by integrating often and early during the development, as we have illustrated in figure 2.1. This allows for shorter
feedback loops that enables the developers to assess the integrated code at any given point and reduces the time between a bug detection and fixing it. Furthermore, as CI often is achieved through automation, it reduces the amount of manual labour in a project (Duvall, Matyas, and Glover, 2007). In their book Duvall, Matyas and Glover covers the practices for, according to them, successfully implementing CI along with popular tools used. We will extract the key principles and practices, then presenting the basis for our understanding of what CI is.

Duvall, Matyas and Glover states that CI is a practice and not a methodology, meaning that it can be used together with XP or any other software development method as it complements other software development practices. However, it is considered to work best with an agile methodology (Duvall, Matyas, and Glover, 2007; Fowler and Foemmel, 2006). The following description of CI is based on Duvall, Matyas and Glover's book.

They describe that in CI development is done by having every developer write their code in small chunks that, when done, gets integrated into a main code base that contains the super-set of all developers finished code, called the repository. Thus there is the notion of a pre-integration state when a developer has a local copy of the repository along with the changes to be integrated. When the chunk of code is finished the developer begins the automated integration process.

The integration process begins with the developer sending his code change to the repository where his changes are incorporated into the main code base. The code in the repository is then compiled, which means that the code is turned into an executable program, and the compiled code is tested for basic functionality. If the compilation or the basic functionality tests fail the repository reverts the new code addition and goes back to the last functioning state. The developer is notified and can correct the changes on his local machine. If the compilation and the basic functionality tests passes further testing, that take more time, can be performed, such as performance testing. If the heavier tests fail, the developer is again notified and the repository can be reverted to a previous version. We recognise that this
Duvall, Matyas and Glover states that CI reduces risks by being a "safety net" to the code base by continuously checking that all software compiles and that tests are passed. It also reduces the amount of repetitive tasks involved with checking for defects in code additions, thus freeing time for developers to focus on development of new features. Furthermore they argue that CI improve the ability to assess the current state of the software as it is always in a updated, runnable state. This enables for better informed decisions and creates confidence in the development team. Also, it generates software that at any given point is ready for deployment (Duvall, Matyas, and Glover, 2007).

2.3 Continuous Deployment

CD extends the CI practice of continuously integrating code by including the entire development process, from developing code until its shipped to a customer (Neely and Stolt, 2013; Olsson, Bosch, and Alahyari, 2013). The process aims to increase the deployment frequency to production environment and thus increasing the ability to gain experience from that environment swiftly (Neely and Stolt, 2013; Olsson, Bosch, and Alahyari, 2013). However, there is no formal definition of CD present in current literature (Rodriguez et al., 2016). What exists is a common understanding that CD is a practice that organisations use to deploy software to customers as often and as fast as possible after new functionality has been produced (Rodriguez et al., 2016). We illustrate traditional software development along with a CI and a CD process in figure 2.2.

We will cover the existing body of knowledge on the topic of CD by first presenting what previous research has found to constitute a continuous process of deploying software. We have separated this presentation into deployment and organisation. Integration will not be presented here, although
it is an integral part of CD, as the practice of integration is handled in the previous section (2.2). Following the sections describing CD, we will position ourselves to the body of knowledge. We will discuss the framework that CD constitutes, where CI is included, and relate it to our purpose.

### 2.3.1 Continuously Deploying Software

Building on the concepts of testing and integration present in CI, automation plays an important part in CD (Neely and Stolt, 2013; Schermann et al., 2016). The CD pipeline ensures that the quality of the software is continuously assessed by the execution of automated tests without tampering with the ability to release often (Olsson, Bosch, and Alahyari, 2013). By releasing often, the set of features or changes deployed to customers is smaller, thus allowing for feedback mechanisms to guide the design and development (Olsson, Bosch, and Alahyari, 2013). Furthermore, defects are minimised as the batch size decreases for each release, thus increasing the ability to focus on the cause of potential failures when they arise (Neely and Stolt, 2013). In figure 2.3 we illustrate the different steps of a generic CD process.

The higher frequency of releases in a CD process reduces the stress on operators during each release since they get to exercise the upgrade procedure more often (Neely and Stolt, 2013). Releasing often requires the release process to be lightweight with minimal manual administration (Agarwal, 2011;
Neely and Stolt, 2013). Manual actions are considered to decrease the reliability and comparability of the process and introduces the possibility of careless mistakes (Goodman and Elbaz, 2008). Neely and Stolt highlights the need for automation in CD by claiming that automating the narrowest manual bottlenecks is one of the first steps that should be taken when adopting CD (Neely and Stolt, 2013).

Achieving CD should not affect quality negatively, but rather reduce the risk of each release (Agarwal, 2011; Claps, Svensson, and Aurum, 2015). An increase in quality has been noted with increased release frequency, this is argued to be a result of the increased transparency and overview that is gained from frequent releases (Rodriguez et al., 2016).

The practice phased deployment is a central part of CD. It refers to only exposing a selected flow of customers to a new release (Schermann et al., 2016). It is used to isolate the impact of bugs that could potentially be included in the release to selected users and to enable immediate feedback from that set of users (Rodriguez et al., 2016). Facebook uses this to gain
feedback from employees before deployment to production where it is exposed to customers (Feitelson, Frachtenberg, and K. L. Beck, 2013).

Phased deployment is often coupled with the use of a rollback procedure which works as a failover strategy if users notice problems (Rodriguez et al., 2016). There is then a defined rollback procedure that dictates the actions and management of flow to be taken in case of failure. This practice is often, fully or partially, automated in a developed CD process (Neely and Stolt, 2013).

Software architecture should strive to separate the system into parts that are as independent as possible. This is referred to as the system being separated into modules and loosely coupled (Olsson, Alahyari, and Bosch, 2012). Companies should employ micro-services instead of a monolithic architecture to be able to utilise CD (Schermann et al., 2016). The architecture needs to enable rollback of releases that have been deployed as this might be a required action in production when upgrades are not performing as expected (Neely and Stolt, 2013).

Furthermore, CD practices embraces an inclusion of configuration in the development pipeline (Agarwal, 2011). The management of configurations should be similar to the one of code (Feitelson, Frachtenberg, and K. L. Beck, 2013; Neely and Stolt, 2013; Humble and Farley, 2010). Utilising the same development pipeline for configuration as for code allows for operations and developers to have insight in the transformation of the configuration, as well as allowing the configuration to be verified by automated tests and deployed the same way as source code (Meyer et al., 2013).
2.3.2 Organising Continuous Deployment

CD does not only require agile processes in teams but in the entire organisation (Rodriguez et al., 2016). As such, adopting CD practices requires an increased integration of traditionally separated organisational functions such as R&D, sales, operations and QA (Rodriguez et al., 2016; Olsson, Bosch, and Alahyari, 2013).

Some researchers suggest versions of the CD practices in which the software developers employ an exploratory role, deploying experimental software and recording feedback to guide future development (Olsson, Bosch, and Alahyari, 2013). This differs from traditional development practices where predefined stakeholder requirements guide the development. There are technical requirements to enable this flexibility in the development, the system need to be highly configurable in regards to functionality and usage data or customer feedback need to be continuously monitored and analysed. (Olsson, Bosch, and Alahyari, 2013).

An increased customer involvement as a driver for innovations is one of the main characteristics of CD (Olsson, Bosch, and Alahyari, 2013). There are implementations of CD where there is a structured manner in which customers relay feedback to the developers (Rodriguez et al., 2016; Olsson, Bosch, and Alahyari, 2013). Sometimes this is done without the knowledge of the customers. Instead of relying on their direct feedback the their usage is monitored and analysed to verify the release (Feitelson, Frachtenberg, and K. L. Beck, 2013).

Releasing software changes frequently implicates that the user experience changes rapidly. To limit the confusion experienced by users, changes which are introduced in each release needs to be made transparent to users (Rodriguez et al., 2016). Deploying the software more often also reduces the time to market introduction. When a newly developed feature is delayed, the following release is days away, not weeks (Neely and Stolt, 2013). The frequent deployment has impacts on other parts of the organisation as the development flow of the product will shift. If other business sections have adapted
their processes to fit the previously cyclical development flow they need to find ways to become conform to this new pace (Neely and Stolt, 2013). This further emphasises the need for the organisation aiming for adopting CD to opt for continuous, incremental efforts across the whole organisation.

