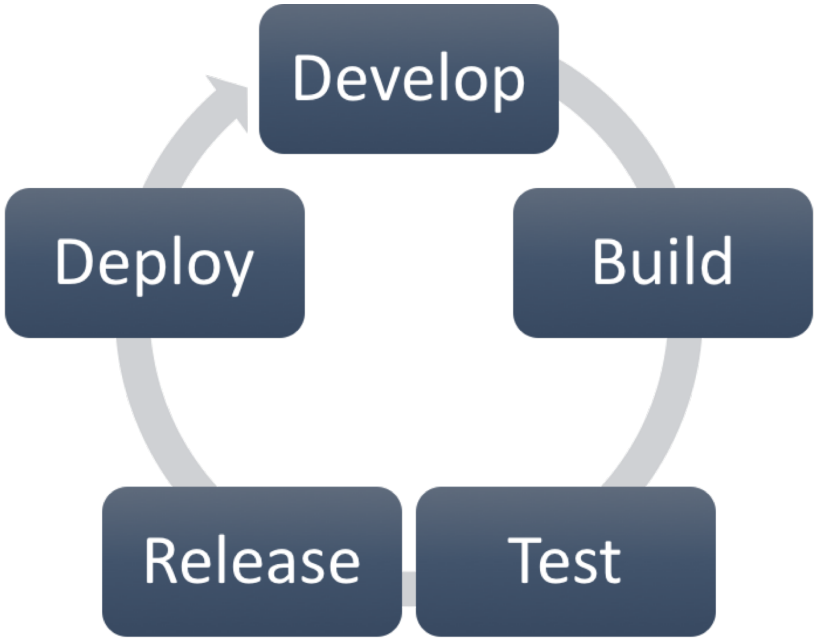

Continuous Deployment for Android Applications

Dive in or stay away

Master's thesis in the Department of Computer Science and Engineering

PÁLMI ÞÓR VALGEIRSSON

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2017

Master's thesis 2017

Continuous deployment for Android applications

Dive in or stay away

PÁLMI ÞÓR VALGEIRSSON

CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Software Engineering
Department of Computer Science and Engineering
Chalmers University of Technology | University of Gothenburg
Gothenburg, Sweden 2017

Continuous deployment for Android applications
Dive in or stay away
PÁLMI ÞÓR VALGEIRSSON

Supervisors: Grischa Liebel and Eric Knauss,
Dept. of Computer Science and Engineering, Chalmers
Industrial Advisor: Peter Kristoffersson, Football Addicts
Examiner: Robert Feldt, Dept. of Computer Science and Engineering, Chalmers

Master's Thesis 2017
Department of Computer Science and Engineering
Software Engineering
Chalmers University of Technology
Gothenburg University

Cover: A diagram showing the steps of continuous deployment

Typeset in LaTeX
Gothenburg, Sweden 2017

Continuous deployment for Android applications
Dive in or stay away
PÁLMI ÞÓR VALGEIRSSON
Department of Computer Science and Engineering
Chalmers University of Technology
Gothenburg University

Abstract

Software development companies need to react quickly to customer requirements due to fast-changing and unpredictable markets. The methodology of continuously deploying software allows companies to shorten the feedback loop and the development cycles. However, there is little empirical data on the adoption of continuous deployment in industry.

In order to fill this gap, this study investigates the challenges, risks and the information need to successfully adopt to continuous deployment for mobile applications that lack automated testing. Thus, the objective is find out if the introduction automated acceptance tests can be considered enough for continuously deploying software.

An action research was conducted in collaboration with an Android team in a small software development company located in Gothenburg. Few applications were considered but the implementation was done for a small application which is not in active development. The study involved finding a solution for continuous deployment, configuring continuous deployment and implementing acceptance tests. Moreover, processes and tools were introduced to support the adoption of continuous deployment.

The application studied was successfully deployed automatically. The developers were confident to continue deploying the application automatically with the implemented automated acceptance tests if automated tests would be added along with new features. However for a bigger, more business critical applications they felt that there was a need for including manual testing in the process.

The case company, the team, and the application are small. Further, the application under study is not in active development. Thus, the generalization of the results is limited.

The tools and methods introduced increase the productivity of the developers and show that continuous deployment can be implemented successfully by introducing automated acceptance tests in a context as the one the study was performed in.

Keywords: *Continuous deployment*, *software development processes*, *automated acceptance testing*, *mobile application testing*, *Android*, *software release*.

Acknowledgements

This study was conducted in the spring of 2017 at Chalmers University of Technology as a master thesis in the Software Engineering master's program at the Department of Computer Science and Engineering. The study was conducted in Football Addicts, a small software development company located in Gothenburg, Sweden. I was allowed to introduce and apply my academic knowledge to their teams and applications.

Everyone at Football Addicts made me feel welcome and I was involved in all day-to-day business. Whenever I needed to get help with anything there was always someone willing to help out. Peter Kristoffersson, an Android Developer at Football Addicts, played the role as an industrial advisor. He, and everyone else in the Android team, were very supportive, gave me full access to their systems and helped me out whenever I needed.

The study was supervised by Grischa Liebel and Eric Knauss at Chalmers University of Technology. They supported me throughout the whole study and gave me constructive comments which allowed me to improve the quality of the study and the report. I am thankful for all the help and effort they put in.

When I started working, after having finished my bachelor degree from Reykjavík University, my brother said that I would never go back to school to pursue my master's degree. I was certain to prove him wrong and now I have. Thanks for challenging me!

Pálmi Þór Valgeirsson, Gothenburg, June 2017

Contents

List of Figures

List of Tables

1
Introduction

Software development companies face the challenge of having complex customer requirements due to fast-changing and unpredictable markets [3]. Thus, researchers and practitioners have been paying increasing attention to understand and improve software development processes [6]. In recent years, agile practices have been gaining a lot of attraction due to different reasons, one being that they increase the ability to address the rapid changes and complex customer needs [7].

One of the methodologies that agile practices describes to address this issue is an Extreme Programming (XP) practice called Continuous Integration (CI). Continuous integration is a software development practice where multiple software engineers integrate their work frequently [8]. The integration is then verified by an automated build which runs all tests available, both unit tests and acceptance tests [8, 9]. The automated build should run every time a code change is detected and has several different tasks, such as building the project, running the tests and storing valuable output.

Even though CI is an interesting option for software development companies, some believe that the process can be extended to automatically delivering and even deploying the software. By extending CI, in a way that every change is considered proven to be releasable, is called continuous delivery [9]. In continuous delivery, the code-base should always be in a state where it is production-ready and the deployment itself should be automated such as it can be started with a click of a button [9]. The highest level of automating software release is called Continuous Deployment (CD). Continuous deployment has all the same steps as continuous delivery with the addition of having the software automatically deployed to production servers without having to manually initiate the deployment.

Continuous deployment requires disciplined work and a different way of thinking than the conventional process of developing software [3, 2]. However, companies are excited to head into the direction of continuously deploying their software as it focuses on automating processes for otherwise error-prone and time consuming tasks [9]. Additionally, CD focuses on releasing frequently, so each release becomes smaller which reduces the risk for each software release [10, 9].

There are benefits for companies to automate parts of the software development process. According to Humble, Jez and Farley [11], the process of releasing software into production is manually intensive, error-prone and risky. So the more automation there is, the less error-prone and risky it becomes. Another big advantage of

implementing CD is that it shortens the time to market [3]. However, when developing mobile applications, user experience is vital and having bugs, errors or typos in an application can have negative effects on users [12]. Many mobile applications rely on revenue from advertisements and if users are unsatisfied with the user experience it is likely that they stop using the application.

CD for mobile applications is different from CD for cloud-based applications. The main difference is that when a broken version is deployed, it is easy to fall back to a previously working version for cloud-based applications which is not possible for mobile applications [5] since it is not possible to force updates of mobile applications as the users can decide not to update [13]. Thus, CD of mobile application contains higher risk than CD of cloud-based applications and is considered more challenging [2].

Chen [9] states in his study on continuous delivery that further research about continuous delivery is required for applications which are not amenable to continuous delivery. As CD depends on automatic testing [10, 14], applications who lack automated testing can be considered not being amenable to CD. Thus, this study will aim to find out if CD can be successfully adopted for an application that has limited automated testing, i.e., where automated tests do not cover the functionality of the application completely.

CD requires having some automated testing but how much testing is required is difficult to estimate and can vary between different applications. As a part of this study a small suite of automated acceptance tests will be implemented in order to see if the results of automated acceptance tests can determine if a release is successful. Further, it will be discussed to what extend acceptance tests can be automated for Android applications and if there is always a need to perform manual testing phases in addition to the automated tests.

The case company will be introduced to new concepts, techniques and tools. First, CD will be introduced and implemented as well as automated acceptance tests. Second, tools and processes that support the adoption of CD will be introduced. Thus, the study involves introducing new concepts, tools and techniques to people as well as implement new processes.

1.1 Purpose of the study

The purpose of this study is to learn about the information need and risks when adopting to CD. It will try to address the challenges, risks and the information need to successfully adopt to CD for an Android application that has limited automated testing. Further, an investigation on if it is enough to include limited automated acceptance tests or if more thorough automated tests are required for successful adoption of CD will be performed. The company under study has few different products which run on different platforms, such as Android, iOS and Windows Mobile. Their products are for recreational usage and are therefore not safety critical.

The study will be performed in collaboration with the Android team and therefore the focus is on Android applications.

Acceptance tests are higher level than unit tests and are of more user perspective than unit tests [15]. Writing acceptance tests can allow for achieving high testing coverage with less effort that unit tests. Further, for this study the acceptance tests will focus on the most common functionality of the application. Writing unit tests so that high coverage is achieved is a time consuming task and the application being studied does not have any unit testing in place at the start of the study. Thus, the researcher decided to focus on only implementing automated acceptance tests instead of both unit tests and acceptance tests.

Studying if automatic deployment for an application in the context described, can be beneficial for many companies when deciding on how to structure their continuous integration and deployment process. Having the knowledge about the challenges and obstacles of CD, before diving into the implementation of it, can save time and resources during the deployment planning process. The study will investigate if it is achievable and what can be a good approach to do it. Further, the study will aim to describe what information is required for successfully deploying software continuously.

1.2 Research questions

The study aims to get a more thorough understanding and more comprehensive knowledge on the topic of CD. The focus of the study is to investigate if CD can work in environments where automated tests are limited and if implementing limited automated acceptance tests is sufficient, or if there is more information needed to implement it successfully. Further, the study will describe a way of implementing CD for an Android application. Thus, the research questions, this study will aim to find answers to, are the following.

RQ.1 To what extent can and should automated acceptance testing be automated to support continuous deployment?

RQ.2 How can the results of an automated acceptance test determine if a release is successful?

The main aim for **RQ.1** is to study how much testing in the form of automated acceptance testing is required for a successful adoption of CD and if having only acceptance tests and not thorough unit tests is sufficient for CD. The requirements for a successful CD will be defined and the results will reflect on if the requirements defined are acceptable as well as if they are covered.

The aim for **RQ.2** is to study the results of deploying continuously and how the results from the automated acceptance tests can reflect the results of the actual deployment.

1.3 Case company

This study is conducted at one case company, Football Addicts. Football Addicts is a Swedish company based in Gothenburg. They are the creators of Forza Football, one of the most popular football apps in the world [16]. Forza Football keeps users up-to-date on their favourite football teams and competitions, has a great coverage of video highlights along with other features which improve your experience as a football fan. The app has over 10 million installs and over 2 million monthly active users. At peak hours it handles thousands of requests per second and sends millions of push notifications. Football Addicts has in total about 35 employees, of which about 25 software engineers. Football Addicts have a mission of democratizing football by listening to the fans, making sure that the voices of the public are heard. They aim to bring the fans closer to every aspect of the game. Further, Football Addicts have worked in collaboration with few different teams such as the Swedish national team and Brøndby IF, on an mobile app where the teams can show matches and other events live, allowing the fans to interact on the broadcast by sending messages and reactions.

The company has had difficulties in their development process when distributing binaries. Their process of delivering the software, both to production and in-house, is immature and requires a lot of manual labour. The applications at hand are deployed with multiple languages and documentation is in different languages and making sure all steps are performed correctly is complex.

As part of this study, the process of deploying new software, to the test environment will be completely automated as well as the implementation of acceptance tests. If that proves to be successful the process will be extended in to deploying the software to the production environment. Today the company has a lot of manual testing done for their mobile application which will be taken into consideration when studying the complete process.

