Implementation of a Continuous Integration and Continuous Delivery System for Cross-Platform Mobile Application Development

Samuel Nilsson

Supervisor: Anders Fröberg
Examiner: Erik Berglund
Upphovsrätt


Copyright

The publishers will keep this document online on the Internet – or its possible replacement – for a period of 25 years starting from the date of publication barring exceptional circumstances. The online availability of the document implies permanent permission for anyone to read, to download, or to print out single copies for his/hers own use and to use it unchanged for non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional upon the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility. According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement. For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its www home page [http://www.ep.liu.se/](http://www.ep.liu.se/).

© Samuel Nilsson
Abstract

When working in software development teams, there are challenges when it comes to always keeping the software stable and reliable. Continuous integration are frequently used to increase the stability and reliability. Extensive research has been performed on the matter of development processes of continuous integration, but there are no consensus on how systems to support continuous integration should be implemented for best results. In this report a continuous integration system is implemented based on best practices and to support the general continuous integration development process, by using Jenkins and other open source tools. The system is adapted to work well with the cross-platform mobile development framework CoffeeMaker developed by VISIARC AB and the general needs of the company. In order to roughly estimate the increased developer productivity and product quality when introducing the system, a questionnaire that discusses the system and working habits was sent out to the developers. The evaluation lead to the conclusion that the productivity would improve by approximately 30-60 minutes per week and developer. It also lead to the conclusion that the quality of their developed applications would most probably increase by introducing such a system.
Contents

Abstract iii

Contents iv

List of Figures vi

List of Tables 1

1 Introduction 2

1.1 VISIARC AB .................................................. 2
1.2 Aim ...................................................... 2
1.3 Problem description ......................................... 2
1.4 Research questions .......................................... 3
1.5 Approach .................................................. 3
1.6 Delimitations ............................................... 3

2 Theory 4

2.1 Continuous integration ...................................... 4
  2.1.1 Work flow and practices .................................. 4
  2.1.2 Continuous integration tools .............................. 4
  2.1.3 Unit testing ............................................. 6
  2.1.4 Visualizing continuous integration ....................... 6
2.2 Build automation ........................................... 6
  2.2.1 Important factors ....................................... 6
  2.2.2 Build iOS applications .................................. 7
2.3 Technologies used in CoffeeMaker .......................... 8
  2.3.1 Cocos2d-js ............................................ 8
2.4 Methodology .............................................. 8
  2.4.1 Case study research ..................................... 8
  2.4.2 Data collection ......................................... 9
  2.4.3 Productivity evaluation ................................ 10

3 Method 11

3.1 Architecture and implementation ......................... 11
3.2 Evaluation ............................................... 11
  3.2.1 Data collection technique .............................. 11
  3.2.2 Choice of method for data collection ..................... 12
  3.2.3 Define content of questionnaire ......................... 12

4 Result 14

4.1 Architecture ............................................ 14
  4.1.1 Version control system ................................ 14
  4.1.2 Visualizing continuous integration ..................... 14
# List of Figures

2.1 The typical developer work flow of continuous integration. . . . . . . . . . . . . . . 5
2.2 The framework architecture of Cocos2d-x taken from their Github page . . . . . . 8

4.1 An overview of the design of the continuous integration system implementation. . 18
4.2 An example of the branching model that the continuous integration system imple- 21
mentation is adapted for

A.1 A screenshot of the build list in the build monitoring view of the web application. 32
A.2 A screenshot of the starting page of the web application. . . . . . . . . . . . . . . . . 33
A.3 A screenshot of the build trend graph in the build monitoring view of the web 34
application. The horizontal axis shows the build number and the vertical axis
shows the duration of the build in minutes.

A.4 A screenshot of the test trend graph in the build monitoring view of the web ap- 34
plication.
List of Tables

3.1 The pros and cons of using interview versus online questionnaire as data collection methodology for the evaluation. .................................................. 12

4.1 Decision basis for the choice of communication channel ........................................ 15
4.2 The minimal requirements used as basis for the build monitoring application ........ 16
4.3 The minimal requirements of the continuous integration server ............................ 17
Chapter 1

Introduction

A problem with software development in teams is to always keep the software stable and reliable even though there are multiple developers working on the project. Technical resources used, such as version control systems, continuous integration servers, software testing, the complexity of the developed software as well as the behavior of the developers are all factors that affect the magnitude of the problem.

This thesis project focuses on how to set up a continuous integration and build environment, including a build monitoring system, continuous delivery, build automation and support for software testing. Extensive research has been performed on the development process of continuous integration but less on how to configure and set up an environment to support the process. Therefore it is not clear for a company that wants to use such an environment, how it should be set up to give a great result in form of increased stability and reliability. How to set up such an environment in a good way is what is investigated in this report. This report also focuses on challenges with continuous integration in combination with cross-platform development.

1.1 VISIARC AB

This project is performed in cooperation with the app and game studio VISIARC AB in Linköping. VISIARC AB develops a framework - Coffee - which is a framework for cross-platform mobile application development. On top of Coffee they have developed a visual development studio called CoffeeMaker. Coffee uses the open source game engine Cocos2dx-js and the Javascript engine Spidermonkey developed by Mozilla. The mobile applications are written in Javascript and compiled with Cocos2dx-js. Currently the framework supports development towards Android and iOS platforms.

1.2 Aim

The primary goal of this thesis project is to increase the quality and stability of the applications developed in CoffeeMaker. A secondary goal is to increase the developer's productivity by decreasing broken builds and avoid what M. Meyer [1] in his journal article calls the "it works on my local machine" syndrome.

1.3 Problem description

The employees at VISIARC AB have noticed that problems occur due to the fact that they don’t have a centralized location to build their software or the ability to get a working packaged copy of the application for testing or demonstration. Currently every employee checks out the project they are working on and commits their changes to the main line. However,
1.4 Research questions

The problem described in Section 1.3 ended in the following research question formulations:

- How can build processes be automated to decrease the time required to perform manual build steps?
- How can a continuous integration system be realized to increase developer productivity and product quality?

1.5 Approach

At first, a literature study will be performed to identify how continuous integration are typically implemented in practice and which effect different continuous integration practices and implementations have on developer productivity and product quality. The literature study will also cover testing configurations and metrics and the effect they have on product quality. Secondly, the manual steps of the current build process will be identified and automated. After that, a continuous integration system will be implemented, using the automated build process and the testing configuration. The continuous integration configuration will be based on the literature study about continuous integration adapted to the needs of the company. To measure the success of the study, the implemented system will be evaluated. How to evaluate the project will be decided based on how evaluations of similar studies have been performed earlier and how it is possible to measure developer productivity and software quality.