2.3.3 Between Benefits and Challenges of CD

Most previous studies conducted, point out automation in the complete SDLC as important in achieving a successful CD process (Rodriguez et al., 2016; Neely and Stolt, 2013). Furthermore, customer interaction and a surrounding administration for development are also highlighted as important aspects to facilitate CD (Rodriguez et al., 2016). As our research questions regards the encounter of CD in a regulated financial environment, we will not dwell upon how each of the steps included in the CD process can be put into practice. Instead we will here position ourselves to the literature in regards to what constitutes a CD process and to the benefits and challenges associated with adopting CD.

As CD stems from the agile methodologies (Claps, Svensson, and Aurum, 2015) we question the eligibility of CD in large and complex organisations (Dybå and Dingsøyr, 2008). We concur with literature that advocates adopting continuity in the entire organisation, not only for development teams (Fitzgerald and Stol, 2015), in order to streamline the organisation around deploying software to customers (Fitzgerald and Stol, 2015). We view this as a potential obstacle for adopting CD, when the organisations’ core business is not software development and not organised in an agile manner. Especially as adopting CD prerequisite an agile work process (Neely and Stolt, 2013; Rodriguez et al., 2016), changing an organisation from traditional waterfall methods to CD directly puts too much stress to change management and the organisation as a whole (Neely and Stolt, 2013; A. W. Brown, Ambler, and Royce, 2013).

Beck describes a set of practices that highlights XP as an agile software methodology (see section 2.1.2) (K. Beck, 1999). Small releases, testing,
refactoring, collective ownership and CI are found in XP as well as CD. Small releases and CI are tightly coupled with CD as described. Refactoring is seen as a natural way of transforming the code in XP, as it evolves the code in a natural way (K. Beck, 2004). Refactoring refers to the re-writing of old code in order keep it updated. It is viewed as a crucial part of development in CD processes (Rodriguez et al., 2016).

One practice that is present in many SDLCs, whether they are agile or not, is code reviewing (Bacchelli and C. Bird, 2013). It is the practice where a developers code is inspected for mistakes prior to integration or deployment. There is no research on the field of CD on the impact of using code reviewing in combination with a CD process. Bacchelli and Bird concludes, in their research on the topic of code reviewing in general, that although the aim is finding defects, it is not the principal outcome of code reviews. Instead the main outcome of code reviews are knowledge transfer and code improvements (Bacchelli and C. Bird, 2013). In XP, code reviewing is achieved by pair programming, the practice that all code should be written in pairs (K. Beck, 2004).

Rodriguez et al. reports on various benefits of successfully implementing CD in their systematic literature mapping. They identify the most commonly discussed benefits to include shorter time-to-market and continuous feedback (Rodriguez et al., 2016). Other benefits include increased customer satisfaction, productivity in development, release reliability, increased rate of innovation and more focused testing (Rodriguez et al., 2016). Other studies also raise these benefits as outcomes of employing a CD process (Olsson, Bosch, and Alahyari, 2013; Neely and Stolt, 2013).

Of the 50 studied publications in the literature mapping by Rodriguez et al., 80% were published in 2010 or sooner with 24 studies published sooner than 2013 (Rodriguez et al., 2016). The majority of the studies are conducted by practitioners and most studies lack empirical rigour (Rodriguez et al., 2016). This manifests our view that CD is a nascent field and it supports the correctitude of deriving criticism from the originating methodologies, namely agile methodologies. Whether or not CD is old wine in new bottles, as is a
common criticism for such new methodologies, or not, we deem irrelevant for our research questions.

Challenges in adopting CD successfully, other than aligning the organisation, has been reported (Rodriguez et al., 2016). However, Rodriguez et al. notes that more benefits than challenges has been reported. One challenge commonly discussed is the problem of aligning the organisation to continuously deploying software, as discussed above (Rodriguez et al., 2016; A. W. Brown, Ambler, and Royce, 2013; Neely and Stolt, 2013).

Another challenge is customers’ inability or unwillingness to receive software updates in an increased pace. This is due to the increased effort on the customers part to adopt to new features (Rodriguez et al., 2016). There is also an increase in the demand of QA as deployment is done more frequently. This puts increased demand on testing as a whole, as it needs to adapt to ensure the quality of every new feature developed and deployed in an increased pace (Rodriguez et al., 2016).

Although the literature brings up the importance of a loosely coupled architecture (Olsson, Alahyari, and Bosch, 2012), we do not consider this as a prerequisite for CD. It can be seen as an advantage to have such an architecture but in essence it is a question of knowing your system and how changes affect it. Loosely coupled architecture can facilitate this by not having to worry about changes affecting many parts of the system (Olsson, Alahyari, and Bosch, 2012). On the other hand, this can be covered by a rigorous test framework that ensures that changes are tested in every part of the system in the integration step (Neely and Stolt, 2013). To enable successful CD a high degree of confidence in the system is needed, confidence is gained from having a high degree of transparency in all stages of the deployment pipeline. Refined monitoring is required to achieve transparency, one needs to be able to identify negative outcomes quickly (Neely and Stolt, 2013).
Chapter 3

Empirical Setting: The Financial Sector

This chapter presents the reader with a contextual background. The characteristics of technology in the financial markets will be described followed by a description of the studied companies.
3.1 Technology in Finance

Today the financial markets are digitised, relying on integrated systems relaying transactions from financial institutions to digital markets where orders matching in price and quantities are matched and executed (Zaloom, 2006). The financial sector of today is no longer the intense trading floor characterised in popular culture, but rather a complex network of digital systems.

The different actors in this financial technology market could, as a simplification, be categorised into three distinct groups. Firstly the trading systems. These trading systems are actively participating in the markets. They are primarily utilised by financial institutions, that provide access to markets not only for their own account but also as a service to their customers. The access and trading is conducted through this trading system which can be connected to several marketplaces and facilitate many different customers and the financial institution itself. These systems varies in complexity, some send orders as specified to a specific market and some are smarter in the sense that the system calculates to which market it should send the order to and may even divide orders into smaller ones and send it simultaneously to different markets. It should be noted that this description is simplified and is subject to research in itself.

The second group are the market places. These are the stock exchanges, although its physical presence has fundamentally changed; its function has not. The markets are essentially digitised systems that receives orders sent by traders that either want to buy or sell a certain quantity of an asset at a certain price. The orders are consolidated into order books and orders that have matching buy and sell price will be executed as trades. Just as matching orders is one of the market’s core functions, communicating prices to the market participants is another. The market will continuously communicate the state of the order book to connected systems by sending information containing prices and quantities present in the market.
The third and final group is composed of the engineers of the financial systems, which both the traders and the markets rely on. This thesis focuses on this group, as they are software and hardware engineers that develop and maintain these complex systems. They are either employed by market participants, where the particular systems are used, or employed by a third-party company that delivers, and potentially operates, the system for any of the two previous groups. As an illustrative description of the role of the engineers, it could be said that if the markets would be cities in the early nineteenth century America, the engineers would be the people that build and maintain the railroad on which the traders ride to conduct trades.

Regulatory instances impose requirements on financial trading systems. These regulatory requirements aim to provide stability to the financial markets. As an example of the increased regulations of the financial markets; the European Securities and Market Authority (ESMA) who dictates technology standards in the European financial markets are due to release their policies MIFID II and MIFIR. These will give ESMA the power to intervene in any stage of a trade, including the pre-execution stage. This gives them authority to inspect product development, including the development of financial trading systems. Similar regulations are already in place in the US where FINRA oversees software development processes of financial systems.

3.2 Subject companies

Two main FinTech companies have been studied in this thesis. Both companies have been active on the FinTech markets for more than 10 years.