The company are the creators of few different applications. Their biggest and most famous application is Forza Football which was described previously. In addition to Forza Football, they are the creators of an application called Forza News which allows its users to get personalized news feed of football news. There is a big difference in size of the two applications and how crucial they are to the business. Forza Football has a big user base and is their main revenue stream. Thus, trying out new things for that is more difficult. However, Forza News has a smaller user base and no revenue stream and is therefore more open to experimentation. Thus, Forza News will be the main subject of the study, but other applications will be taken into consideration. Finally, the company has its own white label solution which allows football teams and associations to have their own live video streams. This study will be worked in collaboration with the Forza team and therefore the two Forza apps will be under study.

1.4 Outline

This section provides a description of the structure of this report by describing the content of each chapter briefly.

Chapter one contains an introduction to the study. It introduces the main concepts and the background for the study. Further a description of the purpose is presented and the research questions. Finally a small description about the case company is presented.

Chapter two contains information about related work. The chapter has three different sections where each section discusses related work for a certain topic related to this study. Further, previous studies and their results are discussed.

Chapter three contains a description about the methods used during the study, threats to validity and ethical considerations. The methods used consists both of the research strategy, choice of method and how the implementation was performed.

Chapter four describes the results from the study. The chapter has three sections, one section for the interviews conducted and analyzed and one section for the implementation results.

Chapter five discusses the objectives and the results of the study in relation with other related studies. Further it discusses the achievements of the study and future research about the topic.

Chapter six, which is the final chapter, presents the conclusions of the study as well as recommendations for future work on the CD.

2
Related work

Introducing related work is needed as it can give the reader a better understanding about the subject under study as well as related topics. How much literature exists on topics can vary. Thus, it is valuable to describe the basic concepts for the study by covering existing literature. The method used for the literature review which is the basis for the related work chapter is described in Section 3.3.1.

This chapter contains an introduction to previous studies, conducted by different researchers, which relate to software development processes, acceptance testing and CD. Since little academic literature exists about CD [10, 3], related topics will covered to get a good understanding of the base for CD. The first section covers previous studies about how automation can improve software development processes in general. Following that is a brief introduction about studies on acceptance testing and how that can relate to CD. Finally, studies about CD will be covered with special focus on a study performed by Rossi et al. [5] about CD of mobile applications at Facebook.

2.1 Software development processes

How software is built has been developing and studied over the past half a century and is constantly being reviewed by researchers and practitioners [17]. Software process research focuses on understanding, describing, evaluating, automating and improving the procedures, policies and techniques used when developing software [17]. Recently, research on software processes, primarily Agile development processes, has evolved significantly and is considered a multidisciplinary research domain [17].

XP is one of the most well-known agile methodologies. The academic software engineering community felt a need for changing the conventional way of developing software so that cost could be reduced, quality improved and development cycles shortened [18]. Rather than planning, analyzing, and designing for a long time early in the process, XP aims to do all these things at the same time [18]. It introduces technical and managerial practices which allow for rapid feedback, simplicity, incremental changes, embracing change and increased quality [19]. It can be argued that the popularity of XP has been a ground for developing other agile processes and methods [19].

As well as introducing different methods and techniques, XP encourages automation of the software processes [20]. Automation can improve different aspects of software

development such as configuration management, quality assurance, software build management and system operations. Fugetta and Di Nitto [17] discuss several aspects with respect to automation. First, that XP encourages writing automated tests which can be run frequently to assure for quality. Secondly, they state that tools for source code analysis exist which can provide indicators and metrics about the internal quality of the software. Finally, they discuss that collecting software dependencies, compiling, running tests, running source code analysis and finally building software can all be configured to run automatically using a build management tool. Running all these steps configured in the build management tool upon every change is what CI is.

According to Fowler and Foemmel [8] CI is a software development practice where software engineers integrate their work frequently whereas each integration is verified by an automated build to detect defects and integration issues as soon as possible. Further, Kent [20] describes CI as a set of software engineering practices that enable software engineers to speed up software delivery by decreasing integration times. A lot of studies exist on CI [10, 3] and therefore it will only be discussed briefly. Understanding CI is required to successfully implement CD as CD is an extension of CI. Thus, knowing the challenges for adopting to CI is necessary. Olsson, Alahyari & Bosch [14], mention few different hurdles when adopting to CI. One being that the company has to have knowledge on agile practices whereas another one is that the transition towards CI requires having automated tests and modularized development.

2.2 Acceptance testing

According to Haugset and Hanssen [15], test driven development and unit testing are the most known and practiced research approaches in agile development. However, they discuss that acceptance testing has emerged in agile development. Whilst unit testing is about testing small specific blocks of functionality, acceptance tests integrate at a higher level, often through user interface [15]. Thus, acceptance testing is about comparing internal requirements and end-user requirements to an implemented system [21]. Having acceptance tests can provide an important indication if software has enough quality to be released [21]. According to Hsia, Kung and Sell [21], acceptance testing should focus on the main functional and performance requirements and the interaction between the user and the system. Therefore this study will aim to define the requirements needed for an acceptable deployment, implement the acceptance tests and analyze the results. Acceptance tests do not necessarily cover all functionality of the system so bugs are likely to come forward after the system is deployed. Thus, when analyzing the results of the study different metrics will be taken into consideration when deciding about if the deployment is successful. An example metric is the number of issues identified or reported after deployment.

According to Olsson, Alahyari & Bosch [14] having automated tests is required to follow CI. However, they do not discuss about what type of tests to use and how

much testing can be considered sufficient. It is difficult to define how much testing is required and how to measure it. Stolberg [22] wrote about how he integrated his work as quality assurance (QA) manager into the adoption of CI. He defined and executed acceptance tests during sprint planning. He defined acceptance tests that he considered being "just-enough" for the introduction of CI. Doing this allowed the developers and the customers to be confident that they were testing the right functionality. Further, he automated close to 100% of the acceptance tests. He discusses that automating that amount of acceptance tests is difficult and takes a lot of time to implement. However, he also states that having automated acceptance tests dramatically decreases the technical test debt in the form of manual test cases. Stolberg [22] also emphasizes that developers should write unit tests for all new code and that the unit tests should be run with every build.

Both Olsson, Alahyari & Bosch and Stolberg [22, 14] discuss the necessity of having automated tests when adopting to CI and therefore it can be considered even more necessary for CD as having issues in the production environment is more critical than having them in the development environment. In this study an emphasis will be put on implementing CD from ground up, defining and implementing "just-enough" automated acceptance tests and see if only having limited acceptance tests can really be considered satisfiable for continuous deployment of mobile applications. Further, as a part of the study, a CI platform will be set up where tests will be run as well as binaries uploaded to a safe accessible storage. Thus, a lot of currently manual work will be automated which will save time and resources for the case company.

2.3 Continuous deployment

CD is a software engineering practice where software updates are tested, built and deployed to production environments incrementally [2]. It can be considered an extension of CI with the additional step of automatically deploying the software [3]. Thus, the main difference between CD and traditional software development is that CD has more frequent deployments [10]. For an even more detailed description of CD, Savor et al. [2] list four key elements of CD:

1. software updates are kept as small and isolated as reasonably feasible;
2. they are released for deployment immediately after development and testing completes;
3. the decision to deploy is largely left up to the developers (without the use of separate testing teams); and
4. deployment is fully automated.

When CD is introduced into an organization a significant cultural shift is required [10]. Savor et al. [2] describe three different requirements that must be met in order for CD to be successful. Firstly, there has to be full support from senior management as organizations tend to subvert to a traditional process in case of a failure. Secondly, highly cohesive, loosely coupled software increases the probability of having the updates small and isolated. Thus, each release is more flexible and less problematic. Thirdly, having the appropriate tools is important. It can increase

Tool	Description
Automated testing infrastructure	All tests need to be fully automated. Having such automation enables frequent testing with reduced overhead. Test environment should be as close to identical to the production environment as possible.
Deployment management system	A system which supports the release engineering group to manage flow of updates.
Deployment tool	A system which is able to execute all the steps necessary for automatic deployment, i.e. compilation, running tests, and the actual deployment. The tools should also allow for rollbacks to previous updates.
Monitoring infrastructure	A system which can quickly identify if there are issues occurring with the live release.

Table 2.1: Important tools for continuous deployment [2]

the productivity of developers, decrease risk which comes with manual operations and allow repeatable deployment. Finally, Savor et al. highlights few tools that are particularly important for continuous deployment. These tools are described in Table 2.1.

2.3.1 Benefits

The reason why so many organizations go through the changes needed to adapt to CD is because of all the different benefits it brings. Multiple authors discuss different kind of benefits of introducing CD into organizations [14, 3, 5, 2]. Leppänen et al. performed a qualitative study on CD where they held interviews within 15 companies. They asked about the perceived benefits of CD and came up with six categories of the benefits of CD. The six categories they came up with were *faster feedback*, *more frequent releases*, *improved customer satisfaction*, *improved quality and productivity*, *effort saving*, and *a closer relationship between development and operations*. A more detailed description of each category is shown in Table 2.2. Chen and other authors [9, 2] agree with Leppänen that CD improves software quality and developer productivity. Additionally, Chen et al. [9, 10] discuss how it can allow organizations to build the right product and improve release reliability.

The quality and the reliability of mobile applications can have a lot of influences on their usage. Further, having released a bad quality or unreliable application can be unacceptable as updates cannot be pushed to the users' devices [13]. So according to the benefits described by Chen et al. [9, 10], i.e. having more reliable, more frequent releases can allow organizations to increase the quality of each release and which can lead to higher user satisfaction.

2.3.2 Challenges

Introducing new processes and changing the way of working within an organization can have its challenges and obstacles [3, 9, 10]. As Chen et al. [3, 9] discovered, in-

Category	Description
Faster feedback	The immediate feedback through CI minimizes waiting time for tests results e.g which reinforces developers' sense of accomplishment and increases their motivation. Also being able to receive more frequent feedback from customers and users allowed more freedom for trying out new features.
More frequent releases	Having more frequent releases leads to less waste because features are deployed as soon as they are implemented, i.e. do not wait in the development pipeline to be released. It is also valuable to customers as the results are visible to them sooner.
Improved customer satisfaction	Developers can respond faster to customer feedback in terms of new features and maintenance releases.
Improved quality and productivity	Code quality improved because CD emphasizes build and test automation together with reduced scope for each release. Also, CD requires automated tests which in turn is likely to improve quality.
Effort saving	Having more streamlined release process which includes automation reduces manual labor.
Closer relationship between development and operations	CD can prevent development silos as tend to happen when development occurs over long periods without any connection to the production environment. So communication between operations and development has a tighter schedule than before because releases are more frequent.

Table 2.2: Benefits of continuous deployment [3]

troducing CD involves both technical and process related challenges. They identified different challenges, such as that there was a resistance to change for some of their subjects. Another challenge identified by Leppänen et al. is that developers must be confident in their work and trust it. As the code must be close to defect-free, a lot of the companies thought that their software was too vulnerable, e.g. because it did not include enough automated testing [3]. Those companies ran through series of manual test before each deployment to ensure that all the requirements were in place. If that will be the case for this study, the developers will be asked if having an additional phase, where manual testing is performed, would increase their confidence in CD.

As for this study, automated testing will not be extensive, so results will mainly be derived from qualitative studies. The confidence level of developers and testers is a big factor about the outcome, i.e., if only having limited automated tests is sufficient for CD. Those results can support or argue against Leppänen et al. [3] theories about developer confidence level being one of the big challenges for CD. Further, Leppänen et al. [3] emphasize that before starting CD, the organization must look into the challenges identified and think about how to overcome these challenges.

Finally, Chen [9] states that there is a need for further research on CD when dealing with applications that aren't amenable to CD. Applications can be considered not amenable for different reasons, one being the lack of automated tests. The case company's product lacks automated tests which makes it not amenable to CD.

Therefore, this study addresses the need stated by Chen.

2.3.3 Mobile applications

According to Rossi et al., cloud-based software environments allow for many small incremental deployments since they are usually not inconvenient for the end user. Moreover they discuss that cloud-based environment supports features which reduce risk of potentially deploying erroneous software and a simple way of "rolling back", i.e. undoing changes, by redeploying previous version. However, the environment for mobile applications is completely different. Thus, applying CD to mobile applications has additional challenges in addition to the ones already described in Section 2.3.2 [5].

Rossi et al. [5] performed an extensive analysis on the mobile application deployment process at Facebook. The study was performed over a seven years period and during that time Facebook made significant progress in increasing the frequency of its mobile application deployments. The key challenge that Rossi et al. [5] identified is that it is not possible to deploy mobile applications in the same manner as for cloud-based services, due to the following reasons:

1. The frequency of software updates may be limited since software updates on mobile platforms are not entirely transparent.
2. Software cannot be deployed module by module, the complete software binary is deployed at each time; which increases risk.
3. Risk mitigation actions are limited. E.g. hot-fixes and roll-backs are largely unacceptable.
4. The end user can choose when to upgrade the mobile software. So, different versions of the software can be running at the same time and all need to function correctly.
5. Many hardware variants and multiple OS variants need to be supported at the same time. This increases the risk of each deployment significantly because the size of the Cartesian product: *app_version* $\times$ *(OS_type & OS_version)* $\times$ *hardware_platform*

Rossi et al. [5] also studied thoroughly the testing process at Facebook as that is a crucial part having automated deployment process. The testing process proved to be comprehensive with tests on different abstraction levels. The types of testing they perform are *unit tests*, *static analysis tests*, *integration tests*, *screen layout tests*, *performance tests*, *build tests* as well as manual tests. The purpose of each one of those types are described in detail in Appendix B.