1.6 Delimitations

The realization of the continuous integration system only covers one type of implementation which is the one most suitable for the needs and the size of VISIARC AB. That means that the build system will be adapted to the tools and environments used by CoffeeMaker. The implementation will only include the use of free and open source software.
Chapter 2

Theory

2.1 Continuous integration

Continuous integration is a software development or more specifically an extreme programming practice where new code is integrated with the main line several times per day [2] [3]. The extreme programming practice was introduced by K. Beck [2] as he saw major flaws in the earlier used waterfall model [4]. Beck criticized that the waterfall model were based on the assumption that the users always know exactly what they want, which is rarely the case.

Continuous integration is originally a part of the extreme programming practice but focuses on the work flow of the developer team’s daily work [1]. The goal of continuous integration is to keep the software in a functional state all through the development process [5]. The practice of continuous integration is known for increasing both the productivity of the developers [3], the quality of the software and minimizing the time to delivery [6].

2.1.1 Work flow and practices

How companies actually implement and work with continuous integration depends on how they interpret it. Daniel Ståhl and Jan Bosch [7] made a literature review and concluded that the way of implementing continuous integration differs from case to case and that there are no consensus on how it should be implemented or interpreted.

However, the original explanation of the continuous integration concept is that the code should be integrated every few hours, and concurrently the source code is built and unit tests are ran to detect errors, if any [2]. How often code should be integrated, built and tested have been discussed leading to different conclusions, as described in Section 2.2.1. There are also other factors that are not clearly defined or discussed in the original definition, which also is described in Section 2.2.1.

Jez Humble and David Farley [5] promotes the developer work flow shown in the flow graph in Figure 2.1 as a way to always make sure that the software works on a machine with the same configurations as the continuous integration server, and increase the time the software is in a functional state.

The developers should avoid getting broken code [8]. Before retrieving the code the developer should therefore make sure that the build is not currently running or that the build is broken.

2.1.2 Continuous integration tools

It is important that it is possible to start the build of the software from the command line. This is because it is supposed to be able to be started automatically by a server. It also makes it easier for the developers to maintain the build process. The build command should build the application, test it and deploy it automatically. [5] . There exist a lot of continuous inte-
Figure 2.1: The typical developer work flow of continuous integration.
2.2. Build automation

Mathias Meyer [1] states that there exists a lot of continuous integration servers. He specifically mentions Jenkins, Teamcity and Bamboo as self-hosted servers and CloudBees and Travis CI as hosted ones. However, the literature review on continuous integration servers did not provide a comparison of such servers, but they all serve the same purpose.

2.1.3 Unit testing

Unit testing is a test of the smallest unit or units in the source code. The feedback from the tests should be given quickly. Opinions from employees at different companies state that the main reason to use unit tests is to test the functionality of the software, but another commonly stated reason is to increase the quality of the software. [9]

2.1.4 Visualizing continuous integration

It is important that the developers are notified quickly about the status of the build of the software under development, since detecting build errors in time means that they can be fixed quicker which also lowers the cost of fixing them [10]. John Downs, Beryl Plimmer et al. [11] believed that there at the time didn’t exist any best practices on how to design build monitoring systems with the purpose of informing the developers of the build status, so they performed a case study in order to conclude requirements of such systems. A brief explanation of the requirements are listed below:

- The system should make sure that the entire team is aware of problems in the build, which applies a pressure to solve them.

- The system should highlight when a developer has broken a build and who it was that made the changes that lead to a broken build. However it is important that the team are aware of the fact that broken builds aren’t necessarily that bad and that the system presents the information in such a way so that an individual developer aren’t blamed for the broken build.

- The system should collect and spread the information about the build status without the need of manual user interaction. This statement is also supported by the work of S. Hamdan and S. Alramouni [8] who states that the developer should be getting feedback from the system immediately. This requires that no manual action needs to be taken.

- The information should be presented separately from other communication channels.

- The system should be user friendly and it should not require any big effort from the developers to use the system, especially since the system should be used multiple times per day.

2.2 Build automation

2.2.1 Important factors

Build duration

One important factor when it comes to build automation is what an acceptable duration of the build is and what impact it has on the development. The build duration obviously differs between different projects. However, the important part is that the build needs to be completed in time for the continuous integration server to keep up to date with the changes in
2.2. Build automation

code and report failed builds to the developers in time [6]. Jez Humble and David Farley [5] believes that a build and test process that takes more than 10 minutes is unacceptable, since the developers tend to check in less frequently or actually stop observing if the build and tests pass.

Build triggering

Another factor is when a build should be triggered. In a literature review by Daniel Ståhl and Jan Bosch [7] they report that there exist multiple statements on when a build should be triggered. Most of the references stated that a build should start as soon as there is a change in the source code while some others state that builds could be executed on a scheduled basis. An alternative is to use a mixed approach.

Broken build

In Section 2.1.4 the terms "broken build" and "build status" are mentioned. Here it is important to define what is considered a broken build. The definition of a broken build differs between teams. S. Hamdan and S. Alramouni [8] states that any kind of failure included in the continuous integration build should be considered a broken build. In a case study of a development team [12] they concluded that the team members considered a build to be broken if the code fails to compile or if any unit test fails. However, that is typically not applicable to dynamic languages such as for example Javascript, Python and Ruby simply because they aren’t typically compiled but instead ran directly. Instead unit tests are the only part of the build, as pointed out explicitly by M. Meyer [1].

According to Jez Harman and David Farley [5] it is also possible to consider a build broken on compiler warnings and violations of code style conventions. This can be considered a little bit to harsh depending on the situation, but it enforces code discipline and, after all, compiler warnings do exist for a reason.

2.2.2 Build iOS applications

This section describes how to automatically build iOS applications in a continuous integration environment.

Xcode is a collection of software development tools developed by apple and is required to build iOS applications. The standard way of building iOS applications in Xcode requires the developer to use the graphical user interface provided with Xcode. This introduces a problem when it comes to continuous integration and automated builds since the build process should not require manual user interaction. However, there exist command line tools for Xcode to build applications from the command line.

In order to actually run the iOS application for testing and beta distribution the developer needs to generate so called IPA file that can be installed on a system running iOS. The IPA simply contains the actual application. The standard way to create the IPA file from the built project requires manual steps, but there exist open source, third-party tools to generate IPA files from command line. An example of such a tool is Shenzhen.