The primary company, hereinafter referred to as Company A, is a financial technology provider situated in Sweden. Company A delivers systems to a multitude of financial institutions. The systems are connected to several venues (stock exchanges) and propagate client orders to these markets and handle venue responses. The employees and the management of the com-
pany have all been participating in the FinTech industry for the majority of their career. *Company A* is the company with whom we have actively been participating as part-time employees during the last year.

*Company A* has recently set forth on a path to adopt CD processes in their development. Their organisation is small with less than 100 employees. The development process is characterised as ad-hoc and it is centralised around projects that typically span in length from three months to one year. The systems that they deliver are primarily operated by employees at *Company A*.

The second company, hereinafter referred to as *Company B*, is a financial institution that develops the majority of their technology in-house. *Company B*’s organisation is to be considered as large with in excess of 100 employees. Their systems are primarily used by private customers directly. The development is divided into teams that work on separate modules, integration between the teams is done sequentially approximately every two weeks. They are actively adopting agile principles in their organisations, all development teams are organised according to these principles.
Chapter 4

Method

This chapter serves the purpose of presenting the methods used to conduct this research. We present the research design followed by our data collection methods; semi-structured interviews and participant observation. In the participant observation section we handle the problem of going native. Subsequently, we discuss sampling and anonymity. Lastly, validity and reliability are discussed.
4.1 Research Design

As our research questions regard the encounter of CD and the regulated environment that the financial industry constitutes, we have valued a contextual understanding of the problem. As such, the methodology used in this study takes an interpretive approach to science by being influenced by the context in which the data has been collected (Collis and Hussey, 2013). Since we, the researchers and authors of this thesis, are part-time workers at a company that currently resides in the intersection of software development and this regulated environment, we have utilised our employment and studied this meeting with the software development company as a lens. Thus we have chosen an ethnographic methodology, immersing us in the studied company through extended participation (Flick, 2009). We are using our employment to minimise problems associated with ethnography such as building trust and becoming a member of the group (Collis and Hussey, 2013).

Ethnographic studies are traditionally associated with participant observations (Collis and Hussey, 2013; Flick, 2009) and observations play a part in our thesis as well. However, saying that we rely on only one form of data collection is incorrect as we have combined different qualitative data collection methods (Flick, 2009). Albeit our ethnographic approach we do not find our methods of data collection subordinated to a general attitude of letting the observations guide the research, as is often the case in ethnographic studies (Flick, 2009). Instead we have held a number of semi-structured interviews which constitutes the lion’s share of our research data. The interviews dictates what aspects of the meeting of CD and the regulated environment we study. The observations enables us to ask the right questions during interviews and understand and interpret the answers in a deeper, more contextual manner. Aspects that have been observed but not covered in the interviews are of course not abandoned. See section 4.2 for more discussion on the treatment of observations that are not present in the interview material.

We acknowledge that our values are influenced by the subject that we study and consider this as an asset to us in our work as it helps putting our
findings into context (Collis and Hussey, 2013). We aim to be as transparent as possible in declaring our subjectivity, allowing the reader to assess our credibility and use our work as they deem fit, as is often the case in qualitative studies (Flick, 2009).

4.2 Participant Observation

The fact that we are part-time employees of Company A, that is setting about a change to a CD process, has enabled us to immerse ourselves in the organisation while studying the meeting of CD and the regulated financial environment. This was done in order to obtain a detailed understanding of the motives and values underlying the practices observed. Other than observing individuals our work in the field included document analysis, direct participation, ethnographic interviews (note that this does not refer to the semi-structured interviews described in the previous section) (Flick, 2009; Blomkvist and Hallin, 2014). This was done at a number of different locations, primarily in the office of Company A but also at two different offices of financial institutions. At all times we have informed the observed individuals of our ongoing research to comply with ethical codes (Collis and Hussey, 2013). The observations were conducted during five months between February and May of 2016. Our observations have been compared and discussed between ourselves continuously throughout the research.

The majority of data obtained from our observations have been second degree constructs (Blomkvist and Hallin, 2014) as the main purpose of the observations has been to understand the underlying values of the topics brought forward in our semi-structured interviews. This does not mean that first degree constructs have been omitted, instead this data serve a purpose of filling eventual gaps that the interviews did not cover.

During the five months of observations, three different phases can be distinguished (Flick, 2009). First there was a period of descriptive observation where we assimilated the obstacles of CD as well as the practical implications
of such a development process. During this time our research was still diffuse as we explored the complexity of the problem (Flick, 2009). As the problem became clearer our research questions and purpose solidified. It was in this second phase of focused observations that our participation increased in the ongoing work at Company A with appropriating CD. During this time our perspectives were narrowed on specific topics.

In the transition between phase two and three we held semi-structured interviews to tap into the views of employees of Company A and this enabled us to use their perspective as guidance for what aspects are considered problematic with implementing CD. During the third phase our observational efforts focused on understanding the results from the interviews. This understanding includes what the respondents brought up during the semi-structured interviews. In analysing what was not brought up during the interviews we had a great asset in our interview with the participant from Company B, which helped us revisit our observations and view it from different angles.

4.2.1 Going Native

Our part-time employment is twofold. While it provides an asset in getting access and acceptance in the organisation it also enhances the problem of going native, a problem that is present in all studies utilising participant observation methods (Blomkvist and Hallin, 2014; Flick, 2009). Going native is especially problematic if the researcher is already familiar with the organisation under observation (Blomkvist and Hallin, 2014). In such a case it is integral to maintain a systematic critical viewpoint while gaining an internal perspective on the studied phenomenon (Flick, 2009). Awareness and reflection of going native can be sufficient in reducing the risk of going native and loosing the critical external viewpoint (Flick, 2009). Other than continuously reflecting on the issue we have taken two conscious decisions when designing our research to minimise this risk.

Firstly, we have varied the degree of participation in Company A’s activities among the authors. One has been involved in the practical work of
adapting the current software development process into one of more continual nature while the other has observed with a more non-participant nature, without being involved in any actual decision making for the firm. As a consequence one researcher could participate in discussions and drive them in order to exhaust information while the other could observe the work from a more critical viewpoint.

Secondly, we have chosen to primarily rely on the semi-structural interviews as guidance in our analysis. By doing this we aim to obtain a foundation for us to build on with our observations. Furthermore the interviews was mainly moderated by the researcher that observed of a less participant nature. This was done as an effort to encourage the respondents to speak in a more open manner, not having to worry about what CD actually is or how Company A actually plans to use it.

4.3 Semi-structured Interviews

In order to tap into the knowledge of the participants of Company A and Company B that often is a result of many years in the software development business and working in the regulated environment that the financial industry constitutes, we have used semi-structured interviews. The goal of the interviews was to obtain qualitative data on what the respondents value when developing software and their views and reasoning on how this is conducted in this regulated environment (Blomkvist and Hallin, 2014; Collis and Hussey, 2013; Flick, 2009). Building on the early phases of our observations (as described in section 4.2) we held interviews when we understood the setting but required clarification on the respondents underlying values and reasoning. Since every person express their views in different ways a semi-structured interview was considered appropriate. This approach allows time for the interviewees to reflect on certain topics of the situation and elaborate on matters that everyday work does not allow (Collis and Hussey, 2013).
Before the interviews a series of topics were identified and documented in an interview guide (see Appendix A) (Blomkvist and Hallin, 2014). Selecting subjects to interview was done in an ad-hoc manner. Initially we asked for the management’s permission to interview the employees (including management). Subsequently, we asked persons we deemed as relevant during our observations if they were available to be interviewed. Initially we held a pilot interview in order to flush out problematic topics and practice the art of asking open questions. This method of selecting and planning interviews was possible as we were participant observers in the organisation.

The interviews started with us explaining what our research is about, why we held the interviews, the importance of anonymity and asking for permission to record the audio of the interview. Thereafter the person being interviewed were to talk a bit about their background. This was done in order for the respondent to get used to being interviewed as well as opening for comparison of the current ways of working to previous work places. After that we asked open questions getting the interviewee to talk on our prepared topics. The interview guide served more as a reminder on topics to be covered than an actual guide for the conversation. We relied on probing and open questions in order to cover all topics in a natural way for the interviewee (Collis and Hussey, 2013). In the end of every interview we concluded by asking the open question if they had anything else to add (Collis and Hussey, 2013).