CD can be a completely automated process but some companies feel the need to have some manual testing as well [3, 5]. The CD process at Facebook for mobile applications is an example of that. A team of about 100 is used to manually test the mobile applications. The team does various smoke tests and edge-case tests to ensure the apps behave as expected, primarily for new features for which automated tests have not been implemented for [5]. Finally they are responsible for UI testing where they are required to focus on language translations and the quality of user

experience.

Testing is important for mobile applications for a few reasons. One of the reasons is that the options available for taking actions when critical issues arise post-deployment are limited [5]. To mitigate the potential risks of deploying critical issues Rossi et al. describe Facebook's testing 5 testing principles.

The first principle is coverage. The coverage principle states that the tests are comprehensive. Second principle they mention is that tests are responsive, thus allowing regression to be caught early which makes it easier to address. Third is quality, i.e. the tests should identify issues with precision. The fourth principle is that tests are automated as much as possible which allows them to be run repeatable, on regular basis. The final principle is prioritization. Because testing requires a lot of computing resources and therefore the focus is put on running only tests which the committed code affected for each commit and only run the full test-suite for the high priority branches.

As for results, Rossi et al. list seven key findings for their extensive study. These findings are listed as following:

1. Engineer's productivity, measured in lines of code (LoC) or number of commits, remains constant even as the number of engineers working on the application grows by a factor of 15.
2. The CD processes do not negatively affect productivity even though the number of engineers scales significantly.
3. The number of deployments has little effect on the number of critical issues that arise from deployments.
4. The length of the deployment cycle does not significantly affect the number of launch-blockers.
5. Shortened release cycle does not appear to affect the number of crashes per deployment.
6. Changes committed on the day of a deadline are of lower quality.
7. The software quality decreases with the increase of developers involved in adding changes to the code base.

Their findings are especially interesting for this study as they focus on mobile applications. As for the case company for this study, it will be especially interesting to look into certain items of the list. Firstly, items 3 and 4 are interesting as they can both be strong arguments on how CD can be implemented successfully for mobile applications. Secondly, items 6 and 7 give examples of good practices to follow when doing CD.

The study Rossi et al. performed was an extensive study which prolonged over a long period. It was performed in a company which has a lot of resources and mature, extensive release and testing processes. Even though the scope for that study was a lot bigger and in a bigger organization, it's findings and processes are valuable to this study as it is, to the researcher knowledge, the only systematic study on CD for mobile applications.

3

Research Methodology

This chapter will present the research strategy, the choice of method and the methods that were used in order to fulfill the purpose of this study. The first section explains the general research strategy which is followed by the choice of method. Further, the applied methods during implementation, data collection and data analysis will be thoroughly described. Finally, sections on threats to validity and ethical considerations that were addressed during the study are presented in order to increase the trustworthiness of the results.

3.1 Research strategy

The research paradigm is characterized in a way that the researchers measure the effect of one variable on another [23]. Thus, it must be performed in an closed, none changing environment. Runeson and Höst [24] mention that the research paradigm is not a good match when investigating human interaction with technology. Since CD requires a mind-set shift and an organizational change such that all stakeholders, both software engineers and management, need to know the status of the project, the research paradigm is not suitable for the investigated topic [9]. The aim of this study is to set up CD at a company and analyze the results using different techniques. Doing a study in this way, i.e. introduce tools and processes and reflect on the outcome is one way of performing action research [25]. Since the study involves people and the introduction of new tools and methods, it will be performed as action research.

3.2 Choice of method

Dick [25] describes action research as a natural way of acting and researching at the same time which is an open and general way of describing action research. Reason and Bradbury [26] say that there is no simple way of describing what action research is but they define it as a:

> ...participatory, democratic process concerned with developing practical knowing in the pursuit of worthwhile human purposes, grounded in a participatory worldview which we believe is emerging at this historical moment. It seeks to bring together action and reflection, theory and practice, in participation with others, in the pursuit of practical solutions

Action research	**Design science**
Emphasizes on the utility aspect of the future system from people's point of view.	Products are assessed against criteria of value or utility.
Creates knowledge for guidance in modification.	Produces design knowledge such as models and methods.
Involves both taking an action and reflecting on the action.	Building something and reflecting on it are two main activities of design science.
Carried out with the researcher and an external system or individuals.	Initiated by the researcher(s) interests. Focus on solving a local problem in collaboration with the external team.
Modifies an already existing system or develops something new.	Aims to solve construction problems and to improve. Does it by producing new innovative things or improve existing.
Generates knowledge that has been tested and modified in order to improve.	Generates knowledge which is then used through the building action.
The researcher acts on the problem	

Table 3.1: Similarities of the fundamental characteristics of action research and design science [4]

> to issues of pressing concern to people, and more generally the flourishing of individual persons and their communities.

In other words, action research is a research methodology which is used when a study requires to do a certain act, such as implementation of some technology, and then observe the results. In a way, action research is related to case study with the exception that case study is strictly an observational process whereas action research includes some kind of an act [24, 25, 27]. Researchers that do action research aim to solve real world problems while simultaneously studying the experience of how to solve them [27]. Thus, action researchers aim to intervene in situations with a purpose of improving the situation [27].

Design science is another research method which is common in the field of software engineering and is similar to action research [4]. Järvinen [4] lists seven different fundamental characteristics of action research and design science where he discusses its similarities. These similarities are described in Table 3.1.

Having listed the similarities between design science and action research, what is it that makes action research a better fit for this study? First, the study is about introducing CD to a team that does not have any automation when it comes to their release process. Second, it is about en-lighting the software engineers of the team about different tools and techniques that are valuable when adopting to and implementing CD. Finally, it is about introducing and implementing a new process for the team and see the influences it has and then iterate to improve it even further if the first iteration proves to be valuable. As described earlier, action research is about doing a certain act, which in this case is to implement CD process, and then evaluate the influences it has [4]. However, design science is about building something and evaluating it afterwards. This study is about implementing CD, a widely known development process, in a specific environment. In addition, it will

introduce acceptance testing and reflect on how much testing can be considered enough to allow for successful CD. If more than one iteration is required, in order to come to a conclusion, another iteration will be performed. Thus, following the action research methodology is a suitable way of performing a study like this one.

Action research has been criticized for having less value, being biased and impossible to generalize since they are different from controlled studies [24]. Controlled studies measure the effects of one variable on another but action research aims to gain deeper knowledge about the topic under study [24]. Thus, it is necessary to make sure that data collection and analysis is performed in a disciplined way [24]. It can be difficult to extend the results of a study that is performed using action research out of the local context it is performed in [28]. Thus, using tools and techniques that can be applied in different contexts is required to allow for a more general conclusion on the matter.

The fact that only one iteration will be performed, it can be argued that it is not action research. According to [29] Avison et al., action research is a process where researchers want to try out theory with practitioners in real projects, collect feedback and modify the theory according to the feedback and then try it again. Further, they describe that each iteration of the process adds to the theory. In this study the researcher did work with practitioners on a real project, collected feedback but did not modify the theory. However, according to Dick [25] action research is a way of acting and researching at the same time by working with practitioners on a real project and then reflect on it. Hence, according to Dick, it does not require multiple iterations to be considered action research.

3.3 Applied methods

This section describes the methods applied while conducting the study. Multiple methods were performed for different parts of the study. This section contains subsections for literature review, implementation, data collection and data analysis. Each method is described, followed by a description on how it will be performed. The methods used are different from each other, especially since this study is an action research. The methods used for implementation are more hands on approach whereas the ones used for collecting data include methods commonly used in academic studies.

3.3.1 Literature review

As part of the preparation of the study a literature review was conducted. Studying existing literature allows the researcher to gain a thorough and deeper understanding of the topic under study. Bryman & Bell [30] state that by reviewing existing literature allows the researcher to identify the concepts and theories that are already known in the area of interest. During the literature review it was discovered that CD is a topic that has not been studied a lot [3, 10]. Thus, the literature review was performed in a structural way where literature on different related topics were

grouped. The topic groups that the literature review focused on were CD, CI, agile processes, automation and acceptance testing. Performing the review in groups like this allowed the researcher to understand CD and its dependencies as well as getting thorough understanding of other topics related to the study.

The literature review for this study was performed in an unsystematic way, by looking up literature in electronic databases [30]. Some of the referenced literature was retrieved from the Chalmers Library but most of it was found on different electronic libraries using the search engine Google Scholar. Different keywords were used when searching for literature, keywords such as: *Continuous deployment, continuous integration, agile software processes, software automation, acceptance testing, etc.*

3.3.2 Implementation

This section describes the methods used and the tools introduced during the CD implementation process. The implementation will only be done for Forza News and not Forza Football due to the fact that it is more open to experimentation as described in Section 1.3. After having implemented CD for Forza News the company will be provided with recommendations on how to proceed with CI or CD for Forza Football.

3.3.2.1 Branching model

Having CD in place requires software engineers to work with high discipline [10] and with focus on writing high quality code. Committing low quality code, which will then be automatically deployed, can lead to bad user experience. Further, having a model allows the team members to develop a shared understanding of the branching and release processes [1]. Thus, a new branching model will be introduced to improve the structure of the code repository as well as creating a shared understanding between team members. The branching model that will be introduced is called GitFlow. GitFlow provides a couple of concepts that make it simpler to work on new features, pushing new releases and working on fixing bugs.

GitFlow is a model where each repository has different branches all with their special purpose; some that always exist and others that exist for a period of time [1]. There are two main branches, called master and develop. The master branch should always only have released code whereas the development branch contains code ready to be released. Other branches exist for work-in-progress, like when a new feature is introduced or a hot-fix is required. A graphical explanation of how the model works can be seen in 3.1.

A git extension tool exists which will be used when introducing the branching model. It is an extension on top of the default git command line tool to provide high-level repository operations GitFlow [31]. Having a tool like this extracts away the multiple commands needed to implement GitFlow and allows the researcher to emphasize how the model works instead of having to describe each command. The tool is well documented and outputs a really good explanation of supported available commands

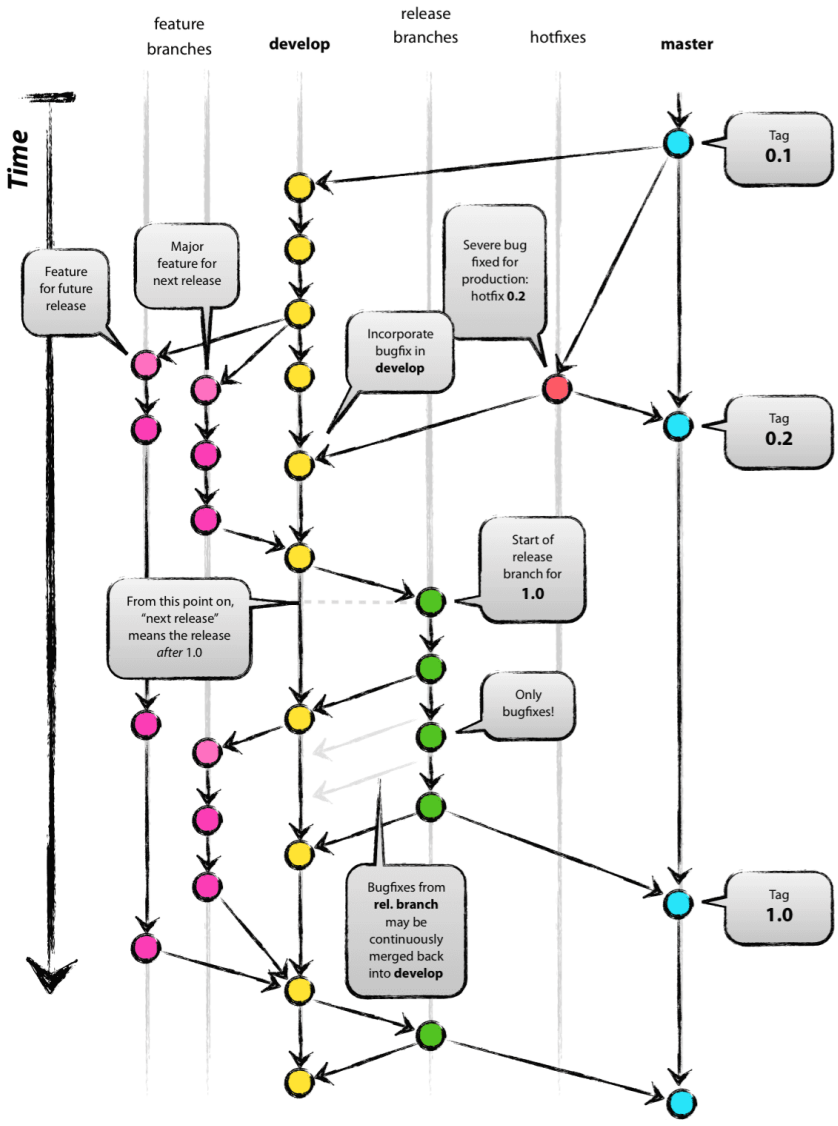

Figure 3.1: GitFlow branching model [1]. Swim-lanes for each type of branches are shown and how they are merged.

after each run command which is very helpful.