In order to install the application on an iOS device without the use of Xcode, so called provisioning profiles are required. The profile needs to contain the unique device identifiers of the devices that is allowed to install the application. The number of allowed identifiers are 100, which needs to be taken into consideration when working in a big team or wanting to distribute the application to many people for testing or demonstration purposes. [13]
2.3 Technologies used in CoffeeMaker

2.3.1 Cocos2d-js

Cocos2d-js is a Javascript version of Cocos2d-x which contains Javascript bindings to Cocos2d-x. Cocos2d-x is an open source multi-platform framework for building graphical applications. Figure 2.2 shows the framework architecture of Cocos2d-x as shown on their Github page.

The applications created in CoffeeMaker are written in the Javascript language and compiled by Cocos2d-js to Android and iOS native platform. Therefore only a fraction of the framework representation shown in Figure 2.2 is used. As seen in Figure 2.2, the Javascript API is translating the source code to C++ code using the Javascript engine SpiderMonkey developed by Mozilla. The way CoffeeMaker uses Cocos2d-x is that it builds so called generated applications which means that the build target are platform-specific and is built in the same way as native applications [14].

2.4 Methodology

This section describes the evaluation and data gathering methodologies used in other work that is similar or could be applicable to projects of the same nature as this thesis.

2.4.1 Case study research

Case study research is a well known empirical research methodology, which purpose is to study a predefined phenomena in its context. The case study research methodology is suitable for software science studies. It could for example be used to study the effect of a change, which is often the case when applying case studies to software engineering. A case study is

---

https://github.com/cocos2d/cocos2d-js
https://github.com/cocos2d/cocos2d-x
https://developer.mozilla.org/en-US/docs/Mozilla/Projects/SpiderMonkey
https://www.mozilla.org
2.4. Methodology

flexible but still it requires planning. The objective, case, theory, research questions, methods and selection strategies must be predefined. The case could be many things, but in software engineering projects it is common that the case is a software development project. Methodologies are the ways of collecting data for the study and the selection strategy is where to seek data. [15]

Per Runeson and Martin Höst [15] describes the case study research process as follows:

1. Define objectives and plan the study
2. Prepare and define procedures and protocols for data collection
3. Collect the data
4. Analyze the collected data
5. Report the study

2.4.2 Data collection

The process of collecting data for a case study could be either qualitative or quantitative. The results from a quantitative data collection methodology are typically numbers while in qualitative data collection the results are descriptions, words, visualizations etc. There is a clear difference in how the results from these to types of methodologies are analyzed. Quantitative studies are analyzed using statistics while qualitative studies are analyzed by categorization and sorting. Qualitative data collection is the most common and often the most suitable data collection method in case studies. [15]

Timothy C. Lethbridge, Susan Elliott Sim et al. [16] describes three different techniques for collecting data which is applicable to and suitable for software engineering projects. The techniques are ordered according to the amount of human interaction required and can be summarized as follows:

- First degree: Typically interviews, questionnaires, brainstorming etc. where the researcher has control over which data will be gathered.
- Second degree: The data is gathered using for example some type of software recording equipment, which gather data about the actual usage of the tool or software that is the case of the study.
- Third degree: Data are gathered using output from using the tool or software that is the case of the study. Examples of output are source code, documentation and reports.

Interview

Interview is a first degree data collection technique. It has the advantage that the researcher can outweigh for misunderstanding of questions and ask for more information on unexpected responses. However, they have the disadvantage of being time inefficient, mostly due to the need of transcribing audio or video recordings from the interview. [16]

The interview questions are, in case studies, based on the research questions. The goal is that the answers to the questions in the interview, when analyzed, lead to answers to the research questions. The structure of the interview can be unstructured, semi-structured or fully structured. Unstructured interviews are based on questions of general interest for the researcher but allows discussions about the topics of the questions. In fully structured interviews all the questions are decided in beforehand and asked in the planned order. Semi-structured interviews are the most common types of interviews in case studies. The questions are a mix of open and closed ones. The order in which the questions are asked differs depending on the situation. Semi-structured interviews open up for improvisation and exploration of the case under study. [15]
2.4.3 Productivity evaluation

In order to decide on the improvement of using the implementation that will be provided in this thesis, it needs to be evaluated in relation to what is currently used by the company. This section lists different methods of evaluation of systems similar to what is implemented in this thesis, and therefore could be applied to what is in the scope of this project.

Required time of completing task

David Saff and Michael D. Ernst [10] performed an experiment to determine the improvement of using continuous testing when developing software. The experiments were performed on students with a mean of 2.8 years of programming experience. Even though the situation of the experiment differs from the one in this report, since this report focuses on continuous integration rather than continuous testing and the system is going to be used by more experienced programmers, they use methodologies and metrics that could be interesting to apply in both situations.

One metric they used to determine the improvement of the continuous testing was the time needed to perform a certain task when using the continuous testing in relation to when not using it. That also includes not succeeding to complete the task at all within the given time. They highlight that the difference in time is not necessarily only due to the continuous testing, but also the difference in experience of the participants.

Even though the metric of time worked in this situation, it does not necessarily mean that it could be applied in larger scale than completing a smaller task. They also highlight that the participants were provided already written tests and therefore the time needed to actually write the tests was not included in the complete time required to finish the task.

Online questionnaire

As a way to gather information about how the participants actually experienced the experiment, David Saff and Michael D. Ernst [10] used an online questionnaire to be answered by the participants. The questions were both formulated as to be answered by yes or no only, and more open ended ones. This way, they got both quantitative and qualitative answers from the answers. Examples of yes or no statements used was "I completed the assignments faster" and "I wrote better code". They do not state exactly what question or statement they used when wanting the more open ended answer but it seems like they asked for general feedback on the experiment.

A questionnaire of similar kind should be possible to apply in order to evaluate the implementation presented in this report. However, quantitative questions don’t necessarily say that much in this project if the only participants are employees at VISIARC AB since the number of employees are less than ten.
Chapter 3

Method

3.1 Architecture and implementation

This section describes the strategy used to make decisions when the system was imple-
mented. The implementation decisions was decided to be based on best practices according
to scientific theory on the subject of continuous integration systems. The specific needs of the
company were also a part of the basis for the decisions. However, the practices of continu-
ous integration and the use of continuous integration systems differ and there are not always
one equivocal answer to how the practices and implementations should be. Therefore, some
decisions were made based on literature reviews that covers how continuous integration and
continuous integration systems are most commonly used in practice.

When difficult and large implementation decisions was required, a table listing the pros
and cons of using such an implementation was created. The content was based on what was
described in the previous paragraph in this section. After the table was created the pros and
cons were discussed to make a decision.