All interviews were transcribed word for word and collected in our empirical material. This was done in order to allow for an analysis of the interviews in a structured manner. The material was analysed to identify the topics that was discussed and the quotes that was considered of particular importance. These quotes were then collated and keywords highlighting the topics they covered were attached. During this process the source of the quotes was still maintained to ensure that we were able to utilised our knowledge of the participant to enhance the foundation of our interpretation (Collis and Hussey, 2013). As common topics crystallised they were codified and grouped together. We then grouped the codified material while discussing
the underlying meaning and the implication of the data (Collis and Hussey, 2013). As we became more familiar with the data we re-organised it further and collapsed it into more general categories. When we were satisfied with the data, in regards to its structure and comprehensiveness, we related it to the theoretical framework and tackled our research questions (Collis and Hussey, 2013).

4.4 Sampling

In total, 14 semi-structured interviews were conducted with people of various positions at Company A, including operators, developers and management. All but two of the interviewees had at least ten years experience of the Fin-Tech business. All interviews were audio recorded along with the guarantee that the interviewees would enjoy anonymity. All of these 14 interviewees have been observed during the five month period with knowledge of us conducting this research, and us participating and collaborating with them in their everyday work.

In addition to the interviews held with individuals at Company A a semi-structured interview with an executive at Company B, hereon called external interview, or external interviewee. The external interview was held on one occasion during 45 minutes on the same topics as the aforementioned interviews at Company A. This external interview was not recorded, so we sorted to taking notes during the interview. Neither Company B nor the external interviewee was subject to our observations.

The majority of interviews were conducted with employees of Company A as we strive to gain a comprehensive understanding of the views of practitioners in the financial industry. We considered that focusing on one company within the industry was preferable as it gives us the possibility to get a profound apprehension of views of practitioners in the industry while also being practically feasible to accomplish during our limited time frame. The drawback of focusing the interviews with participants from one company is the
reserved generalisability that comes as a result, to limit this we conducted an additional interview with a participant from *Company B*.

All interviews were conducted in Swedish as it is our native tongue and the native tongue of the participants. We were both present for all interviews.

### 4.5 Anonymity

This qualitative study aim to study the specific phenomena that occurs when CD meets the regulated financial environment. This has been done through observations and interviews with *Company A*. As the way in which they develop software is considered a competitive advantage as well as the nature of confidentiality that resides within the financial industry we have anonymised all participants in this study.

We have omitted all names in our empirical data in order to maintain confidentiality (Collis and Hussey, 2013). Furthermore, we have no name referencing in the report and refer to the companies where we have conducted interviews by using *Company A* and *Company B*. For additional information about the participants in the interviews see Appendix B.

### 4.6 Validity & Reliability

As our methodology of obtaining data to analyse our research questions are heavily based on participant studies as well as semi-structured interviews, where we have used our observations to further understand the interviewees, our study lacks reliability (Collis and Hussey, 2013). However as with any study under an interpretivist paradigm reliability is of little importance (Collis and Hussey, 2013).

Just as we lack reliability by conducting a qualitative study we gain validity by our increased contextual understanding (Collis and Hussey, 2013). In
observing and exhausting our interviews by using a semi-structured approach we can explore on topics discovered relevant to our purpose.
Chapter 5

Results

In this chapter the qualitative results from the semi-structured interviews and the observations are disclosed. The sections that constitute the results are presented as the main topics that were discussed and observed. The topics are organised as a summary of the takeaways from the aggregated interview material and observations; Automation, Fault Tolerance, Organising Software Development and Innovation in Regulated Environment.
5.1 Automation

Automated testing was a topic brought up by all 14 interviewees. It was not a novelty to any of the interviewees and all regarded automated testing positively although it was acknowledged that it requires additional work in designing and producing automated test frameworks and test cases. One interviewee highlighted the aspect of the feedback that testing gives you, arguing that only by testing your system can you thoroughly evaluate your system, be it functional or performance testing. Furthermore, the problem of manual testing was highlighted by some operators and developers as they explained that it is not only tiresome but also requires knowledge about the system and requires experience and skill.

One experienced developer argued that automated testing ensures a minimum level of quality, which manual testing does not. The problem of having representative test data for both manual and automated testing was another aspect brought up, thus the flexibility of the system and how the system is configured plays a part in the testing, as more manual configurations and a multipurpose system is harder to test.

Another coherent view was that efforts should be made to automate any manual testing that are a part of the development process. However, one operator pointed out that manual verification of a new feature is often quicker than writing an automated test case. Especially if there is no framework in place to verify that specific feature. The joint view was also that it is impossible to test all aspects of a system, one area known to be difficult to our subjects was the testing of Graphical User Interface (GUI) which three of the interviewees brought up. One interviewee brought up the topic of micro-services when discussing testing. He had experience in working with micro-services as well as the current system which is more monolithic. He compared advantages and disadvantages of both with the micro-services being more flexible and easier to deploy as you can deploy isolated parts of the system. Although he noted that you quickly risk losing performance in
such a system as collaboration between the different teams developing these services requires administration.

Another benefit brought forward was the safety net that testing provides for developers, ensuring that their produced code does what it is supposed to and does not break any other part of the system. This was seen as especially important due to the complexity of FinTech systems. It was expressed that this safety net can facilitate newly hired employees in getting into the routines of producing code without the anxiety of disrupting the work of others. One interviewee argued that if there are automated tests with an accepted level of coverage one does not need to focus on assuring the quality of individual changes but rather ensure that each release passes the minimum limit put up by the automated tests. One highlighted the safety net aspect of testing and related it to the joint view of all interviewees that it is impossible to test all aspects of the system; "If we aim to have perfect testing we cannot move forward, you need to view testing as a safety net facilitating rapid development. No one is perfect" (Appendix B participant 4). This was a reference to the effort that is required to ensure good test coverage in combination with the high level of complexity in financial trading systems.

5.2 Fault Tolerance

Fault tolerance here refers to the aspect of tolerating incidents in a production environment. This was a topic brought up by all interviewees as an answer to what differs development in financial industry from the development of other IT systems. All interviewees highlighted the limited tolerance for failure; one described the difference as; "If you lose frames in a video it is not as bad as losing a few orders in the closing call" (Appendix B participant 7) another one described the risk as "If you are down for half an hour and it is the half hour when a lot happens [in the market] it might turn out to be tremendously expensive" (Appendix B participant 10).
Although all interviewees stressed the importance of fault tolerance, five of the interviewees pointed out that even if the financial costs caused by production failures can be immense there are no lives at risk as might be the case in MedTech or avionics, "Failures hurt, but no one dies" (Appendix B participant 1).

One interviewee pointed out that even though the costs associated with failure can potentially be huge, it does not necessarily mean that other business are more tolerant to failure. Similar views were expressed in the interview with the external participant during discussions about the characteristics of development in the financial industry. He considered that even though the financial damage due to technical failure may be significantly greater in the financial industry it is not the regulators that drive the demand for fault tolerance but rather the competitiveness and customer satisfaction. It was argued that precisely as in other IT sectors downtime and service disruption will impact the customers and the competitiveness of the market will thus push for a high level of stability in the systems.

During the discussion about low fault tolerance of FinTech systems the participants emphasised the complexity of the systems as well as the environment as drivers behind this. On this topic one of the interviewees said "We work against a stochastic system and anything can happen out there" (Appendix B participant 7) referring to the complexity in communication with external systems in the FinTech environment.