By following GitFlow the team has a structure for their code repository which gives them a clear overview of which features are ready for release and which features are under development. Further, it allows them to easily respond to issues that surface in production by creating hot-fix branches. Without a clear working process for branching, unfinished features could easily be accidentally deployed which can have bad consequences. Thus, introducing and integrating a branching model encourages developers to work in a more disciplined way which is essential for CD to be successful.

3.3.2.2 Acceptance tests

The aim for the study is to investigate if only having limited automated acceptance tests can be considered enough for automatically deploying mobile applications. Acceptance tests are used to verify if the end-user requirements are met in the released software [21]. Thus, implementing acceptance tests can give an indication if the software supports the functionality that it is supposed to support. Further, it does not mean that the software is completely tested or without issues, but should give a certain confidence that the functionality required for acceptance is working as expected [21].

The first task of this study is to define the requirements for acceptance. The QA manager at the company will be responsible for defining the initial criteria by coming up with test cases. The researcher along with the software engineers will then implement these test cases using an automated test framework. The results of the first iteration will then be a deciding factor if the already implemented acceptance tests can be considered enough for CD, if more tests need to be implemented or if the team members are not confident at all about that having only automated acceptance tests can be enough for CD.

3.3.2.3 Continuous deployment solution

As CI is already a widely known and used concept in software engineering, different tools and solutions exist. Early on during the study, an analysis will be performed where few solutions will be investigated. The solutions can be different in many ways and finding a good fit for the project, the team and the company is necessary. If the software engineers that will be using the solution do not like working with it, it is more likely that if some issues surface, they are not willing to spend time fixing it.

Four different CI solutions will be investigated. Two of which are self-hosted and the other two are cloud-based. The main difference between a self-hosted solution and a cloud-based one is that the self-hosted one usually requires more configuration and maintenance. Thus, hosting one is more time-consuming but has advantages such as being more flexible and allowing custom setup aligned to the needs of the organization. Having a cloud-based solution does not provide the same flexibility but requires less setup and maintenance. The four solutions investigated are called *Jenkins*, *TeamCity*, *Travis CI* and *CircleCI*. A description of each one of them and their attributes is listed in Table C.1 in Appendix C. However, which solution is to be selected depends on the results of the analysis the researcher made about how the solutions fit the project as well as the developers opinion from the initial interview described in 3.3.3.1.1. The researcher will recommend a certain solution depending on those results and then a selection will be made in collaboration with the team.

3.3.2.4 Development process

The software development process for the team studied is not well defined. They have a planning session once a week where they define the tasks to work on during

the coming week. Further, they communicate within the team about current status of tasks, if there are blockers and when to release. They are capable of doing this since the team is small. Most often feature branches are created for new features and then code reviewed before it is merged to the master branch. It happens regularly that untested code gets merged to the master branch which increases the risk of having completely untested code deployed to production.

In the beginning of the study the application had more or less no automated tests which means all functionality needs to be manually tested before each release. When testing is required, a binary is built locally on the developer's computer, and then sent to the QA manager with a small description on what has been implemented. If an issue is detected, it is reported to the developer who works on it and then builds a new binary and sends it back to the QA manager. This is an iterative process which continues until the QA manager approves. Thus, the overhead of binary distribution for testing is very time consuming and distracting for developers.

In order to be able to adapt to CD, a new work-flow is going to be introduced. The first step is to introduce GitFlow as described in Section 3.3.2.1. By introducing GitFlow, developers get a better overview of the work in progress and increases their discipline as they are required to follow a certain process of working with their code base. Further, a continuous integration and deployment solution will be set up where different tasks will be automated.

The CI server will take care of multiple tasks. Firstly, on each commit to the development branch, it will run the tests, build a binary and if successful, deploy it to a cloud based storage which can be accessed by all employees of the organization. Further, it will store reports about the testing results and take screen-shots for all languages configured for the application at hand. Having screen-shots allows them to quickly see if there are any obvious issues with different languages. A common issue is that in a certain language the text overflows differently because of different character count. Each binary is stored with description about which branch it was built from and when, making it is easy to find out what is included in each build. A high level model of how the development process that will be introduced can be seen in Figure 3.2.

In addition to running the build on the CI server per commit, a daily task will be set up that runs on an office server. This task involves running Spoon, which is a tool that aims to simplify running Android tests on multiple devices and to report results in a meaningful way [32]. The task will use Spoon to run the acceptance tests on all the devices the organization has in their so called *device table* which is where their devices are kept. An image of the *device table* can be seen in Appendix C. Further, in Appendix C an example Spoon report is shown in Figures C.1 and C.2.

CI and CD requires different tasks to be automated [10], scripts to support automation were implemented. Luckily, the default build tool for Android, Gradle, supports different tasks for building and running automated tests so there was no

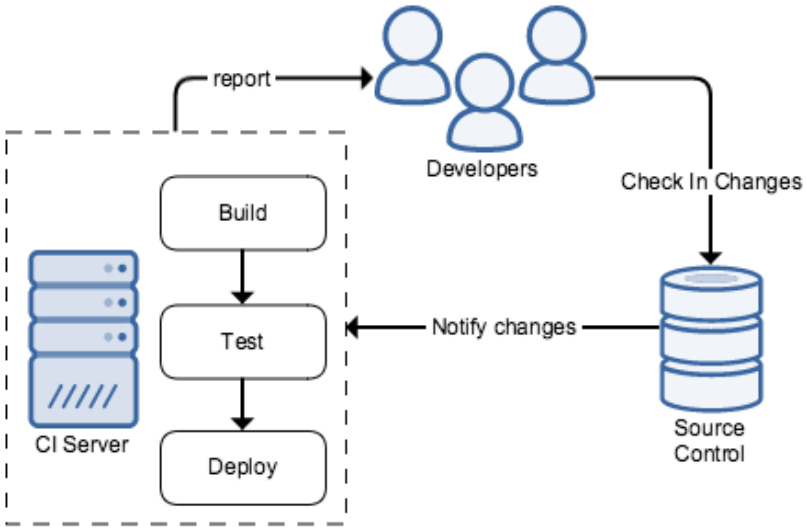

Figure 3.2: A graphical explanation of how CI server is integrated into the new development process.

need to implement scripts for that purpose. Further, a plug-in for Gradle which supports deploying binaries to the production environment was discovered and integrated. However, scripts for deploying binaries to a cloud based file storage were implemented as well as a script that runs Spoon periodically was created and installed on an office server. Finally, since Android binaries need to be encrypted, a script which fetches the keys, required for encryption, was created.

3.3.2.5 Release process

The release process the team follows today is quite unstructured and completely manual. The release is performed by the responsible developer locally on their computer. Doing the process locally increases the likelihood of having invalid configurations or old builds interfering with the new release. Further, they have a release checklist which they follow. The checklist consists of items such as *run tests*, *check for dependency updates*, *bump version codes* etc. Following a list like this manually is risky as there are many steps in the process and with each step it is more likely that something goes wrong. A graphical explanation of the current release process can be seen in Appendix C, Figure C.4

In order to support CD, the release process will need to oversee a change. To begin with, a certain defined process needs to be in place to make sure each release goes according to plan. Savor et al. [2] describe two key principles to follow in CD release processes. First, each release should be relatively small and independently deployable. The size of the update will depend on the nature of the features included in the release as well as the organization. Second, the software developers are responsible for the release. Thus, developers must be involved in the full deployment cycle, i.e. responsible for configuration, testing and staging, and respond to issues when they occur. Having such a big responsibility provides fast turn-around time in

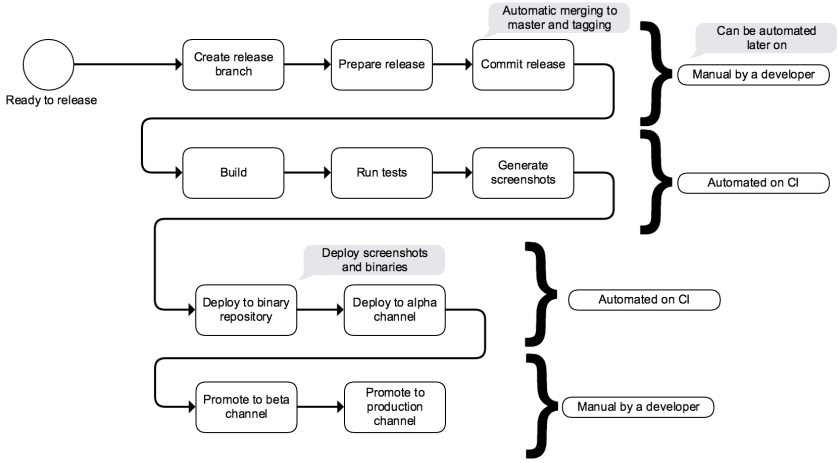

Figure 3.3: A graphical explanation of the new release process.

the event of failures. The feedback loop is shortened which is an important aspect of the process. It allows developers to quickly act on issues as the implementation is likely to be fresh in their minds [2]. With these principles in mind, a new release process was designed in order to support CD.

GitFlow involves a certain process of releasing software, which is the initial stage of the new release process. When all features and hot-fixes have been merged to the development branch a new release branch is created. The release is prepared on the release branch, i.e. version numbers are updated and production configurations are prepared. At this stage, no new functionality should be introduced. Thus, if the team feels that some manual testing is required or if some translations need to be reviewed before the release it is done at this point in the process. If no major issues are discovered, the release branch is finished. Finishing a release branch in GitFlow involves automatic merging to the master and development branches as well as automatic tagging. Further, committing the release branch triggers the CI server to perform an automatic deployment.

When the CI server detects a new commit to the master branch its configurations tell it to perform a release. A release task on the CI server is close to being the same as for each commit to the development branch, with the addition of deploying the software to the production channels. First it builds the binary, runs the tests and generates testing reports and screen-shots. Note that if any of these tasks fail, the build will not continue. Secondly, the binary is deployed to the cloud based file storage along with all other generated data such as reports and screen-shots. Finally, the build performs deployment to the the alpha channel in the production environment. Deploying a binary to the alpha channel allows all users that have been invited to it to get the newest update. At this stage, a limited group of users can update the application and get the newest, automatically deployed, updates. Depending on crash reports, user feedback and overall confidence of the developers the version is then promoted to the beta channel and finally the main production channel.

3.3.3 Data collection

This study involves data collection using different techniques. This section describes these techniques, how they are performed, to what extent and why they are suitable for this study.

3.3.3.1 Interviews

When conducting qualitative studies, such as this one, collecting data through interviews can be very valuable [33]. When interviews are used for data collection the researcher asks a series of questions to subjects about topics related to the study [24]. Different structures of interviews exist where the structure defines how the interviews are conducted. Runeson and Höst [24] describe three different interview structures, *unstructured*, *semi-structured* and *fully-structured*.

In this study, two types of interview techniques will be used, both *unstructured* and *fully-structured*. Runesön and Höst [24] describe these structures in this way. In an *unstructured* interview the questions are formulated in a way they relate to the interests and general concerns of the interviewer. They are in a way a discussion between the interviewee and the interviewer on topics that interest the interviewer. On the other hand, in a *fully-structured* interview the researcher has planned all the questions as well as the order of them. Further, a *fully-structured* interview can have scales so that the interviewee could both have to answer on a scale as well as with their own words. Thus, a *fully-structured* interview is similar to a questionnaire-based survey. All interview questionnaires for this study can be found in Appendix D.

3.3.3.1.1 Introduction interview

Early on in this study an *unstructured* interview will be conducted in order to gain more understanding of the team's knowledge of the topic being studied. The interview includes questions about what they think about CI and CD, how they would implement it in their current environment and their opinion about it. Further, there will be questions about if they believe that CD will be suitable to their projects and what they think of the introduction of acceptance testing. The interview questionnaire for these interviews is presented in Section D.1 in Appendix D.