3.2 Evaluation

This section describes how the evaluation of the project was performed. The methodology
is based on the case study research process described in Section 2.4.1. The objective of the
evaluation was to estimate the increase in productivity of the developers and the quality of
the product when using the system, in contrast to using no continuous integration system at
all.

3.2.1 Data collection technique

As a first step, a decision had to be made regarding the type of data collection method to
use for the evaluation. In Section 2.4.2, three different techniques for data collection are listed,
based on the amount of human interaction required by using the technique. The third degree
data collection technique is not suitable in the case of this evaluation. This is because the
system generates any data that could be used to measure the increase of productivity or
quality.

The second degree data collection technique would probably be the most interesting to use
as basis for the evaluation. This way it would be possible to monitor and analyze the exact
behavior of the developers using the actual system. However, the choice was to not use that
technique. That is since the effects of using a continuous integration system are long-term,
and the time range of this project is too short to perform such an experiment. Also, to set up
the tools required to monitor the activities of the developers could be too time consuming.

The choice for this evaluation was to use a first degree technique for data collection. This
way it was possible to have control over which data that was gathered.
Henceforth, a decision had to be made regarding if a qualitative or quantitative data collection methodology would be used. In Section 2.4.3 it was described that qualitative data collection methods are the most usual and often most suitable data collection method in case studies. That in combination with the fact that quality and productivity are not clearly quantifiable was the groundwork for the decision to use a qualitative data collection methodology.

### 3.2.2 Choice of method for data collection

In Section 3.2.1 a first degree qualitative data collection methodology was chosen as data collection technique for the evaluation. There are multiple choices on how to perform such a data collection. In Section 2.4.2 and 2.4.3 two different first degree qualitative data collection methods were described, interviews and online questionnaires. Based what is described in these sections, a table that lists the pros and cons of the two techniques was constructed. The result is shown in Table 3.1.

<table>
<thead>
<tr>
<th>Interview</th>
<th>Online questionnaire</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Pros</strong></td>
<td><strong>Cons</strong></td>
</tr>
<tr>
<td>Able to outweight</td>
<td>Time inefficient due</td>
</tr>
<tr>
<td>for misunderstanding</td>
<td>to transcribing</td>
</tr>
<tr>
<td>Able to complement</td>
<td>The attendants have</td>
</tr>
<tr>
<td>answers by asking</td>
<td>more time to reflect</td>
</tr>
<tr>
<td>attendant questions</td>
<td>over the question</td>
</tr>
<tr>
<td>Allows discussion</td>
<td>The attendants have</td>
</tr>
<tr>
<td>of questions</td>
<td>more time to formulate</td>
</tr>
<tr>
<td></td>
<td>a clear and well-</td>
</tr>
<tr>
<td></td>
<td>reasoned answer</td>
</tr>
</tbody>
</table>

Table 3.1: The pros and cons of using interview versus online questionnaire as data collection methodology for the evaluation.

Both the methodologies have their pros and cons. The main con of choosing to use an online questionnaire is that it is not possible to ask for complementary information. However, since the attendants in the case of this study is in the same building it is possible to ask for complementary information afterwards if needed. For the same reason, it is also possible for the attendants to get clarification if a question was not understood. The ability to discuss the subjects would get lost by using an online questionnaire. The final choice was to use an online questionnaire because of the time savings in a project with a limited time frame, even though the ability to discuss the questions was lost.

### 3.2.3 Define content of questionnaire

This section describes and motivates how the online questionnaire was formed. As motivated in Section 3.2.1 the questionnaire was decided to be qualitative and structural. This means that the questions had to be formulated in a way so that it was not possible to give quantitative answers to them. The questions in the questionnaire had to be formed in such a way that the answers, after analysis, could be used to reach the objective of the case study which is stated in the beginning of Section 3.2.

The system has before the evaluation been demonstrated to the attendants of the study. Therefore, no information about the system itself was required as a part of the questionnaire. In order to reach the objective of the study, the questions were chosen so that they ask for earlier working habits and problems when it comes to continuous integration. Also, some questions was chosen to ask for information about how these working habits and problems would probably change when introducing the system. In order to identify the time savings,
3.2. Evaluation

the questions was chosen to ask for an estimation of how much time that was used fixing the
problems without the system. This way it was possible to estimate the increased productivity
of the developers when using the system.

Since, according to Section 2.1, the main purpose of continuous integration is to keep the
software in a functional state all through the development process, one of the questions was
focused on how often the developers believe that their applications are in a non functional
state. So, the first question was:

"How often do you believe that the software (in your case the applications) in the main
line are in a non-functional state during the development process, i.e. you pull
non-functional code?"

The question above should produce answers that describes how often the main problem
that continuous integration are intended to solve occurs. However, since the objective was to
estimate the productivity improvement, information about how much time that was needed
to get the software back into a functional state was needed. Therefore, the following question
was added:

"If the case above occurs, can you roughly estimate the average time it takes to identify
when the problem occurred, why it occurred and the time it takes to get the software on
the main line back into a functional state?"

In order to get some idea about how the quality of the applications will be improved after
introducing the system, which was a part of the objective of the study, some question was
needed to address that. Since quality is not quantifiable the question was chosen to be open-
ended in order to not restrict the attendants idea of how the system could be used to improve
the quality of the software. The following question was therefore added to the questionnaire:

"You have earlier participated in a demonstration of the system, its capabilities and its
features. How do you believe that such a system could be used to improve the quality of
the applications?"

In order to find out other situations when the system could be useful the following ques-
tion was added to the questionnaire:

"You have earlier participated in a demonstration of the system, its capabilities and its
features. If you remember, describe one or more examples of situations where the system
could be useful."

Even though the system could be used for even more situations than covered by the ques-
tions, the ones above covered the most important problems that the system intend to solve.
Therefore, it was decided that no more questions had to be added to the questionnaire.

Since the questions were broad and could be considered hard to answer directly, the at-
tendants were encouraged to make assumptions, motivate and discuss the questions. The
attendants of the study were decided to maintain anonymous in the report since the names
of the attendants are not relevant. However, their names were decided to be required in the
form in case it was needed to ask for complementary information, which should be possible
according to Section 3.2.2. Complementary information, if any, was documented and added
to the report of the study. The attendants was also informed to ask questions if they believed
that any of the questions in the questionnaire was unclear. Since all the attendants of the
study spoke Swedish as first language, the questions was translated to Swedish and the at-
tendants were informed to answer in Swedish. The full questionnaire containing the same
content that was sent to the attendants can be seen in Appendix B.
Chapter 4

Result

This section describes the methodology used for the project. The first part of the section describes the design and implementation of the new continuous integration system. The second part describes how the evaluation of the newly implemented system was performed.