Just as all interviewees brought up fault tolerance as a topic characterising the development, all of the interviewees answered no to the question if it is possible to develop a competitive electronic trading system that does not bear the risk of failing in production. As one operator put it; "That nothing will ever go wrong; it is impossible to guarantee that" (Appendix B participant 9). The view was that you need to do what you can to prevent failures from happening by testing and if it fails in production you fix it as soon as possible and design the system in a way to minimise the impact of failures, "The key is to avoid the extreme effects of failure, to limit your downside" (Appendix B participant 1).
On the topic of incident management multiple interviewees pushed for the use of phased deployment in order to direct a part of the users to a new release. As one interviewee put it; "Phased deployment is one way to minimise impact of production incident by regulating the flow that is subject to a new version" (Appendix B participant 4). The external participant emphasised their use of this feature to enable earlier deployment of new features. They regulate their flow in different phases starting with exposing new releases to internal users.

Three interviewees pointed out, based on their experience, that the handling of incidents can have positive effects on customer satisfaction. One experienced operator explained that if you handle incidents well you might end up better off than if you had no incident at all as the customer might be impressed with your handling of the situation.

5.3 Organising Software Development

The topic of organisation regards the aspects of processes valued by the interviewees, the type of organisations that they value and how they perceive financial institutions to organise their software development. Firstly, all the interviewees described their current organisation (Company A) as being flat with informal structures and an ad-hoc approach to developing software. They all preferred such a way of working although downsides was expressed as interrupts in the daily work and unclear responsibilities makes it difficult to know who to ask. Furthermore one downside might be lacking authority which obstructs effective decision-making as pointed out by one interviewee.

When asked to paint a general picture of how financial institutions are organised based on their experiences in working with them, it was said that the financial institutions in general had clearer processes that often had the purpose of being agile. However, it was considered that their current organ-
isation was more agile in their actual work than financial institutions. The main reason for this was simply due to the size of the financial institution as it was bigger and thus required clear processes.

The joint description of the development process at Company A was one using CI and deployment of new releases as frequent as the customer was able to handle. Frequent releases were however considered preferable with the main arguments being easier to manage the release process and to find eventual problems in the release as it is closer in mind. Regarding deployment the viewpoint of every interviewee was that it should be automated to minimise risk for manual errors and to achieve better scalability. To facilitate this three of the interviewees brought up the topic of configuration and that it should be treated in the same manner as regular code; "Configuration as code is a prerequisite if you are to deliver financial systems at scale" (Appendix B participant 11).

When asked whether it is achievable to deploy an electronic trading system continuously no interviewee believed it to be impossible even though the fault tolerance is low. On the other hand several interviewees agreed that limiting deployment to once a day, off trading hours, is sufficient and feasible; "You can automate a lot but it is not likely that deployment will ever occur during trading hours" (Appendix B participant 9). The reason for not deploying during trading hours was motivated by the argument that the marginal gain from doing so was considered too small. One of the more experienced interviewees made the comparison with making modifications to the software of an airplane, "you would much rather perform the upgrade when the airplane is still on the ground compared to when it is airborne" (Appendix B participant 1).

The external interviewee expressed a view that shortened release cycles augmented the stress on developers as they had to be production ready in a shorter period of time. During that interview it was also expressed that the release frequency was considered a soft variable dependent on the context. The external interviewee was in line with the other interviewees about the positive effect of a short release cycle. However, a position was con-
veyed by the external interviewee regarding the need of focus on stability; "It is not sufficient to focus on frequent releases, one have to focus on stability" (Appendix B participant 15). Stability was considered to require a high degree of monitoring and fault management. Furthermore, the external interviewee described Company B as agile. They had specific agile managers whose primary purpose was to enforce the agile mentality. They were organised with product owners who communicated with customers and guided development teams to develop features to customers.

Refactoring is an area where there was consensus among the interviewees, almost all expressed it as improving innovation capabilities. As one interviewee put it, "Refactoring is important so that you do not end up in a state where you are afraid of making changes. You can easily find yourself locked in old versions building on your technical debt." (Appendix B participant 13).

The observations illuminated that developers grew reluctant to undertake refactoring when the development processes required a written motivation for each change. This view was further established by being expressed by multiple interviewees.

5.4 Innovation in Regulated Environment

Innovation was considered by most of the interviewees as crucial in the technology part of the financial industry. Regulations were pointed out as being the new expected driving force of innovation, taking over from innovating for faster systems; "It is not as important with innovation anymore with regards to latency. Those who do not have low latency are not in the market any more. What will get more important is how to innovate and adapt to changing regulations" (Appendix B participant 10).
The observations revealed that financial institutions are experiencing increased demands of testing and test evidence from the regulatory instances. This was observed in documents demanding that financial institutions have a clear process for tracing changes when developing software. As the regulatory instance might request documentation of all steps, from the code being written to it being deployed in production, with sign-offs from managers along the way. It was observed that this caused financial institutions to take on a path of demanding accountability and traceability for all changes with clear business value. The increased demand for traceability and accountability was observed to impose frustration on developers and causing them to be hesitant to do refactoring.

One of the methods to impose accountability in the development was observed to be the use of code reviews, where developers validate the work done by others by reading the code and assessing its quality and looking for defects. There were interviewees that expressed concern regarding relying on code reviews to ensure software quality and to enforce responsibility according to it. "You can rarely guarantee that any code is bug free and putting your name down as responsible for someone else’s code is scary and source of a big concern." (Appendix B participant 4), was one way this view was expressed during the interviews. Other interviewees expressed similar thoughts regarding code reviews but they considered the code review to be positive while it remained on a holistic level, such as approving that a change does not have obvious errors in it.

Furthermore, this disruption of innovation, or the ability to move freely as many interviewees phrased it, is aggravated by the way financial institutions practice accountability required by regulators. The view of the majority of the interviewees was that accountability fostered individualism, made developers more territorial and thus reduce their incentives for teamwork. It was also noted during the observations that these demands, as they introduced additional steps in the development process, reduced the efficiency of the developers.
How to enable long term discrete changes with processes that promotes incremental small changes is a question that many interviewees raised. While the upsides of small incremental changes are not disputed they do not see how large refactoring and long term discrete innovation efforts fit together with a model that only allows for small incremental changes. One of the interviewees brought up the fact that these big changes take time to stabilise and that it may require increased planning and adoption to suit an iterative process. Some people even considered long term changes to go against CD practices.

The use of feature toggles, the ability to switch features on and off, to include new features in a release but hide them from customers was brought forward by one of the interviewees as well as by the external interviewee. It was considered to grant the organisation with the possibility to do big changes while deploying them iteratively. The external interviewee explained that feature toggling is in use at *Company B* and that it enables them to work with big, long term changes, while maintaining an iterative work flow.
5.5 Summary

It was considered that the demands on FinTech systems impose limitations and push for focus on stability, performance and innovation in the development. Several interviewees put forth a multitude of practices that are part of the CD process and expressed them as accepted and utilised in the development of financial systems. Such practices are the use of phased deployment, automated testing and refactoring. In table 5.1 the key findings on the topics regarding CD and the financial industry are presented.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Key Aspects</th>
</tr>
</thead>
<tbody>
<tr>
<td>Automation</td>
<td>- Auto. tests for min. quality</td>
</tr>
<tr>
<td></td>
<td>- Auto. deployment for min. errors</td>
</tr>
<tr>
<td></td>
<td>- Difficult with specialised systems</td>
</tr>
<tr>
<td></td>
<td>- Configuration as code for transparency</td>
</tr>
<tr>
<td>Fault Tolerance</td>
<td>- High cost of failure</td>
</tr>
<tr>
<td></td>
<td>- Non-lethal consequences</td>
</tr>
<tr>
<td></td>
<td>- Minimise impact of incidents</td>
</tr>
<tr>
<td></td>
<td>- Phased deployment</td>
</tr>
<tr>
<td>Organisation</td>
<td>- Financial institutions inflexible</td>
</tr>
<tr>
<td></td>
<td>- Accountability brings individualism</td>
</tr>
<tr>
<td></td>
<td>- Deploy off trading hours</td>
</tr>
<tr>
<td></td>
<td>- Refactoring to avoid being locked in old versions</td>
</tr>
<tr>
<td>Innovation</td>
<td>- Regulations drive innovation</td>
</tr>
<tr>
<td></td>
<td>- Code reviews causes concern</td>
</tr>
<tr>
<td></td>
<td>- Refactoring enables innovation</td>
</tr>
<tr>
<td></td>
<td>- Demand for stability, performance &amp; functionality</td>
</tr>
</tbody>
</table>

Table 5.1: *Key aspects of the results.*
Chapter 6

Analysis

This chapter will present the analysis of the empirical data and the theory. The purpose of this thesis is to explore the encounter of CD and the regulated environment that the financial industry constitutes. The analysis is based on qualitative empirical data gained from interviews with participants who develop technology platforms in the financial sector and experience gained during participant observations in this environment. We analyse our results by comparing them to the theoretical framework discussed in chapter 2.
6.1 Automation

The performance requirements, that was expressed as something that characterise the financial industry, could impact the hurdle to achieve automated testing and deployment. As the demand for performance dictate specialisation of systems and thus reduces the ability to standardise it was suggested that this has a negative impact on the required effort of automating. This aspect has not been noted in the theoretical framework, as this is something that may be particular to the financial industry it needs to be taken into account when implementing CD practices in the industry.