3.3.3.1.2 Implementation results

A *fully-structured* interview will be conducted with all team members at the end of the implementation. The interview will consist of questions related to the current status of the CD adoption process. Participants will answer questions on a scale with an additional comment if wanted for each question. An example template for a question of this sort is shown in Figure 3.4. The questions will be about how they think the status is currently. If they believe that the current project status allows for starting automated deployment to production, if they believe that more testing is required, or if they have any other concerns. The reason for having this

Figure 3.4: Example template for a structured interview question in the implementation result interview.

interview structured is so that the participants can only give a concrete answer of their opinion. Along with other metrics, the results from these interviews will be one of the main deciding factors about if CD can be implemented in the study's context. The interview questionnaire for these interviews is presented in Section D.2 in Appendix D.

3.3.3.2 Acceptance tests

Since the project at hand is lacking automated tests, a suite of automated acceptance tests will be implemented as it is required for a successful implementation of CI and thus CD [14]. A description of how they will be implemented can be found in the implementation chapter, section 3.3.2.2. Pölzleitner [34] discusses multiple advantages of having automated tests. Advantages such as generating code coverage reports, getting benchmarks and performance reports. The number of test cases and functional coverage will be one of the factors that affects the conclusions of this study. Both by analyzing coverage as well as interview answers will be used when defining how much automated acceptance testing can be considered enough. Thus, it provides valuable information for concluding **RQ.1** and **RQ.2**.

3.3.3.3 Issue reporting

As mobile applications run on the users' devices, issue reporting can be a valuable source of data. Since there is no ethical way of accessing data on the users mobile devices without their consent, the application has integrated an issue reporting tool. The tool that is in use at the case company is called Crashlytics. Crashlytics is part of Fabric which is a platform that helps mobile teams to build better applications,

understand their users and grow [35]. Fabric can be easily integrated into different mobile platforms such as, Android, iOS and unity. Crashlytics is a powerful, yet lightweight crash reporting solution [35]. It allows software engineers to spend less time finding issues and more time fixing them. It supports different types of reports which provide deep and actionable insights on crashes and configurable logging.

This study will use the issue reporting tool at hand to compare the number of issues reported and their severity between previous versions and the ones that will be automatically deployed. If the number of critical issues, e.g. issues that crash the application increases dramatically it will not be possible to conclude that the release was successful. However, it will be a positive factor for a successful deployment if the number of issues stay the same or even reduce. Thus, having a issue reporting tool is helpful because otherwise it would be difficult to get information if the users are experiencing crashes as it is not running in an environment that is easily accessible.

3.3.4 Data analysis

This section describes how data analysis will be performed to explain the results from the data collection. It contains different subsections for different data collection methods where each section describes how data analyses was performed for the method at hand.

3.3.4.1 Interviews

Two methods will be used to analyze the data collected from interviews. Two different types of interviews will be conducted, one *unstructured* introductory interview and then a *fully-structured* one after the implementation. All interviews will be transcribed in order to be able to analyze the information they provide.

The *unstructured* interview will be performed early on during the study to introduce myself and what my aim at the case company is as well as get to know the team. The purpose of that interview is to gain more knowledge about how the team is currently working and what knowledge and opinion they have about CI and CD. After having transcribed the interview answers, the most relevant parts of it will be summarized in an abstract way in order to set the ground for following work.

The *structured* interview will be conducted after the implementation phase. The purpose of it is to find out how confident the team is about doing CD. The interview is structured where the team members can select options such as *strongly agree* and *strongly disagree* which generates quantitative data. However, since the sample is small, with only three participants, the data analysis will not be done in a quantitative way. The results will be analyzed and then summarized. Thus, the data analyses will be done in a qualitative way even though the data can be considered quantitative. Having the data quantifiable allows the researcher to get a clearer view on the participants answers and the threat of misinterpreting answers is kept minimal.

3.3.4.2 Acceptance tests

Automated testing is one of the key requirements for having CD [14]. Before the study, the application being studied had no automated tests. During the implementation phase some automated acceptance tests will be implemented. The number of these tests and how much of functional coverage they have will be analyzed in order to define how much automated testing can be considered enough. The results from those analyses will be introduced to the interviewees before the *structured* interviews that are held in the end of the implementation phase. These tests can be one of the factors that will increase the teams level of confidence in CD. Defining functional coverage can be very difficult. Thus, how much code coverage the acceptance tests cover will be used as a supporting metric.

3.3.4.3 Issue reporting

The app has a built in issue reporting tool called Crashlytics as described previously in Section 3.3.3.3. The number of issues reported by the new release will be compared to the number of issues reported for previous versions. However, if the team is not confident that the version at hand has enough quality to be promoted from the alpha channel to the production channel there will be very few users using the application. Thus, it is likely that number of issues reported will be fewer than for previous versions.

3.4 Threats to validity

This section discusses threats and risks that can affect the validity of this study. The validity of a study is important because it denotes the trustworthiness of the results. The results should be true and not be biased by the researchers subjective point of view [24]. Thus, it is necessary to identify and discuss possible threats to the validity of a study.

According to Runeson and Höst [24], there are different ways to classify aspects of validity and threats. They also describe a classification scheme to assess threats to validity in software engineering. The scheme distinguishes between four different aspects, *construct validity*, *internal validity*, *external validity* and *reliability*. Following sections describe these aspects and how they relate to this study.

3.4.1 Construct validity

> *Construct validity* concerns generalization of the study's result to concept or theory behind the study [36].

In other words, *construct validity* is about to what extent the operational measures that are being studied reflect the researchers intention [24]. E.g. an interview question could be interpreted differently by the researcher and the participant.

In this study few different methods were used in order to increase the construct validity. The interview questions were kept high level to allow participants with little

domain knowledge to take part. Before each interview the interviewees were encourage to ask questions during the interview if they were unsure about the meaning of the question. Further, if the researcher felt the need to provide a deeper understanding about the asked topic, he presented a small introduction about it. However, as interviewees come from different backgrounds there is always a chance that some questions might have been misinterpreted.

The implementation result interview was a structured interview. Having it structured was done in order to minimize the influence of the researcher during the interview. However, only the fact that the researcher was present, watching the subjects answer, might be considered a threat to the construct validity. It could have been performed as an anonymous survey but the developers still would have known that the researcher was performing it. Thus, having the interview as anonymous survey would not have reduced this threat.

3.4.2 Internal validity

> *Internal validity* concerns matters that may affect the study's results with respect to causality, without the researchers knowledge [36].

Feldt et al. [37] describe *internal validity* in a similar way, i.e., that it concerns matters that may affect the result or outcome of a study, with respect to causality, without the researchers knowledge.

The fact that the researcher is performing the study within the company can be a threat to the study's *internal validity*. The researcher is involved in daily activities within the company, both during daily working hours and social activities. Thus, the researcher creates relationships between him and the participants which is a threat to the internal validity due to few reasons. First, the researcher is more likely to be suggestive while conducting interviews. Second, unscheduled discussions, which are not recorded, can affect the outcome or the interviewees and researcher's opinions. Finally, the participants are more likely to answer the researcher in good will rather than criticizing, in order to keep the social relationship good.

In order to increase the internal validity the researcher tried to keep study related tasks as much separated from daily business, at the company, as possible. That is, he tried not to influence the team with study related things. However, if there was a need for notification or presentation of a newly implemented functionality, it was achieved by having a presentation or by sending out a group message. Further, all data collection methods were performed in as professional way as possible, e.g., making sure interviews would not wander off topic.

3.4.3 External validity

> *External validity* concerns generalization of the study's result to other environments than the one in which the study is conducted [36].

According to Wohlin et al. [36], *external validity* is mainly about if the results from

the study can be generalized outside the context of the study. That is, if the same study could be carried out in different context, but still come up with the same results. For this study, the biggest threat to validity are the ones that concern external validity.

The case company is small and the study is limited to it. The company as a whole will be included in the study but CD will only be adopted within a single team. The team that will be studied is the Android team and one of their application will be adopted to CD. Thus, generalizing the results can be difficult as the participants are few and the application under study is small and not in active development.

Action research is an iterative process and therefore multiple iterations should be performed with aim for improvements during each iteration. This study will start with introducing acceptance testing and then set up the environment and configurations needed for CD. Doing all this is a time consuming process. The study is limited to about 20 weeks which only allows for a single iteration. Not being able to go through multiple iterations might affect the results and make it more difficult to generalize the results.

3.4.4 Reliability

> *Reliability* concerns with to what extent the data and the analysis are dependent on the specific researchers [24].

Wohlin et al. [36] describe reliability threats as concerns with to what extent the data and the analysis are dependent on the specific researcher. That is if another researcher would do the study in the same context, would come up with the same conclusions. It is difficult to say for sure that another researcher would come up with the same conclusions. Especially in a study like this which depends a lot on implementation.

Having planned so much implementation as a part of the study contains some risk. It requires both attention and resources from the case company as well as dedication and knowledge from the researcher. Fortunately the researcher has a lot of experience with setting up CI and CD which will be valuable to the process but could also influence the conclusions. Another possible risk worth mentioning is that implementing the automated acceptance tests is required and devoting time for that might be difficult for the software engineers of the company during the time period.

In order to get over some of the barriers with CD adoption, multiple tools and techniques will be introduced. Because of that, the focus point of the subjects can be influenced by all the tools and techniques instead of the study's aim. Thus, in the implementation interview the subjects might build their responses from the whole experiencing rather than the topic being asked about specifically which is a threat to the reliability. In order to minimize this threat, the researcher will emphasize the specific topic being asked for in each question.

3.5 Ethical considerations

Bryman and Bell [30] discuss the necessity to consider ethical issues when carrying out a research. Andrews and Pradhan [38] state that research study is built on trust between the researcher and the case but even though, Runeson and Höst [24] emphasize that explicit measures must be considered to prevent ethical problems. Further, Runeson and Höst [24] state that ethical issues should be considered from early on and throughout the study. Finally, they describe few ethical factors required to consider when conducting a study. This section describes these factors and how they are approached for this study.

The first factor is *informed consent*, which means that all subjects must explicitly agree to participate in the study. It also means that the researcher must collect data with consent and not by using indirect or independent data collection methods. The researcher prevented uniformed consent by explicitly asking the participants for consent in all data collection methods, e.g. interviews, before it was performed. All subjects were given a choice if they approved to taking part in the study without any pressure.

The second factor Runeson and Höst discuss is *confidentiality*. They emphasize the fact that issues about confidentiality and publication should be regulated in a contract between the researcher and the case company. In this study no confidential data about the company is a part of the study so that will not be taken into consideration. However, employees opinions might be sensitive for different reasons. In this case, the researcher must have the right to keep their integrity and adhere to agreed procedures. For this study it is also necessary to consider not to characterize the subjects as the company is small and employees can be easily identified.

Inducement is the third factor they mention. They discuss that there are always some kind of incentives, tangible or intangible. For this study it is difficult to prevent inducement since the researcher is located in the company during the research and takes part in daily activities. This is considered to be a threat to validity as described in Section 3.4.

The final factor to consider is *giving feedback* to the participants. Doing that creates trust between the researcher and the participants which can increase the internal validity of the research as well as having negative effect on the external validity. In this case the transcripts of the interviews were shared with the participants and they were allowed to comment on it if they found some errors in the transcription. Further, the analysis and the results were presented to them in order to gain more trust, even though they might not necessarily agree with the results.

4
Results

This chapter presents the results of the data collected during the study as well as the challenges faced. A description of how the data was collected and how it was analyzed can be found in Sections 3.3.3 and 3.3.4 respectively. In addition to the results for the data collected, the challenges and achievements for the implementation phase will be presented.

4.1 Introduction interview

The aim of the introduction interview was to gain understanding about the company and the team's knowledge and opinion of CI and CD. Further, to find out if they are confident that their applications can be adopted to CD. Three Android developers were interviewed and each interview was approximately 25 minutes. Overall they had little knowledge about CI and CD and what the aim of the researcher was. Thus, conducting these interviews proved to be beneficial for both the researcher as well as the team. A good way of getting common understanding of the topic and the study.

The developers that participated in the interviews all had little experience with CI and CD but were familiar with the concepts. They had tried using CI but not CD. Further, they described CI as a way of working with software development that introduces more structure and higher quality with automation for otherwise time consuming tasks. The developer that had the most experience with CI also emphasized that having CI in place encourages knowledge spreading, e.g., allowing any developer to perform a release.