4.1 Architecture

As described in Section 2.1.1 how to implement and choose the architecture of a continuous integration system is ambiguous. This section describes the design choices made in the implementation in the scope of this thesis, based on the theory and the needs and limitations of the company. This section focuses on the overall architecture and information flow of the implemented system, and does not go into specific technical details.

4.1.1 Version control system

The company already use git as a version control system for all their projects. The git repositories corresponding to their projects are stored in a central internal server accessible by every developer in the company.

Based on the literature review considering build triggering in Section 2.2.1 it can be concluded that the most common opinion on when a build should be triggered is when there are changes in the source code. That was the reason why that option was used in the resulting implementation design. In order for the build to be triggered when the source code changes, the git repository needed to be configured to notify the continuous integration server when changes occurred. A convenient way to achieve that seemed to be using Git Hooks which can be used to trigger operations on certain events. In the case of this implementation it seemed convenient to trigger a request to the continuous integration server to start a build on the event of a push to the git repository.

4.1.2 Visualizing continuous integration

In Section 2.1.4 the visualization of continuous integration was described. The requirements described required the possibility to customize the behavior of the user interface and that the user interface should be separate from other communication channels.

Table 4.1 was used as basis to the decision of which communication channel to use. Three communication channels were compared. The choice of possible communication channels were based on what seemed reasonable to the time scope of this thesis, and resulted in Email (or other one way communication channels), web application and desktop application as possible candidates. In this case a desktop application should be interpreted as a stand alone, native application for the development platform used by the developers.

4.1. Architecture

The first three requirements were based on the requirements of build monitoring systems in Section 2.1.4. The last parameter in the table were based on the developer work flow shown in Figure 2.1 which indicates that it should be possible to always see the build status since the developer should not check in any new code while a build is currently running.

One way communication channels are easier to set up but based on Table 4.1 it does not meet enough requirements to be used in a good build monitoring system. Web applications and desktop applications both fulfilled all the requirements listed in the table. However, web application were chosen since it does not require installation of any additional software (since all the developers have a web browser installed already), and it is generally more time efficient to customize the user interface of a web application.

<table>
<thead>
<tr>
<th>Theoretical requirements</th>
<th>Email, SMS or other one way communication system</th>
<th>Web application</th>
<th>Desktop application</th>
</tr>
</thead>
<tbody>
<tr>
<td>Separate communication channel</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Ability to notify about broken build immediately</td>
<td>Yes, assuming low delay from email or sms provider</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Need of manual user interaction</td>
<td>No, given that notifications are enabled</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Other features</td>
<td>Request more build information</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Table 4.1: Decision basis for the choice of communication channel

4.1.3 Build monitoring system requirements

In the previous section it was decided that a web application was to be used to visualize continuous integration, however, the behavior of the application was not specified. This section aims towards listing and motivating the requirements that were used as a basis for the design of the application. The requirements should be seen as the minimal requirements of a good build monitoring system, according to the literature studied in Chapter 3.

The requirements were primarily based on the build monitoring system requirements presented in Section 2.1.4, but also on the important factors of build automation systems presented in Section 2.2.1 and the work flow and practices of continuous integration described in Section 2.1.1. The requirements are listed in Table 4.2.

Requirements nr. 2, 3 and 9 were based on the requirements presented in Section 2.1.4. The first and fourth requirements were formed to support the continuous integration work flow described in Section 2.1.1. Requirement nr. 6 was chosen as a way to ease the process of identifying what went wrong in a failed build. Requirements nr. 7 and nr. 8 were chosen based on wishes from the company.
4.1. Architecture

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Requirement description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Should show a list of builds and their status.</td>
</tr>
<tr>
<td>2</td>
<td>Should notify the developers when there is a problem with the build without the need of manual user interaction.</td>
</tr>
<tr>
<td>3</td>
<td>Should be easy to use.</td>
</tr>
<tr>
<td>4</td>
<td>Should show when a build is currently running.</td>
</tr>
<tr>
<td>5</td>
<td>Should show general information about each build.</td>
</tr>
<tr>
<td>6</td>
<td>Should show the build log.</td>
</tr>
<tr>
<td>7</td>
<td>Should be able to generate and create a default Jenkins project that works for applications created in CoffeeMaker.</td>
</tr>
<tr>
<td>8</td>
<td>Should be able to manually trigger a build with specific build parameters.</td>
</tr>
<tr>
<td>9</td>
<td>Should show the changes that lead to a new build, and the name of the developer that made the changes.</td>
</tr>
</tbody>
</table>

Table 4.2: The minimal requirements used as basis for the build monitoring application

4.1.4 Continuous integration server

As described in Section 2.1.2 it is not clear which continuous integration server to use, but they all serve the same purpose. However, the decision was to use the continuous integration server called Jenkins. First of all, Jenkins is free and open source which was a requirement from the company. Thanks to its popularity it also has many plugins that could be used to extend the servers functionality. It also provided a remote access API which among other things was useful to create the build monitoring system described in Section 4.1.3 and to communicate between different parts in the system design.

4.1.5 Continuous integration server requirements

To motivate the configuration and setup of the continuous integration server a list of requirements was formed. The requirements are based on what the company expects and what is generally considered good practices based on the theory section. The requirements are shown in Table 4.3.

4.1.6 iOS build server

The company provided a server where the continuous integration server and the web application could be hosted. The server is running a Linux based operating system. In Section 2.2.2 it is described that Xcode is required to build iOS applications, but since Xcode only exists for the OSX operating system the Linux server would not be able to build applications for iOS. Therefore, the continuous integration server needed to distribute the build of iOS applications to some other hardware that is running OSX. This could be achieved by using distributed builds which was possible by using Jenkins. By using distributed builds it was possible to build the application for Android and iOS on separate computers. The builds could also run in parallel which made the build process less time consuming.