6.1.1 Testing

Automated testing is the area that was brought up most frequently by the participants, this may be related to the tremendously low fault tolerance in production that characterises the systems that our participants operate and develop. The literature brings up automation as crucial for a CD process to function and propose that one should focus on automating the time consuming tasks in the development (Rodriguez et al., 2016; Olsson, Bosch, and Alahyari, 2013). We believe that the low fault tolerance is what is driving the emphasis on automation of testing as the low fault tolerance push for good test coverage which is, as noted in our interviews, time consuming to enforce with manual testing.

There was an unanimous agreement among our interviewees that automated testing provided the developers with a safety net that prevents releases that do not fulfill minimum requirements from being deployed, this is an issue that was frequently brought up in the literature (Rodriguez et al., 2016; Neely and Stolt, 2013).

Just as automation was a topic frequently discussed in our interviews, it is frequently discussed in literature and is seen as an enabler for CD (Rodriguez et al., 2016). The most frequent aspect of automation that was brought up
in our interviews was automated testing. Automated testing in the financial industry was considered to enhance the quality of the product while also improving the flexibility of developers by increasing their understanding of the system. These benefits are noted in several of the previous research papers on CD (Neely and Stolt, 2013; Olsson, Bosch, and Alahyari, 2013). It is also well aligned with the thoughts that was expressed regarding the need for a focus on stability, both the theory and the empirical data point towards an increased level of stability with an increase in automated testing.

6.1.2 Deployment

The release frequency is an important factor in a CD model as it dictates the actual continuity of the process. Our results are consistent with the literature regarding the benefits of a more continuous approach to deployment. The gains from receiving feedback from users with minimal lag and to be able to have a short time to market was expressed in both interviews and the literature (Neely and Stolt, 2013; Olsson, Bosch, and Alahyari, 2013; Feitelson, Frachtenberg, and K. L. Beck, 2013).

The interviewees saw that some regulatory and environmental demands might impact the possibility to go below per day releases as some regulators require change logs prior to upgrade. That set aside, there was no technological barriers for employing a per day upgrade frequency. As was found during the interviews, there is a big step between deploying every day and deploying intra-day as the markets follow a daily routine hence when releasing daily, the deployment does not need to be done on a live system. This is not considered to be a barrier to the adoption of CD processes in the financial industry but it is something that one will have to take into account when the frequency is to move beyond per day deployments. As discussed in the literature, the exact frequency is not the decisive factor but depends on the environment (Neely and Stolt, 2013). With shorter release frequency the set of changes included in each release is minimised and due to this the
troubleshooting after incidents are facilitated (Neely and Stolt, 2013), this is unanimous with the empirical data.

The major problems with deploying systems intra-day was considered to be the regulatory demands of information prior to releasing new versions and the fact that systems then have to be upgraded while they are actively participating on the markets. The marginal cost of deploying new releases while the system is active was considered to be too high compared to the marginal gain of doing so. The release frequency is something that is viewed, both in theory and by our interviewees, as dependent on the context. From these views one may suggest that the financial industry have a natural limit of deployment frequency that follows the daily pattern of the markets.

Given that there are different marginal cost of deploying, depending on time of day, one may argue that deployment should not be fully automated. As one of our interviewees put it; "The operator should be able to choose when to activate a new version, as with our telephones, even if there are continuously new version delivered." (Appendix B participant 14). This is a model that is referred to as Continuous Delivery in the theory, that the software always is in a deployable state, and it might be that this is the model that is most well suited for the financial industry (Neely and Stolt, 2013).

### 6.1.3 Architecture

An aspect discussed in the literature is the need for a flexible system architecture to facilitate CD (Rodriguez et al., 2016; Olsson, Bosch, and Alahyari, 2013). The system needs to be stable in the sense that new features can be introduced with minimised impact on the entire system to reduce bug fixing (Olsson, Bosch, and Alahyari, 2013). Our results show that although systems in a financial environment recognises the need to minimise impact of bugs, it is not achieved by having a flexible system in the same sense that literature advocates. The theory brings up stability and speed as two key aspects of a system (Rodriguez et al., 2016). In our interviews we identified three key aspects of a financial system; performance, functionality and stability.
The stability here corresponds to the stability brought forward in literature. Performance and speed might seem similar at first glance but it is important to note that speed in the theory on CD refers to the speed of the processes (Rodriguez et al., 2016). As such it refers to the speed of deploying software with new functionality where performance refers to the latency of the system hence speed is more similar to our perception of functionality. We illustrate the demands of a financial system, as expressed in the interviews, in figure 6.1.

Our results show that a modular system may not provide sufficient performance to be competitive in the financial context. Although some interviewees had operated and developed a modular system at previous employments they all stressed the difference in performance of those systems and the system that is developed at Company A. The question then arises how integral a modular system is to adopting a CD process. Important features of the system such as the ability for rollback and robustness when introducing changes is not a result of having a modular system according to our observations and interviews. However, what modular systems facilitate is testing and isolation.
when introducing changes. This could be achieved with increased automated testing in the integration step of the CD process. Furthermore, we have observed that robustness is not directly coupled with modularity, it can also be achieved by failing over to back-up systems as well as having mechanisms for recovering after incidents.

6.1.4 Configuration

Automated management of configuration was considered a crucial part of a scalable CD processes by the interviewees, this is in accordance with the literature that emphasises the importance of automation and management of configuration to maintain control with frequent releases (Olsson, Bosch, and Alahyari, 2013). As is emphasised in the literature the CD processes aim for a consolidation of versions towards the utilisation of only one configuration for different types of systems. This goes against the specialisation of systems that is evident in the financial industry as a consequence of the high demands for performance.

The theory is otherwise consistent with the view on configuration that was gained during our interviewees, it notes the inclusion of configuration in the development process as important in CD (Agarwal, 2011). The gains, increased control and transparency, was noted in both interviews and observations and is uniform with the view in the literature (Olsson, Bosch, and Alahyari, 2013). The benefits of transparency might be even more important in the financial industry as we observed a demand for traceability and accountability by regulators. These qualities are enhanced with increased transparency in the development process.
6.2 Fault Tolerance

Both the interviews and the participant observation emphasises the focus on fault tolerance and performance that is present in the financial industry when developing technology platforms. These demands introduce limitations on the processes that is utilised during development which may be considered barriers to the use of CD processes.

The expressed and observed low tolerance for failure in the financial industry is a driver for automated testing. As our results show there is a consensus regarding the positive gains and the need for automated testing to ensure a minimum level of quality of the product. This testing is one of the most important steps in the CD practices as pointed out in the theory (Rodriguez et al., 2016) and the way the use of it is advocated is in line with our observations.