The case company is well suited for the adoption of CD. The reason for that is that the teams are self-managed and if they believe that introducing a new process is beneficial, they get full support from management. Further, Forza News would be a good fit for experimenting with continuous deployment as it does not have much functionality and is not in active development. Writing acceptance tests for it should not be too difficult and it would probably be enough to only have acceptance tests. However, the developers believed that, for a big applications, writing and maintaining good test suite would be expensive and difficult. One developer mentioned that having only automatic test suites for an application that relies much on user experience, can give you a false security. The tests might be running successfully but the user could still be experiencing difficulties.

Having binaries automatically deployed to a location accessible by the company is something they were all attracted to. A common problem they face have had is when testing a specific fixed issue, the version the tester had was not the right one. Thus, introducing automation for in-house binary distribution is desired. Further, having a well defined process for releases, with as much automation as possible, was something that interested them a lot. One developer did though mention that he would prefer doing continuous delivery rather than deployment as he liked having to give his acceptance by clicking a button to deploy.

Finally, a small part of the interview was about what kind of tools and services to use. The requirement proved to be that spending a lot of time maintaining the tools and services was not desired. There is a trade-off between having more functionality and a lot of maintenance, versus having less functionality and less maintenance. Since the team is small, they believed the latter option was more suitable.

4.2 Implementation

This section describes the results from the implementation part of the study. The implementation part consisted of implementing CD and automated acceptance tests as well as supporting scripts.

4.2.1 Branching model

Introducing the branching model to the team was rather effortless as the team was very familiar with git and branching in general. The researcher set the branching model up for Forza News and described to the team how to work with it. As a part of a status presentation a visual explanation of the model and its main functionality was introduced. Later, the team members tried following the model using the git extension with the assistance of the researcher. While the team was overwhelmed and unsure whether or not they wanted to use the model at first they quickly adapted to the model and were positive and wanted to use it for other projects as well.

4.2.2 Acceptance tests

In the beginning of the study, the application being studied had extremely limited automated tests. Thus there was a need to implement automated acceptance tests as the study aims to figure out if CD can work well by only having limited automated acceptance tests.

The QA manager of the company designed test scenarios which covered the most critical features of the application; her requirements for acceptance. Thus, by implementing theses scenarios as automated tests could be considered enough for acceptance. The scenarios were described in a detailed way so that the developers could start writing the automated tests without having to get more information from the QA manager.

In the introduction interview the team was asked if they had any experience with acceptance testing for Android. As their experience with acceptance testing was low, the researcher initially investigated testing frameworks available for Android. There are multiple frameworks available but a decision was taken, in collaboration with the team, to use a framework called Espresso. Espresso is a framework created and developed by Google. It is provided by the Android Testing Support Library and provides APIs for writing automated tests which simulate the user interactions [39], which makes it a candidate for writing automated acceptance tests. The reason why Espresso was chosen is mainly due to the fact that it is a part of the default support libraries for Android development and is maintained by Google who are also the creators of Android.

The API for writing tests with Espresso is not difficult to use and a lot of documentation exists about it. An example on how to interact with the user interface using Espresso can be seen in Listing 4.1. The listing presents a way of using Espresso to find a view with a certain id and performing a click on that view. Thus, writing the tests was not so difficult to begin with. However challenges surfaced when running the tests on different Android devices, physical and virtual. Since Android runs on devices from different manufacturers, with different screen size and hardware, some of the tests had to be modified in order to support all devices. This proved to be a challenging, time consuming task, mainly since running the test on slow emulators could take a long time. Eventually some test scenarios had to be modified a little bit in order to support some of the devices. These modifications were required because with smaller screens, fewer items were listed on the screen and therefore the framework could not locate them.

```
onView( withId( R.id.next_button ) ).perform( click() );
```

Listing 4.1: Clicking a button with a certain id using Espresso

In total, ten scenarios were implemented as automated tests using Espresso. Each test had multiple assertions and some also verified that expected database changes were successful. The code coverage percentage that these scenarios covered was *45%* of the instructions of the code and *36%* of the branches the code has. The coverage results can be found in Appendix E. Most of the test cases were implemented by the researcher due to time constraints the team had. However the researcher made sure to take time with the team to explain how the framework works by presenting the test cases and their functionality. Finally all the team's developers tried running the tests and write some minimal cases in order to get to know the framework so that they could maintain and extend the newly implemented scenarios.

4.2.3 Continuous deployment solution

Different solutions exist for continuous integration and deployment. In this study four different solutions were investigated. The solutions' characteristics are described in Table C.1 in Appendix C. The researcher and the team finally came to the conclusion to use CircleCI. During the introduction interview, all team members agreed

on that they wanted the solution to be chosen to be more or less maintenance free. Thus TeamCity and Jenkins were excluded from the original four solutions as they are self-hosted and require more maintenance. Therefore the choice was between Travis CI and CircleCI. The investigation of these to showed that both solutions are slow when it comes to running tests for Android as well as building binaries. Further at the time, CircleCI had better support for Android applications and the researcher came to the conclusion that configuring the build for CircleCI was simpler than for Travis CI. Therefore the decision to use CircleCI, was taken.

CircleCI has a good support for Android projects and setting up the build was not difficult. The initial build only included building the binaries. However when running the automated tests challenges surfaced. As the tests are run on a virtual device, on a headless server, it is often difficult to find out why tests fail. Thus it took a while to fix minor issues that only surfaced on the server and not when running locally. Unfortunately some tests continued failing, irregularly on the CI server. Tests that had never failed when running locally, neither on an emulator nor a physical device. No good reason for this issue was found but most likely it has something to do with the emulator being slow or if it did not even start.

4.3 Implementation results interview

After having finished automatically deploying the software for the first time a structured interview was conducted in order to understand how confident the team was with the CD process, the results and supporting tools. The interviews were conducted with three Android developers about how confident they were of continuously deploying the application by following the processes and using the tools introduced during the study.

The developers are confident of deploying the application in the current state and that the amount of acceptance testing that has been implemented is sufficient for continuously deploying the application to Play Store alpha channels. They think that the tools and processes work well together and that the tests cover the functionality they would have tested themselves.

According to the developers, having automated acceptance tests increases the overall quality of the application and decreases the likelihood of missing out when testing certain features manually before each release. However, they all agreed on that they would like to have some manual testing as a part of the deployment process. This manual testing should emphasize on user experience and verify that new translations fit well in to the user interface. Additionally it was mentioned that having automated process for taking screen-shots for all of the supported languages would definitely speed up the process of verifying the translations. Further, when re-factoring the code it could affect places in the code that will not be tested during manual testing phase and the automatic tests could cover that. It was also mentioned by one of the developers that, when issues come up, he would like to align a new process for fixing issues in a way that an automatic test is written to reproduce the error before

fixing it.

As a part of the study, new processes were introduced to suit CD. The team was really positive about the new processes. They believed that they could improve their own productivity and give them time to focus on further development instead of spending a lot of time on each release and distributing binaries within the company. They believe that the new release process is really thorough and structured compared to what they currently have and definitely an improvement. It might take time to learn to follow these processes but eventually it will improve productivity and quality. Further, having these processes structured and thorough gives the opportunity to more easily teach others how to release which makes releasing more flexible.

Having said that, overall the developers were confident of continuously deploying Forza News to Play Store alpha channel. They were really positive about continuing this process for Forza News and excited about implementing some of the steps implemented during the study for Forza Football as well. However, they emphasized that for an application as big and as business critical as Forza Football, they would always include some manual testing.

4.4 Issue reporting

Forza News, the application that was automatically deployed to Google Play alpha channel, was quickly promoted to the production channel. Since it was deployed, monthly active users are around 500, which is a small decrease in active users since the previous version. Further some issues, where users have been experiencing crashes, have been reported via Crashlytics. However, all of the issues reported have been reported in previous versions as well, i.e., no new issues in the automatically deployed version.

5
Discussions

This chapter provides a discussion about the different models, processes and tools introduced during the study. That is followed by a more general discussion about the objectives and findings of this study. Moreover the results and challenges of the study are discussed.

During the study, CD was introduced and implemented and as it requires developers to work in a disciplined way [10], the researcher decided to introduce models and processes to support CD. By doing that, the developer discipline should have been increased and therefore the developer confidence in CD. The focus of the study was to see how automated acceptance tests can support CD and how the results of these tests can determine if the release is successful. However the developers' opinion might have not only been affected by the tests alone, but also by the supporting models and processes. In order to explain how the models and processes impacted the study, each one of them is listed below; including a discussion on each one.

Automated acceptance tests

One of the study's objectives was to investigate how automated acceptance testing can support CD. The developers thought that the scenarios implemented were enough to continuously deploy Forza News, but still wanted to include some manual testing as well. By implementing the automated acceptance tests the developers were certain that the main functionality was tested and if some defects would surface they would fix them and redeploy. Even though they could not force updates, they thought as long as the main functionality was working as expected, they were satisfied. However, it is worth mentioning that Forza News, the application that was tested, is not business critical. While the Android team might do CD for Forza Football in the future, they are not likely to do it without having a manual test phase in the release process. Thus, if no manual testing is in place, the necessity of having at least automated acceptance tests for an Android application is high.

GitFlow

In the beginning of the study the team had a lot of different branches where some had not been worked on for a long time. The developers were sometimes pushing code directly to the master branch and sometimes fixes were not merged correctly. Thus, to put more restrictions and work with a more structured way of branching the researcher decided

to introduce a branching model. Due to the fact that all code that is pushed to the master branch gets automatically deployed, not having a certain branching structure increases the risk of having unfinished code automatically deployed. Having a branching model does reduce the risk of this happening even though it does not eliminate it. The developers were very happy with GitFlow and they have even started using it for Forza Football. The introduction of GitFlow did help them to work in a more disciplined way. Thus, it does help following a branching model when doing CD but it can not be considered a requirement.

Development process

The introduced development process was mainly to show the team how to include CI and how it relates to CD. The development process is influenced by the branching model as it requires a certain way of working with the code-base. Hence, it was not necessary to introduce a certain process for development but the researcher believed that it would help the transition to CI and CD. Showing the development process graphically proved to be a good idea. It helped explaining the role of the continuous integration server, and what happens when they committed their code. Further, showing the graphical explanation of the branching model and the development process together allowed the developers to see how to use the model and the process together.

Release process

The introduced release process was mainly to show the team how the branching model and the CI solution would work together. Moreover, it was to show graphically what needed to be done manually and what the CI solution would do automatically. Hence, it was not necessary to introduce a certain process for the release process. However, the researcher believed that doing it would ease out the transition to CD and help when introducing the release process to other developers. Due to the fact that the researcher modelled both the new release process and the previous one allowed him to show clearly the benefits of the introduction of CD. The graphical explanation of both processes showed in a good way how much of the process became automated. One of the developers mentioned that having a well defined process for the release would not only help the current employees but also help them to explain to future employees how releases are performed.

All the different processes and models affected the adoption of CD. However, how much impact each one had was not the same. The main impact came from the automated acceptance tests. Without it it is unlikely that the developers would have accepted to do CD. The branching model also had a lot of impact but in a way that it increased the discipline of the developers. By doing that the developers did feel more confident of doing CD.

From a scientific point of view, the aim was of the study was to investigate how CD

can be implemented in small companies, by implementing automated acceptance tests. Therefore, there was a need to choose a CD solution which would suit the context, e.g., the team, the project and the research. Eventually the team and the researcher took the decision of choosing CircleCI, due to the fact that it is a cloud-based solution as well as the fact that it has good support for the context, e.g., the team and the application. Which solution is chosen can vary a lot between companies, teams and even developers. If CircleCI was the best solution for the task at hand is impossible to argue since there is nothing to compare. As mentioned in Section 3.3.2.3, the self-hosted solutions such as Jenkins, are more powerful as they support wider configurations. Thus, it would be interesting to do further research on this topic, using a more powerful CD solution, such as Jenkins, to find out if that would lead to different results, e.g., if it would increase the confidence of the developers of doing CD.

The CD implementation worked well with CircleCI and configuring the build was simple. However, building and running automatic tests for Android is very difficult because of the differences between devices, e.g., their hardware and their screen size. Wasserman [40] discusses that the majority of the development tools that exist for mobile development are predominantly focused on the developer who is trying to create an application as quickly as possible. When applications become more business critical, it is essential to apply software engineering processes to assure high-quality applications. Further, Wasserman [40] points out that testing mobile applications is complex as it is insufficient to merely test an Android application on an emulator; it must be tested across many different devices running different versions of the operating system. Thus, the current testing capabilities with respect to the variety of devices in the Android ecosystem is insufficient.

The choice of only having automated acceptance tests and no unit tests can be argued against. Stolberg [22] discusses that the CI server should compile the code, run unit tests and then run other tests such as integration, functional and acceptance tests. Due to the fact that the code is heavily coupled, it would have taken a lot of effort to refactor it in a way high quality unit tests could be easily implemented. The team is small and they could not invest the time and resources to proceed with that sort of refactoring. Moreover, having only limited time for the implementation, being able to cover as much of the functionality as possible, required a lot less development work by implementing only acceptance tests. However, since there is much easier to isolate bugs and issues with unit tests [41], the researcher recommended and encouraged the team to write unit test in addition to further acceptance tests in the future.