4.1.7 Architecture overview

A summation of how the architecture of the system was finally decided is shown in Figure 4.1. A brief description of each step in the design architecture is presented below.

https://jenkins-ci.org/  
https://wiki.jenkins-ci.org/display/JENKINS/Remote+access+API  
https://wiki.jenkins-ci.org/display/JENKINS/Distributed+builds
4.2. Implementation

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Requirement description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Should be able to pass parameters to a build containing dependency versions. The company sometimes use different versions of dependencies on different versions of the application. The goal with this requirement is to be able to build older builds using earlier versions of the dependencies.</td>
</tr>
<tr>
<td>2</td>
<td>Should be able to tag a build with a version number if a version number is provided in the build parameters. The company wants to be able to build specific versions of the application and to ease that process they wish to be able to build a specific version rather than a specific commit.</td>
</tr>
<tr>
<td>3</td>
<td>Should build and package both an android and an iOS version of the application.</td>
</tr>
<tr>
<td>4</td>
<td>Should be able to rebuild an older version with the same build parameters as used when building that version the last time.</td>
</tr>
<tr>
<td>5</td>
<td>Should deploy the packaged applications to a shared directory. This requirement is for the ability to download a packaged version of the application from the server and install it to a device.</td>
</tr>
</tbody>
</table>

Table 4.3: The minimal requirements of the continuous integration server

A. The developers check in their changes to the version control system.
B. The version control server automatically notifies the continuous integration server that there has been a change in the source code.
C. The continuous integration server notifies the change and builds the application. Due to the fact that iOS applications can not be built on another operating system than OSX and the build server is running a linux based operating system, a separate build server for building the iOS application was required. The iOS build is triggered by the continuous integration server.
D. The iOS build server responds to the continuous integration server with the status of the build.
E. The continuous integration server notifies the web application on the web server about the changes in build status.
F. The web server notifies the developers about changes in the build status.

4.2 Implementation

This section describes the implementation of each part of the continuous integration system shown in Figure 4.1 in technical detail. The section provides configurations of the git repository and the continuous integration server. It also provides the design of the build monitoring system and the important contents of the build scripts and commands.

4.2.1 Version control system

Not much configuration was needed for the version control system. As described in Section 4.1.1 using git hooks is a convenient method for handling events and changes in the git repository. Hooks are placed in the folder named hooks in the git repositories root directory. The hook itself is a text file containing a script that executes the instructions of choice. The name of the executable script file decides when the script will run. The most appropriate hook to use in the case of this system is the post-receive hook which is a server-side hook. That hook
4.2. Implementation

Figure 4.1: An overview of the design of the continuous integration system implementation.

runs after the entire process is finished and is typically used to notify users or update services, which is the desired behavior in this case. [17]

A text file called post-receive was created with the following content

```bash
#!/bin/bash
curl http://<jenkins_host>:<jenkins_port>/git/notifyCommit?url=<git_url>
```

where `<jenkins_host>` is the host where Jenkins is running, `<jenkins_port>` is the port where Jenkins is accessible and `<git_url>` is the URL of the git repository and must be the same URL as in the Jenkins configuration. The call is not recognized by Jenkins itself but by a plugin called Git Plugin[^5] which configuration is described in Section 4.2.2. The `curl`[^6] command notifies Git Plugin that there has been a change to the source code.

According to requirement nr. 2 in Table 4.3 the continuous integration server should be able to tag a build with a version number. Creating tags is already a feature of git and are used to mark specific important parts in history. It is mostly used to tag releases with version points. The tags are a part of the git repository itself. [17] Therefore git tags were used in this project. However, the tagging are handled by Jenkins, so no manual tagging were needed. How Jenkins was configured for tagging is described in Section 4.2.2.

4.2.2 Continuous integration server

**Version tagging**

Tagging was implemented so that it is possible to pass in an optional build parameter containing a tag. If the parameter is set, a new tag is created in the build process and is pushed to the repository. The tags are pushed only if the build succeeds, to avoid that tagged versions are broken. The system was also implemented so that it is possible to pass in an optional build parameter containing an already existing tag in order to build an old version of the application.

[^5]: https://wiki.jenkins-ci.org/display/JENKINS/Git+Plugin
[^6]: https://curl.haxx.se/
4.2. Implementation

The ability to pass parameters to a build is provided in Jenkins by the Parameterized Build plugin, so that plugin was used in the build configuration. The parameters were chosen to be called NEW_VERSION and OLD_VERSION where NEW_VERSION is the tag to set for the current build and OLD_VERSION is the tag of the old version to build. Since both of the parameters are optional the default values of the parameters were set to be NONE.

The following shell script were written for checking out an older tag:

```bash
if [ ${OLD_VERSION} != "NONE" -a ${NEW_VERSION} == "NONE" ]
then
  git checkout tags/${OLD_VERSION}
fi
```

If OLD_VERSION was passed and NEW_VERSION was not, the old version is checked out. The command on row 4 checks out the tag [17].

In order to push the new tag to the repository, the following script was added at the end of the build process:

```bash
if [ $\{NEW_VERSION\} != "NONE" ];
then
  git tag -a $\{NEW_VERSION\} -m "Tagged new version"
  git push origin --tags
fi
```

In short, the script tags the repository with NEW_VERSION if the value of the parameter is not the default one. The git command on row 2 tags the current checked out version with the new tag [17]. The script pushes the tags to the repository on row 3, if a new tag was chosen to be set. The script is executed at the end of the build process to only push tags of successful builds.

Version branching

Tags are good for marking up specific important points in history, such as version numbers. However, tags are nothing more than pointers to specific points in version control history.

In order to make changes to earlier versions of the application, such as minor updates or bug fixes, a new branch needs to be checked out at the specified tag [17]. The changes are then committed on the newly checked out branch. There exist multiple branching release models for version control with git, and the opinions on which are the best differs.

Vincent Driessen [18] posted a well cited blog post where he advocates a git branching model that he claims to be successful. In his git branching model there are two main branches in the central repository, the master branch and the development branch. The master branch contains all the production releases marked by tags, and should always be in a production-ready state. The development branch is always in a state containing the latest development changes that is going to be a part of the next production release. When creating a new release, the development branch is merged into the master branch and then tagged with an appropriate version number. Beyond these branches there are feature branches, that branch of the development branch. The feature branches are created when working on a new feature, and are merged back into the development branch and removed when the feature is ready. There are also the release branch that is used for preparation of a new production release. It branches of the development branch and are merged into the development and master branch when it is ready for release. Finally, there is a hotfix branch containing small and urgent fixes to a production release. It is branched from the master branch and merged back into the master branch, where it is tagged as a subversion, and into the development branch.

However, continuous integration are not taken into consideration in the description of the model. As described in Section 2.1 continuous integration is a practice where new code

[17] https://wiki.jenkins-ci.org/display/JENKINS/Parameterized+Build
is integrated with the main line several times per day. The model promotes creation of new branches for new features and hotfixes, rather than frequent integration with the main line which could conflict with the purpose of continuous integration.

The model is also criticized in another blog post by Jussi Judin [19] who also believes that the model conflicts with continuous integration. Instead, he promotes a different and more simple branching model. In that model there is a master branch on the shared repository, where all development changes are pushed into. A new main release is created by branching from master into a release branch and is tagged with a version number. Then it is possible for the developers to check out the release branch when bug fixes, hotfixes or small features needs to be added to the release. The changes made on the release branch can also be merged into the development branch.