Actively managing flow to minimise the impact of incidents are an important part brought up in CD theory (Rodriguez et al., 2016; Feitelson, Frachtenberg, and K. L. Beck, 2013), this is something that was highlighted as crucial in the financial industry during several interviews. The low tolerance of failure seem to be the main driver of the focus on mitigation of faults and it may also be driven by the characteristics of the systems. The complexity of the systems was considered to make it unfeasible not to introduce bugs when deploying new releases hence a safety mechanisms was considered to be needed. This view is also expressed in the literature, that rollback procedures need to be in place to minimise the impact of bugs (Neely and Stolt, 2013; Feitelson, Frachtenberg, and K. L. Beck, 2013).
6.3 Innovation in Regulated Environment

Our interviews and observations indicated that a changing regulatory environment is the new main source of innovation, taking over from performance requirements. The advocated transparency and monitoring that was considered as effects of frequent releases in CD is something that is addressing the traceability requirements of the regulations already present in the financial industry. Achieving the requirement of accountability imposed on the financial systems by regulations is currently done through code reviews and gates in the SDLC of the observed financial institution.

Code reviews was a topic frequently brought up during the interviews with the notion that it can put too much stress on the developers. In essence, the view was that it is not reasonable to guarantee the correctness in any code, especially not code that is written by someone else. Furthermore, our interviews showed some loss of productivity associated with code reviewing as the workload increased. A problem that was highlighted was the difference in complexity and ability of the developers, forcing some developers to dedicate a majority of their time to reviewing others code. Many interviewees pointed out the fact that code reviews had been done in an informal manner in the past, not in the structured way that regulators requires.

The discomfort with the use of detailed code reviews to enforce accountability could be coupled with the interviewees’ view that these systems will never become perfect and neither will the changes that it consists of. The interviewees consider the complexity of the systems to be of such a degree that they cannot be guaranteed to function without imperfections hence it may never be possible to guarantee that a change is defect free during review.

In our participant observations we found regulators explicitly required code reviews, as a manual step of verification prior to deployment. Therefore, code reviewing becomes a limitation for financial institutions adopting CD. As such, financial institutions might gain by striving to increase the effectiveness of code reviews.
There is no literature investigating the impact of code reviewing on CD processes. However, research on the topic of code reviewing in general has shown that it may be utilised to improve code quality rather than find defects (Bacchelli and C. Bird, 2013). Furthermore, it can serve as a means of communicating knowledge and progress in development (Bacchelli and C. Bird, 2013). Instead of only enforcing accountability, code reviews could be extended as a tool of communication and thus facilitate the transferring of knowledge (Bacchelli and C. Bird, 2013). However, code reviewing is not a continuous practice but rather a discrete gate in the development process. Thus it would disrupt the continuity of the CD process if incorporated. In XP code reviewing is done by having all code that requires code reviewing written in pairs (K. Beck, 2004).

Another important aspect on the topic of innovation was refactoring and the notion that CD disrupts discrete long term innovation efforts. Our results show that the view on refactoring is consistent with the theoretical framework’s view on technical debt, namely that it is crucial to continuously maintain the code base so that your system does not degrade in quality over time as new changes are introduced (N. Brown et al., 2011). As deployment is done more frequent this becomes increasingly important. Furthermore, the automated testing should be maintained to guarantee the quality of the new additions of code. These views are consistent with literature on CD (Rodriguez et al., 2016; N. Brown et al., 2011).
6.4 Organising Software Development

All interviewees described Company A to utilise an agile, yet informal, approach to developing software. However, Company A is not a financial institution. When the interviewees were asked to describe development at a typical financial institution the joint view was that even though they often utilise agile processes they failed to achieve the same degree of flexibility as Company A. The reason for this was mainly due to the differences in size of the organisation in Company A and financial institutions. It was pointed out that financial institutions often employ agile processes as a way of managing the development effort in the larger organisation, while there was no need of managing the effort at Company A as it was much smaller in size. The view that financial institutions lack agility was not in line with the external interviewee of Company B who described his organisation to be agile, using many recognised agile concepts.

According to the literature, an agile organisation is necessary for implementing CD (Rodriguez et al., 2016). Research on agile methodologies has shown that it might be the case that large complex organisations have problems in adopting agile methodologies (Dybå and Dingsøyr, 2008). Furthermore, the literature on CD advocates the need to adopt continuity in the entire organisation (Rodriguez et al., 2016), which might prove a challenge for financial institutions as they often are big in size and hierarchical. The external interview contradicted this by emphasising their success in adopting agile practices in their organisation.

During our interactions with financial institutions we became aware and familiar with the concept of Chinese Wall. A concept that is present at many large financial institutions. This requires financial institutions to separate certain business functions in order to avoid problems that may arise due to conflict of interest. As large financial institutions are used to this separation we argue that there may be an opportunity to limit the need for adopting agile and continuous practices to a specific development unit within the financial institution. Thus not requiring the adoption of continu-
ity in the entire organisation as literature advocates (Rodriguez et al., 2016). Of course, the developers and operators should not be isolated but a separation of business units might facilitate introducing these agile and continuous processes.

The interviewees propose that the focus on daily tasks seize effort from developers to maintain the systems that are in operation. The interviewees were united in the view that to remain innovative it is required that the organisations adopts processes that minimises the effort needed to maintain an active system in operation. As pointed out in the interviews automation is one of the key factors in achieving this. This is in accordance to the literature regarding CD (Neely and Stolt, 2013; Rodriguez et al., 2016).
Chapter 7

Conclusion

We draw our conclusions by discussing our analysis in relation to the research questions stated in the introduction chapter. Each research question is handled separately, starting with the sub-questions and we finish off by discussing our main research question.
7.1 CD and the Financial Industry

This thesis set out to explore the encounter of CD and the financial industry. Many IT companies utilise the practices of CD to increase their rate of innovation and their service quality. The literature regarding CD do not cover the regulated financial industry, thus motivating research on the encounter of these novel processes and the financial industry.

7.2 Sub-RQ1

What barriers and facilitators to a Continuous Deployment process exists in the financial industry?

Demands put on systems in the financial industry impose limitations and push for focus on stability, performance and innovation in development. As the demand for performance dictate specialisation of systems, in the financial industry, the ability to standardise is reduced. This may lead to a need for increased effort to automate tasks of the development process in the industry. Thus it could be considered a barrier for the adoption of CD.

Even though these limitations exist there are a multitude of practices that are part of the CD process and that are utilised in the development of financial systems. Practices such as phased deployment, automated testing and refactoring are used at Company A and Company B. This can simplify the adoption a CD process.

Regulations in the financial industry demand traceability and accountability in the development. This is enforced at financial institutions by imposing accountability on developers and operators. This has the effect on the SDLC, as it was observed and brought up during interviews, that it leads to developers embracing a defensive approach in development instead of participating in offensive efforts. This focus on maintaining the day-to-day activities limits the ability to innovate and builds technical debt. The reported benefits
of CD, as put forward in the theoretical framework, is targeting these drawbacks of development by automating the tasks and by executing them more often. Thus the friction in the development process can be reduced. With a CD process the transparency in the development may be increased and thus accountability and traceability may be realised. This might facilitate an adoption of CD in the financial industry.

The regulations were expressed by our interviewees to stem from the low fault tolerance for large scale financial systems. Our interviews show that while the tolerance for failure is low there is a coherent view among developers of these systems that the low tolerance for failure should be handled with a balance of testing and minimised exposure to the impact of any production failure. This increased demand for testing and verification can be considered a barrier as it may require additional effort to be put into automation of tests as one adopts CD in the financial industry compared to other industries.

As code reviewing is a requirement put on financial institutions by regulators it needs to be adopted into the SDLC of any organisation developing in this environment. Our results showed that this is a source of concern of developers. The demand for code reviews impose a step in development that cannot fully be automated, since the reviewer is required to be another developer. This could be a barrier to CD in the financial industry as it hinders the continuity of the process.

Our interviewees express a view that there are conflicts in the adoption of agile practices in financial institutions. They argue that agile principles are valued in development but that the willingness of adopting agile principles in the entire organisation is not always a priority. As an agile organisation is an enabler for CD, this may be a hurdle for successful adoption of a continuous process in the financial environment.
7.3 Sub-RQ2

*How can the Continuous Deployment process be modified to cope with the requirements of the financial industry?*

The financial industry puts forward demands that are different from other IT sectors, such as the regulations that require traceability and accountability. Our interviews have indicated that financial institutions are adopting agile processes but are struggling to adopt organisational agility. As software development is not the core business of a financial institution and management might be reluctant to adopt a continuous approach in the entire organisation. Separation of business areas are however not a novelty in the financial context and clear separation of the development could facilitate an adoption of organisational agility in development parts of the organisation.