Rossi et al. [5] discuss the CD process, for mobile applications, at Facebook which can be considered a mature CD process. The process includes a manual test phase at the later stages, before the deployment is performed. Thus, the implementation results interview, which is described in Section 3.3.3.1.2, included questions about introducing a manual test phase to the release process and how it would affect their confidence in CD. The developers all agreed on that. Interestingly, they all mentioned that the manual tests should focus on exactly the same thing Facebook

focuses on with their manual tests, e.g., translations, user experience and new features. However, one developer mentioned that he thinks that it is enough to have only automated acceptance tests but emphasized that having manual testing as well would increase the application's quality. It feels that doing manual tests is more of a human safety factor which increases the developers confidence. The developers do not feel that having only automated acceptance tests on the user interface can cover the human experience of using an mobile application. Leppänen et al. [3] stated that developer confidence is one of the big challenges for CD and the results of this study agrees with that. Thus, future research in this area should hence investigate the role the human factor plays in implementing CD for mobile applications.

All of the developers interviewed raised a concern that user experience can not be tested and that they would like to include a manual test phase. However, we did not ask any further questions concerning user experience testing as we deemed it out of scope of this study. User experience consists of different aspects, some of which can in fact be tested. Interaction time can for example be tested and that is one aspect of user experience. It seems like the developers were thinking more of user experience aspects such as look and feel, and navigation paths. As the company does a lot of user research and A/B testing they are really concerned about that their users can easily find the information they are looking for in their applications. Thus, it would be interesting to ask further about what aspects of the user experience the developers are concerned with not being tested and what they think is the main risk if manual testing would be completely substituted by automatic tests.

Unfortunately, there has not been so much development for Forza News. Thus, the results from the issue reporting tool were about the same for the newly automatically deployed version and the version before. Since there was not really any change between the versions, the results from the issue reporting tool can not be considered as a deciding factor when concluding if the release was successful or not.

The study introduced new development and release processes which align well with CD. The developers all agreed on that these processes were a good fit for doing CD in this context. They were very positive about the automated distribution of binaries and emphasized that it had been an issue for them in the past. Interestingly, when asked if the new process would improve his productivity, one developer discussed that by having processes and the automation introduced, allows him to use his time in a more productive way. He mentioned that he would like to improve their testing coverage and continue with automated acceptance tests. Further, he wanted to extend the processes introduced by integrating the way of handling reported defects. When defects are reported, they should be reproduced by the developers via an acceptance test and then fix it. By doing this, they would continue to maintain their knowledge of the testing framework as well as improving the quality of the application.

Chen [9] discusses in his paper that further research is required for applications that are not amenable to continuous delivery. Chen only mentions one type of applications that are not amenable continuous delivery; namely large, monolithic

applications. This study was done for an application that lacked automated testing; thus the researcher considered it not amenable to CD. However, there should be more research investigating what kinds of systems cannot easily be tested and deployed automatically in continuous fashion.

The application and the team under study are small so further research on CD should be performed with larger teams and larger systems. Further, research comparing how team and company size affect CD with focus on the human factor should be performed. It can be argued that it seems to be beneficial to get automated acceptance testing and CI running, especially for new/small products and teams. However, to get to the full scale, i.e., deploying the application automatically, includes a large overhead plus a human factor - so it might end up being impossible or at least not feasible. The team members all agreed on, that introducing some automation helps them to be more productive.

Only one iteration was performed as a part of this study. If further iterations would be performed, the following topics should be addressed.

1. The introduction of a device farm
2. The introduction and comparison of another CI solution
3. Investigate unexpected test failures
4. More automated acceptance tests

First, a device farm is an app testing service that lets you test and interact with mobile applications on many devices at once [42]. Running tests on multiple devices increases the likelihood of finding device specific issues which could also impact developer confidence in CD. Secondly, the CI solution itself could have impacted the results. Thirdly, the unexpected test failures should be investigated as they could affect the developers opinion on CD. Soon after the first iteration, CircleCI rolled out a new beta version where they promise faster builds which could affect the unexpected test failures. Finally, it would be interesting to see if more automated acceptance tests would have any impact on developer confidence or their opinion on testing user experience.

In summary, **RQ.1** can be answered as follows. Automated acceptance testing can be fully automated in order to support continuous deployment for small applications which are not business critical. However, with larger applications, which are business critical, developers believe that some manual testing must be involved. According to the developers, the reason why the business critical applications can not be fully automated, is because there are some aspects of the user experience that need to be tested manually. Further, **RQ.2** can be answered as follows. The results of automated acceptance tests can determine if a release for small applications is successful. A successful release is whenever the developers believe that the quality and the defect count is acceptable. However, manual testing and some human interaction is required for larger, business critical applications (**RQ.2**).

6
Conclusions

Fast-changing and unpredictable markets in the software industry lead to challenging and complex customer requirements. Software development companies are required to meet these requirements and react quickly [3]. In order to do so, practitioners and researchers have been paying increased attention to improve software development processes [5]. In recent years, the amount of automation in software development has increased with the introduction of practices such as automated tests, automated builds and automated deployments.

The process of deploying software automatically is called CD and is an extension of CI. CI is when every code change triggers the running of different automated tasks such as building projects, running tests and storing outputs. Additionally to the tasks CI consists of, CD automatically deploys the software if the preceding tasks have run successfully. CD for mobile applications is more complex and different from CD for cloud-based applications. The main difference is that cloud-based applications can be redeployed easily without having any interaction from the user whereas mobile applications require the user to confirm the update [5]. Thus, it is considered having higher risk to do CD for mobile applications than for cloud-based applications.

This study consisted of introducing new concepts, techniques and tools to the case company. It was an action research where CD and automated acceptance tests were implemented and then the results were reflected on. It was performed in collaboration with the Android team of the case company and therefore the applications under study were Android applications.

The main purpose of the study was to learn about the information need, risks and challenges when adopting an Android application to CD. It aimed to find out to what extent acceptance tests can be automated to support continuous deployment. Further, it investigated how the results of automated acceptance tests can determine if a release is successful.

Having well defined processes and automation proved to support the adoption of CD. The introduced processes allowed the developers to adapt to and understand CD quickly. It allows them to devote less time on building, releasing and distributing the application. Further, the feedback loop of issue discovery as well as the time between releases has been shortened which aligns with previous studies such as [3, 10]. Thus the developers can use their time in providing the users new versions

of the application, with the reported issues fixed, more rapidly. However, when issues are reported, a certain process should be developed, in order to respond quickly. The issue should be reproduced by writing an automated acceptance test, fixed and then released.

The results of this study show that small Android applications can be automatically deployed having only limited automated testing. The developers were confident in continuing automatic deployment for the application under study if automated tests were written for new features, i.e., having only automated acceptance tests gave them confidence to continuously deploy. However for bigger, more business critical applications, the developers expressed that they would not trust CD without having a manual test phase in addition to the automated acceptance tests. They feel that it is not possible to test user experience automatically and since it is one of the biggest factor for user retention they feel that manual testing should be a part of the release process.

Getting a good coverage of code and features can be achieved easily when writing automated acceptance tests for a small Android application. The testing framework, Espresso, has a powerful API which allows for good user interface interaction. However, maintaining the tests and making sure that they run successfully on the big variety of different devices in the Android ecosystem is difficult. Additionally, the larger the application gets, more time is required to maintain the tests and the more difficult it is to achieve good coverage. Developers confidence decreases when the size of the application increases as more time is required to achieve specifiable coverage. However having automated tests and processes does increase the quality of the release.

The results of the study show how CD, for Android applications, can be introduced and implemented in a small company. It has identified different sorts of challenges and benefits of CD. Chen [9] states the need of doing research for applications that are not amenable to CD. This study is done with an application that was, beforehand, not amenable to CD. The study shows that the application can be considered amenable to CD with limited automated acceptance. However, further research is required about the topic. Both for applications that are not amenable to CD due to different reasons as well as how company and team size affects the introduction of CD. Additionally, it would be very interesting to investigate how developer confidence in CD is different if automated acceptance tests are implemented from the start of development.

References

[1] V. Driessen, "A successful git branching model." `http://nvie.com/posts/a-successful-git-branching-model`, 2010. Accessed: 2017-03-01.

[2] T. Savor, M. Douglas, M. Gentili, L. Williams, K. Beck, and M. Stumm, "Continuous deployment at facebook and oanda," in *Proceedings of the 38th International Conference on Software Engineering Companion*, pp. 21–30, ACM, 2016.

[3] M. Leppänen, S. Mäkinen, M. Pagels, V.-P. Eloranta, J. Itkonen, M. V. Mäntylä, and T. Männistö, "The highways and country roads to continuous deployment," *IEEE Software*, vol. 32, no. 2, pp. 64–72, 2015.

[4] P. Järvinen, "Action research is similar to design science," *Quality & Quantity*, vol. 41, no. 1, pp. 37–54, 2007.

[5] C. Rossi, E. Shibley, S. Su, K. Beck, T. Savor, and M. Stumm, "Continuous deployment of mobile software at facebook (showcase)," in *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pp. 12–23, ACM, 2016.

[6] A. Fuggetta, "Software process: a roadmap," in *Proceedings of the Conference on the Future of Software Engineering*, pp. 25–34, ACM, 2000.

[7] N. Dzamashvili Fogelström, T. Gorschek, M. Svahnberg, and P. Olsson, "The impact of agile principles on market-driven software product development," *Journal of software maintenance and evolution: Research and practice*, vol. 22, no. 1, pp. 53–80, 2010.

[8] M. Fowler and M. Foemmel, "Continuous integration," *Thought-Works) http://www.thoughtworks.com/ContinuousIntegration.pdf*, p. 122, 2006.

[9] L. Chen, "Continuous delivery: Huge benefits, but challenges too," *IEEE Software*, vol. 32, no. 2, pp. 50–54, 2015.

[10] G. G. Claps, R. B. Svensson, and A. Aurum, "On the journey to continuous deployment: Technical and social challenges along the way," *Information and Software Technology*, vol. 57, pp. 21–31, 2015.

[11] J. Humble and D. Farley, *Continuous delivery: reliable software releases through build, test, and deployment automation*. Pearson Education, 2010.

[12] X. Fang and G. Salvendy, "Customer-centered rules for design of e-commerce web sites," *Communications of the ACM*, vol. 46, no. 12, pp. 332–336, 2003.

[13] T. Pradhan, "Mobile application testing," *TCS White papers,[online]. Available: http://www.tcs.com/SiteCollectionDocuments/White%20Papers/Mobility_Whitepaper_Mobile-Application-Testing_1012-1.pdf. [Accessed: 2017-01-18]*, 2011.

[14] H. H. Olsson, H. Alahyari, and J. Bosch, "Climbing the" stairway to heaven"–a mulitiple-case study exploring barriers in the transition from agile development towards continuous deployment of software," in *2012 38th Euromicro Conference on Software Engineering and Advanced Applications*, pp. 392–399, IEEE, 2012.

[15] B. Haugset and G. K. Hanssen, "Automated acceptance testing: A literature review and an industrial case study," in *Agile, 2008. AGILE'08. Conference*, pp. 27–38, IEEE, 2008.

[16] A. W. Services, "Aws case study: Football addicts." `https://aws.amazon.com/solutions/case-studies/football-addicts`, 2017. Accessed: 2017-02-23.

[17] A. Fuggetta and E. Di Nitto, "Software process," in *Proceedings of the on Future of Software Engineering*, pp. 1–12, ACM, 2014.

[18] K. Beck, "Embracing change with extreme programming," *Computer*, vol. 32, no. 10, pp. 70–77, 1999.

[19] D. Turk, R. France, and B. Rumpe, "Assumptions underlying agile software development processes," *arXiv preprint arXiv:1409.6610*, 2014.

[20] K. Beck, *Extreme programming explained: embrace change*. addison-wesley professional, 2000.

[21] P. Hsia, D. Kung, and C. Sell, "Software requirements and acceptance testing," *Annals of software Engineering*, vol. 3, no. 1, pp. 291–317, 1997.

[22] S. Stolberg, "Enabling agile testing through continuous integration," in *Agile Conference, 2009. AGILE'09.*, pp. 369–374, IEEE, 2009.

[23] C. Robson and K. McCartan, *Real world research*. John Wiley & Sons, 2016.

[24] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," *Empirical software engineering*, vol. 14, no. 2, pp. 131–164, 2009.

[25] B. Dick, "Action research: action and research." `http://www.aral.com.au/resources/aandr.html`, 2002. Accessed: 2017-01-16.