The continuous integration server implementation in this report was decided to support a branching and release model similar to the one described by Jussi Judin [19]. The reason is that it is more convenient to use with continuous integration and that the model is simpler for the developers to adapt to. An example of a the branching model that the continuous integration system implementation is adapted for is shown in Figure 4.2.

There are no consensus on how software versions should be numbered. In the example in Figure 4.2 it is assumed that the version number is in the format $X.Y.Z$ where $X$ is increased at a big new release, $Y$ is increased at smaller feature releases and $Z$ is increased when there are bugfixes or hotfixes. A new branch is created when there is a new release. Bug fixes can be performed on these branches and are merged back into the master branch if the fix are supposed to be a part of later versions.

In Figure 4.2 the branch names are in brackets and each dot in the tree is a commit. The commits are ordered in time from top to bottom.

In Jenkins, the following shell script was added that creates a new branch when the build is configured for release

```bash
if [ $(NEW_VERSION) != "NONE" ]; then
    git push origin ${GIT_BRANCH}:release-${NEW_VERSION}
fi
```

where $GIT_BRANCH$ is an environment variable containing the current branch provided by the Jenkins git plugin. The new branch name reflects the new version number.

### Manage multiple branches

As an effect of the branching model described in Section 4.2.2 the git repository will contain multiple branches. This means that Jenkins will need to be configured to manage multiple branches. Section 4.1.1 described the Git plugin available for Jenkins and how it can be configured to automatically rebuild the project when there is a change in the source code. The configuration of the Git plugin also makes it possible to define which branches to build, by providing a regular expression matching the names of the branches. Since the branching model has one branch called master and multiple branches called release-x the regular expression used was $(master | release-*)$ which matches the branch names. Then, when Jenkins is notified that there are changes in the source code, it will automatically rebuild the changed branches.

### Upload packaged version

In Section 4.2.4 it is described how the web application can receive post requests containing the packaged versions of the applications, store them and serve them statically so that it is possible to download the packaged version from the graphical interface of the application. Therefore, Jenkins was configured to upload the packaged version of the application to the web application. This was done by using the following command:
4.2. Implementation

Figure 4.2: An example of the branching model that the continuous integration system implementation is adapted for.

```
curl --form "apk=<path\_to\_package>" http://<web\_app\_host>:<web\_app\_port>/\{JOB\_NAME\}/\{GIT\_BRANCH\}/upload
```

where JOB_NAME and GIT_BRANCH are environment variables provided by Jenkins and are the name of the project and the currently built branch. <path_to_package> is the path to the built apk or ipa file. This command was configured to be executed as the last step in the build process. How the web application handles the request and serves the application files are described in Section 4.2.4.

Store parameters

Section 4.2.2 described how versions are tagged. In order to build an old version of the application in the same manner as when it was tagged for release, the versions of the dependencies needed to be stored. Using more recent versions of the dependencies to build an old version of the application could lead to a broken build. Therefore, when NEW_VERSION is set, the build system was configured to store the used versions of the dependencies into a text file. The text file is then checked into version control and pushed to the remote repository. This means, that each release build contains the versions of the dependencies.
4.2. Implementation

When OLD_VERSION is set, the build system was configured to load the versions of the dependencies from the text file, and use these versions during the build.

The dependencies needed for standard applications developed in CoffeeMaker are Cocos2d-js, Node-js, and the CoffeeMaker framework itself. CoffeeMaker and Cocos2d-js are git repositories located at the companies own server. Therefore, these dependencies were configured to be cloned by the build system into the Jenkins workspace of the project.

If NEW_VERSION is set, the dependency versions are stored by using the following script:

```bash
1 cd ${WORKSPACE}/<CoffeeMaker>
2 export VS=$(git rev-parse HEAD)
3 cd ${WORKSPACE}/<Cocos2d-js>
4 export COCOS=$(git rev-parse HEAD)
5 cd ${WORKSPACE}
6 echo "COFFEEMAKER_COMMIT=${VS}
COCOS2D_COMMIT=${COCOS}
NODE_VERSION=${NODE_VERSION}"
```

where <CoffeeMaker> and <Cocos2d-js> are the names of the directories containing CoffeeMaker and Cocos2d-js respectively. Row 2 and 4 gets the hash values of the currently checked out commits of CoffeeMaker and Cocos2d-js, and stores them in temporary variables. NODE_VERSION is the used version of Node-js and is configured as a build parameter and is therefore accessible at this point in the build script. In row 6 the parameters are stored in a text file in a format that is supported by the EnvInject Plugin for Jenkins, which is later on used to inject these parameters to the build process.

If OLD_VERSION is set, the build system injects the environment variables used by that version using theEnvInject Plugin, and checks out the specified version of these dependencies using git. How this was implemented is shown in the script below:

```bash
1 cd ${WORKSPACE}/<CoffeeMaker>/
2 git reset --hard ${COFFEEMAKER_COMMIT}
3 cd ${WORKSPACE}/<Cocos2d-js>/
4 git reset --hard ${COCOS2D_COMMIT}
5 cd ${WORKSPACE}
6 nvm install ${NODE_VERSION}
```

COFFEEMAKER_COMMIT and COCOS2D_COMMIT are the hash values of the commits loaded from the text file containing the dependencies versions. Row 2 and 4 in the script resets the git repositories to the specified versions. In order to use a specific version of Node-js the build system was configured to use the open source project Node Version Manager which is able to install and switch between different versions of Node-js. Before using node the system was configured to run

```bash
1 nvm install ${NODE_VERSION}
```

which installs the version NODE_VERSION of Node-js if it is not already installed, and switches to use that version.

Running unit tests

In Section 2.1.3 it was described that unit testing increase software quality. Therefore, the continuous integration server was configured to execute unit tests and publish the test results as a part of the build process.

Since the CoffeeMaker applications are mainly written in the Javascript language, the unit testing framework of choice needed to support Javascript. The company mainly uses the Mocha Javascript unit testing framework for testing the applications, and therefore Jenkins was configured to execute unit tests written for Mocha and publish the results. The script used to run the unit tests is shown below.

https://nodejs.org/en/
https://wiki.jenkins-ci.org/display/JENKINS/EnvInject+Plugin
https://github.com/creationix/nvm
https://mochajs.org/
4.2. Implementation

```bash
test/ > test_reports.xml
```

The script executes Mocha on all the tests located in the directory called `test`. The flag `--recursive` makes sure that all tests in the directory and all its subdirectories are executed. This way the developers can put all the tests they write in that directory and all the tests will automatically be executed by the server.