Furthermore, traceability and accountability is consistent with the need for transparency in organising the development and deployment of software continuously. This increases awareness in the organisation as well as enforce personal responsibility, something that is advocated in CD. The difference is the low fault tolerance. It implies that the automated testing and the design of the system should reach a stage where incidents are manageable. If that is achieved, maintaining traceability and accountability may be facilitated, as developers are jointly responsible for the code through code reviewing and development is done with awareness of impact on other areas of the system. Thus the CD process needs to cover mitigation of failure if used in the financial industry.

Deploying continuously does not necessarily imply deploying many times a day, as expressed both in the interviews and the theory. Thus achieving deployment once a day could grant the organisation the noted benefits of CD. The notion of continuous delivery, to continuously have a version ready to be deployed, to be deploy at will, may suit the financial industry as it is favourable to deploy off trading-hours.
7.4 Main RQ

*How does Continuous Deployment practices comply with the development of technology used in the financial environment?*

The CD processes emphasise reduced manual efforts and increased frequency in the deployment of software. The financial environment puts forward demands on performance, stability and functionality that somewhat limits the margin for flexibility in the development. The specialisation of financial systems that are caused by the demand for performance may increase the effort needed to automate tasks in the development process. However, as the industry require traceability of changes the process employed in the development of trading systems need to be transparent and any actions need to be evident. The CD process focus on automation to establish a foundation on which transparency is facilitated and thus increase the traceability in the development.

One could argue that the financial industry has taken its firsts steps towards CD as the automation of testing, which is considered by practitioners as the narrowest bottleneck in deployment, is underway. The demand for code review is however contradictory to the principles of CD as it introduce a step in development which cannot be fully automated, thus limiting the minimum frequency of deployment.
Chapter 8

Discussion

We elaborate on our conclusion by discussing the generalisability of our findings. Subsequently, a discussion on the credibility of our research is presented. Lastly, the discussion is concluded with a section recommending future work.
8.1 Generalizability

This study takes the viewpoint of developing systems for the financial industry and explores the encounter of CD in such a setting. Even within this industry the nature of the systems vary widely, for instance a post-trade system and an order management system differ in purpose and features. Given this our results might be specific for the development of the particular financial systems that Company A and Company B are developing. Although our interviewees’ current tasks are within one field of the financial industry the spread of their background enhances the possibility to draw wider conclusions.

As this study is specific to the context of our problem the conclusions can be generalized. In the immediate proximity to our scope, our conclusions might be extended to those other financial systems in which regulation imposes similar demands and where competition is dictated by a high rate of innovation. Be it trading systems as we have studied or post-trade systems. If our conclusions are to be generalized beyond the financial industry one has to identify areas that are subject to similar restraints, as our conclusions are tightly coupled with the regulations of our context.

8.2 Credibility

As mentioned in the delimitation section 1.5, there are many angles from which our research questions could be answered. We have chosen to take on the problem from a developers point of view, embracing the fact that developers outside the financial institutions themselves can view the regulations from distance.

Observing financial institutions from this perspective have provided us with a contextual understanding of how they function in this regulated environment and we have been able to bring this perspective to our interviews. This has resulted in an exploration of the studied problem from a develop-
ment perspective that we believe is central to the concepts of CD. A problem with this approach is however that are subject to the problem of going native.

As discussed in section 4.2.1, we have been aware of this problem and have taken cautious decisions when conducting our research to prevent this.

8.3 Future Work

Many of our participants positioned the financial industry in the intersect between less regulated industries, such as social networks, and regulated environments, such as MedTech and Avionics. This hints that our research questions may be applicable for industries that are subject to more regulations. Studies investigating the encounter of the principles of CD and these industries are advocated, as a successful adoption of them may generate increased effectiveness in the industries. Furthermore, such a study would further solidify the theory on the nascent subject CD.

To generate increased understanding of the focal point in between CD and the financial industry we propose future action research to be undertaken. These studies could evaluate the applicability of CD in the financial context by examining the adoption of CD in practice. This would grant greater knowledge about the impact that a CD process will have on the development in the financial context as well as providing further guidance on hurdles on the path towards such processes.

Lastly, empirical studies on particular topics in our conclusion such as the impact of code reviews and externally imposed accountability are suggested. A way of adapting code reviewing to the continual nature of CD might be found in the theories of pair programming in XP. We especially warrant an empirical study on the subject of implementing CD in a regulated environment where the requirement of code reviewing is met with the practice of pair programming.
Bibliography


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


Appendix A

Interview Guides
A.1 Company A Guide

Introduction

- What we do
- Our purpose
- Why we have these interviews
- Anonymity

Background

- Education
- Work before Company A
- Work with Company A

Software Development with FinTech

- Company A vs previous work places
- Preferences in ways of working
- Why working in those ways are favourable
- Differences between FinTech and other development sites - specific and hypothetical

View of development at Financial Institutions (FI)

- Development at FI - personal views
- Requirements put on FI
• Why FI do it the way they do
• Experience from FI, elaborate

Differences *Company A* vs FI

• Differences in development
• Differences in operations/deployment
• Why are there differences? Personal thoughts
• How will development/operations change long term
• What is possible to bring forth
• Personal preferences about new work ways
A.2 Company B Guide

Introduction

- What we do
- Previous research on the topic
- Our purpose
- Why we have these interviews
- Anonymity

Background

- Education
- Previous work
- Current work

Software Development with FinTech

- Practice at current work place
- Preferences in ways of working
- Why working in those ways are favourable
- Differences between FinTech and other IT industries

View of development at Financial Institution

- Regulations
- Tolerance for failure
• Deployment, continuity
• Freezing of code
• Requirements put on FI

Sections of CD

• Why continuous
• Discrete innovation
• Feedback from customers
• Manual vs automated tests
Appendix B

Participants
Participants

In total there were 14 participants interviewed at Company A. A short description of each participant is presented in table B.1. Details are omitted as anonymity is crucial. In regards to experience we classify short as 1-5 years, medium to 6-10 years and long to the excess of 10 years.

<table>
<thead>
<tr>
<th>Key</th>
<th>Company</th>
<th>Role</th>
<th>Time in FinTech</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A</td>
<td>Management</td>
<td>Long</td>
</tr>
<tr>
<td>2</td>
<td>A</td>
<td>Developer</td>
<td>Long</td>
</tr>
<tr>
<td>3</td>
<td>A</td>
<td>Developer</td>
<td>Short</td>
</tr>
<tr>
<td>4</td>
<td>A</td>
<td>Developer</td>
<td>Long</td>
</tr>
<tr>
<td>5</td>
<td>A</td>
<td>Operator</td>
<td>Long</td>
</tr>
<tr>
<td>6</td>
<td>A</td>
<td>Operator</td>
<td>Short</td>
</tr>
<tr>
<td>7</td>
<td>A</td>
<td>Management</td>
<td>Long</td>
</tr>
<tr>
<td>8</td>
<td>A</td>
<td>Operator</td>
<td>Long</td>
</tr>
<tr>
<td>9</td>
<td>A</td>
<td>Operator</td>
<td>Long</td>
</tr>
<tr>
<td>10</td>
<td>A</td>
<td>Management</td>
<td>Long</td>
</tr>
<tr>
<td>11</td>
<td>A</td>
<td>Developer</td>
<td>Long</td>
</tr>
<tr>
<td>12</td>
<td>A</td>
<td>Developer</td>
<td>Long</td>
</tr>
<tr>
<td>13</td>
<td>A</td>
<td>Developer</td>
<td>Long</td>
</tr>
<tr>
<td>14</td>
<td>A</td>
<td>Developer</td>
<td>Long</td>
</tr>
<tr>
<td>15</td>
<td>B</td>
<td>Management</td>
<td>Long</td>
</tr>
</tbody>
</table>

Table B.1: An enumeration of the participants