[26] P. Reason and H. Bradbury, *Handbook of action research: Participative inquiry and practice*. Sage, 2001.

[27] S. Easterbrook, J. Singer, M.-A. Storey, and D. Damian, "Selecting empirical methods for software engineering research," in *Guide to advanced empirical software engineering*, pp. 285–311, Springer, 2008.

[28] M. Brydon-Miller, D. Greenwood, and P. Maguire, "Why action research?," *Action research*, vol. 1, no. 1, pp. 9–28, 2003.

[29] D. E. Avison, F. Lau, M. D. Myers, and P. A. Nielsen, "Action research," *Communications of the ACM*, vol. 42, no. 1, pp. 94–97, 1999.

[30] A. Bryman and E. Bell, *Business research methods*. Oxford University Press, USA, 2015.

[31] V. Driessen, "gitflow." `https://github.com/nvie/gitflow`, 2017. Accessed: 2017-04-26.

[32] Square, "Spoon website." `http://square.github.io/spoon/`, 2017. Accessed: 2017-03-19.

[33] C. Cassell and G. Symon, *Essential guide to qualitative methods in organizational research*. Sage, 2004.

[34] P. Anton, "On software testing," 2001.

[35] Fabric, "Fabric website." https://get.fabric.io, 2017. Accessed: 2017-03-06.

[36] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in software engineering*. Springer Science & Business Media, 2012.

[37] R. Feldt and A. Magazinius, "Validity threats in empirical software engineering research-an initial survey.," in *SEKE*, pp. 374–379, 2010.

[38] A. A. Andrews and A. S. Pradhan, "Ethical issues in empirical software engineering: the limits of policy," *Empirical Software Engineering*, vol. 6, no. 2, pp. 105–110, 2001.

[39] Google, "Espresso testing." https://developer.android.com/training/testing/ui-testing/espresso-testing.html, 2017. Accessed: 2017-04-26.

[40] A. I. Wasserman, "Software engineering issues for mobile application development," in *Proceedings of the FSE/SDP workshop on Future of software engineering research*, pp. 397–400, ACM, 2010.

[41] R. Patton, *Software testing*. Sams publishing, 2001.

[42] Amazon, "Amazon website." https://aws.amazon.com/device-farm/, 2017. Accessed: 2017-05-09.

[43] Jenkins, "Jenkins website." https://jenkins.io/, 2017. Accessed: 2017-03-16.

[44] Jetbrains, "Team city website." https://www.jetbrains.com/teamcity/, 2017. Accessed: 2017-03-16.

[45] TravisCI, "Travis ci website." https://travis-ci.com, 2017. Accessed: 2017-03-16.

[46] CircleCI, "Circleci website." https://circleci.com, 2017. Accessed: 2017-03-16.

[47] D. Kovač, "Continuous integration on android: Why we ditched jenkins for circle ci." https://infinum.co/the-capsized-eight/continuous-integration-on-android-why-we-ditched-jenkins-for-circle-ci, 2015. Accessed: 2017-03-16.

A

Abbreviations

Abbreviation	Description
CI	Continuous integration
CD	Continuous deployment
LoC	Lines of code
QA	Quality assurance
API	Application programming interface

Table A.1: Abbreviations

B

Facebook Testing Range

Unit tests:	These white-box tests primarily verify the logic of target units.
Static analysis:	This analysis identifies potential null-pointer de-references, resource leaks, and memory leaks. Because exceptions are often the root cause of resource leaks, the tool is careful to identify where these particular leaks might occur.
Build tests:	These tests determine whether the code builds properly.
Snapshot tests:	These tests generate images of screen views and components, which are then compared, pixel by pixel, to previous snapshot versions
Integration tests:	These standard (blackbox) regression tests test key features and key flows of the app. They typically run on simulators and are not necessarily device specific. Moreover they employ degrees of scope: "*smoke tests*" primarily target specific changes to the code (diffs) or target high-level functionality; long-tail integration tests run the full suite of regression tests and run against a live server so that they also cover client-server integration.
Performance tests:	These tests run at a mobile lab (device farm) to triage performance and resource usage regressions.
Capacity tests:	These tests verify that the app does not exceed various specified capacity limits.
Conformance tests:	These tests verify that the app conforms to various requirements.

Table B.1: Range of tests performed on mobile software at Facebook [5]

C

Continuous integration and deployment

	Description	Platform	Configuration
Jenkins	Self-contained, open-source automation server. Supports automation of all sorts of tasks as building, testing and deploying software. Is highly customisable as well as having big plugin support. Jenkins is free, widely-used and well documented and has vibrant user community [43]. As it is self-hosted and supports having build agents on multiple machines, it supports running highly complex tasks.	Self-hosted, java server.	Web UI as well as OS configurations.
TeamCity	Self-contained, continuous integration and deployment server created by JetBrains. Professional server is free for up to 20 build configurations. TeamCity is widely used and well documented solution [44]. As it is self-hosted and supports having build agents on multiple machines, it supports running highly complex tasks.	Self-hosted, java server.	Web UI as well as OS configurations.
Travis CI	A hosted, distributed continuous integration service. Supports different kind of automation such as building, running tests and deploying software. Has a lot of built in integrations for notifications, deployments and more. [45].	Cloud-based service.	Yaml configuration file. Stored within the version control system.
CircleCI	A hosted, distributed continuous integration service. Supports different kind of automation such as building, running tests and deploying software. Has a lot of built in integrations for notifications, deployments and more [46]. Has a good support for Android development [47].	Cloud-based service.	Yaml configuration file. Stored within the version control system.

Table C.1: Comparison between the four different CI solutions investigated

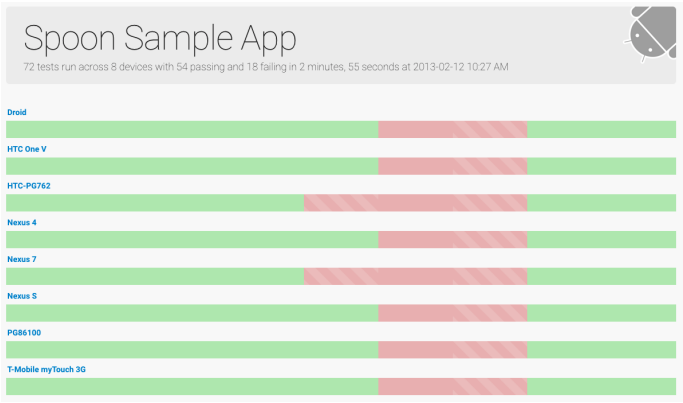

Figure C.1: An overview image of how Spoon displays list of devices where tests did run.

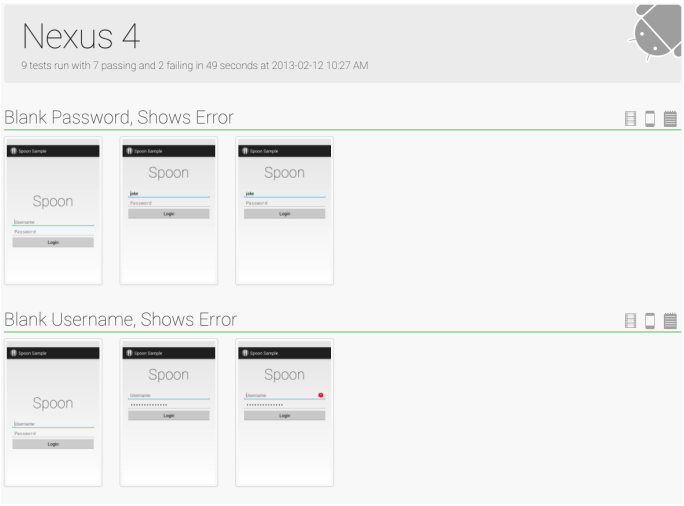

Figure C.2: An overview image of how Spoon displays test results for a single device.

Figure C.3: An image of the *device table* where devices are connected to a computer, charged and tests are run daily.

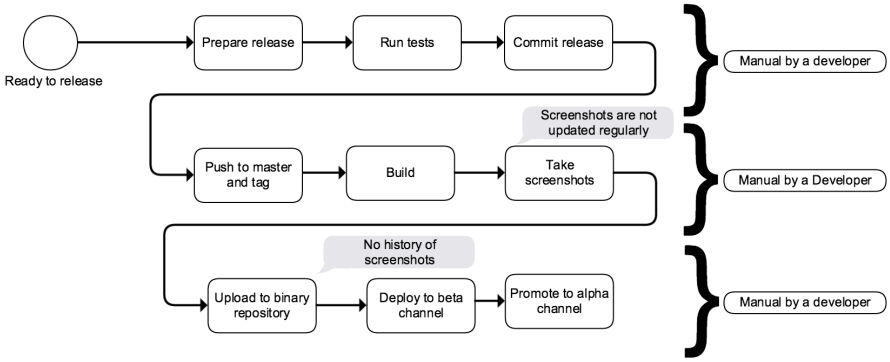

Figure C.4: A graphical explanation of the release process before CI and CD was introduced.

D

Interviews

D.1 Introduction interview guidelines

- What do you know about continuous integration (CI) and continuous deployment (CD)?
- Do you think Football Addicts as a company is well suited for CI / CD ?
  - Do you think management is willing to support this way of working?
- Do you think the application can be automatically deployed? If not, why?
  - If you had a small automatic test suite (acceptance tests), what kind of things / requirements could you test?
  - Do you think that would be change your opinion to deploy automatically?
  - What do you think are the major risks?
- How would you approach the adoption of CD for Android products here at Football Addicts? In respect to the team, management and overall process.
- Do you have any preference about the platform that runs the CI. Do you want to have it highly customizable or rather straightforward
- Have you done any research about Android automated testing tools? Tools such as: Robotium, UIAutomator, Espresso, Appium and Calabash. What do you think about using that kind of tools?
- In an ideal world, how would like to have the release process for Android products here at Football Addicts? Can it be completely automated or do you think there always has to be some manual steps?
- How extensive acceptance tests do you think are enough to start doing CD? Do you think it'll take a lot of time to implement? What about to maintain?

D.2 Implementation results guidelines

This interview was a *structured* interview where each question was on a scale. In addition to answering on a scale, the interviewees were allowed to add further comments about each question. The scale used can be found in the bullets list below.

- Strongly Disagree
- Disagree
- Neutral
- Agree
- Strongly Agree

D.2.1 Questionnaire guidelines

- I am confident of automatically deploying the application in the current state. Why? Why Not?
- I believe that the current amount of acceptance testing covers enough of the application functionality to continuously deploy it to Play Store alpha channel.
- Automated UI acceptance tests for Android application increase its overall quality.
- There is always some manual testing needed for Android applications.
  - If you answered agree / strongly agree in the previous question. For which parts of the system?
    - Backend Integration
    - User interface / user experience
    - Visible text translations
    - Other:
- A new development process has been designed, see the following document. Would you like to follow a process like this one?
- A new development process has been designed, see the following document. I am confident that this process will improve my productivity at work?
- A new release process has been designed, see the following document. I am confident that a process like this is a good fit for the Android team.
- A new release process has been designed, see the following document. Would you be more confident of the release process if a manual test phase would be introduced to it?
- Following a release process like previously introduced will improve the teams productivity.
- Following a release process like previously introduced will improve release quality.

E

Code coverage

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
se.footballaddicts.forzanews.settings		24%		18%	332	407	769	1,038	202	267	56	68
se.footballaddicts.forzanews.news		30%		17%	181	228	433	596	86	128	16	27
se.footballaddicts.forzanews.misc		59%		40%	196	320	380	897	83	166	5	21
se.footballaddicts.forzanews		43%		41%	139	238	321	630	75	151	18	37
se.footballaddicts.forzanews.model		28%		29%	152	252	293	440	107	200	7	29
se.footballaddicts.forzanews.remote.request		70%		77%	51	117	106	326	32	69	10	22
se.footballaddicts.forzanews.fragments		81%		62%	60	199	88	521	16	116	1	30
se.footballaddicts.forzanews.notifications		23%		12%	27	36	74	104	8	16	2	3
se.footballaddicts.forzanews.remote		10%		0%	28	35	56	68	22	29	5	7
se.footballaddicts.forzanews.view		42%		50%	23	35	58	96	20	30	1	5
se.footballaddicts.forzanews.actionbar		60%		55%	20	37	43	98	14	26	3	4
se.footballaddicts.forzanews.apprate		72%		45%	23	32	36	97	9	17	2	3
se.footballaddicts.forzanews.holder		42%		n/a	2	5	3	9	2	5	0	1
Total	12,523 of 22,948	45%	897 of 1,402	36%	1,234	1,941	2,660	4,920	676	1,220	126	257

Figure E.1: The code coverage report from running the automated acceptance tests