Based on the Mocha documentation a typical `mocha` tests looks as follows

```javascript
describe(<name_of_unit>, function() {
  it(<description>, function() {
    <assertions>
  });
});
```

where `<name_of_unit>` is the name of the unit under test and `<description>` is a description of the expected behavior. `<assertions>` are statements that are expected to be equal to true at that point in execution.

The command line option `-R xunit` in the `mocha` execution decides that the xunit test reporter format will be used when generating the test report. The report contains information about the results of the test execution. Jenkins provides an option for publishing test reports of the xunit format, called `Publish JUnit test result report`. The option takes the xunit report file as input and generates test results that can be accessed from the Jenkins user interface or the Jenkins api.

By default, if one or more test fails, Jenkins will mark the build as unstable. This way, it is easier to see if a build is broken or if a unit test failed. Section 4.2.4 describes how the test results are visualized in the web application.

4.2.3 iOS build server

As described in Section 4.1.6 the iOS applications needed to be built on a separate computer running OSX. The same section described that the Jenkins feature of distributed builds could be used to solve the problem. Distributed builds consist of a master node and one or more follower nodes. The master node is scheduling jobs, dispatching builds to follower nodes and presents the result from the build. The follower nodes needs to have Jenkins installed and must be connected to the master. It is possible to associate a job with a specific follower node in the job configuration in Jenkins, which means that the job will be executed on that follower node only. [20]

This configuration was used in the project implementation. The master was the Linux computer and the follower node was the computer running OSX. Since, according to the Jenkins guide, it is only possible to execute separate jobs on a follower node and not specific build steps, a separate job was needed to be created for the iOS build. The main job was then configured to start the iOS build job by using the Parametrized Trigger Plugin [12]. The plugin starts the job in parallel and reports the result as if it was one single job.

Setting up Xcode for continuous integration

An iOS project in Xcode contains something called Xcode schemes [13] which contains targets to build but also build and testing configurations. In order to automate integration, the scheme needs to be available in the source code repository. This was achieved by sharing the scheme in Xcode as described in the iOS developer library [14] and committing the shared scheme to the source code repository.

[12] https://wiki.jenkins-ci.org/display/JENKINS/Parameterized+Trigger+Plugin
Figure 4.3: A simplified view of the build process configuration for Jenkins.
Configuring the build process

In Section 2.2.2 it is described how IPA files are required for testing and beta distribution, and how manual steps are required to generate those. It was also described that Shenzen is a tool to automatically perform the manual steps. However, there exists a plugin for Jenkins called Xcode Plugin which automatically calls the manual Xcode build commands based on provided parameters. When the plugin was installed, an Xcode build step was added in the build configuration. The build step required filling in information such as the name of the schema file created in Section 4.2.3 and project and build directories.

The Xcode plugin provides an option to pack the application and create an IPA file, so that option was used. In order to create the IPA file it was needed to provide a team provisioning profile. As described in Section 2.2.2 team provisioning profiles are required for applications to launch on devices. According to the iOS developer library provisioning profiles contain the application ID, development certificates that are used to identify the developers and a list of devices that are allowed to run the application.

Therefore, in order to make Jenkins able to create IPA files that are able to launch on devices, a development certificate was created to be used by Jenkins. The developer profile was configured to be connected to the apple id of the user running the iOS build process. The certificate was added to the team provisioning profile of the company. Provisioning profiles can be generated or downloaded at any time and can expire. Therefore, the provisioning profile was not added to version control. Instead, a third party and open source tool called Sigh was used. Sigh is able to download provisioning profiles. A shell script command build step was added that downloads all the provisioning profiles for the apple ID of the Jenkins user and looks as follows:

```bash
sigh download_all -u <apple_id> -o certificates
```

where `<apple_id>` is a valid apple developer user that has access to the provisioning profile. In the configuration of the Xcode plugin a path to the profile was added as `certificates/<certificate_name>.mobileprovision` where `<certificate_name>` is the name of the provisioning profile to use for signing the IPA file.

4.2.4 Web application

The implementation decisions that was made when developing the web application were based on the requirements listed and motivated in Table 4.2.

Graphical interface

The starting page of the application shows a list of all the projects in Jenkins. When choosing a project in the list the application redirects the user to the build monitoring view of the project. The build monitoring view shows the most relevant information about the projects status. The components that are shown in the build monitoring view are:

- The name of the project.
- Buttons to download archived and packaged versions of the last successful build of the application, for android and iOS respectively.
- A button to trigger a new build with the default parameters.

---

1. [https://wiki.jenkins-ci.org/display/JENKINS/Xcode+Plugin](https://wiki.jenkins-ci.org/display/JENKINS/Xcode+Plugin)

25
4.2. Implementation

- A button to trigger a new build with custom parameters. The custom parameters are filled into a web form before triggering the build.

- A list of all the builds that are color coded by the success of the build. Green for success, red for failure, gray for aborted and yellow for currently running. Each item in the list is expandable and contains the status, the time, the duration and a link to the build log of the build. A screenshot of the build list is shown in Figure A.1 in Appendix A.

- Graphs and statistics that summarizes the status and history of the project’s builds.

- A list of the changes in the source code compared to the most previous build. This is simply the commit messages for the commits added since the last build.

- The name of the developer that made the changes.

- A list of the failed tests, if any, and the description of each test.

There are two graphs. One of the graphs shows the build duration on the vertical axis and the build number on the horizontal axis and the other graph shows the trend of successful unit tests.

As stated by the requirement number 2, the application should notify the developer when there is a problem with the build. When in the build monitoring view, the application is implemented so that when a new build fails, a web browser notification is sent stating that the build failed, without the need of manual user interaction. When a new build starts, the build is added to the build list with the currently running status, also without any need of manual user interaction. From the starting page it is possible to create a new default project by filling in the name of the git repository corresponding to the project.

Backend

The implemented backend serves the static files that forms the graphical interface. It also redirects requests from the user to the continuous integration server and responds with appropriately formatted data. In addition, a node module was implemented that creates a default CoffeeMaker project from a project name.

Technologies used

The frameworks and technologies used to create the graphical interface and client side scripting are not mandatory since the important part is what is shown on the interface. In this project AngularJS\(^{18}\) was used as frontend framework in combination with the standard graphical components provided by UI-Bootstrap\(^{19}\). Socket.io\(^{20}\) was used to send information to the client without the client explicitly requesting it.

The backend was written in Node-js using the Express\(^{21}\) web application framework, which made it possible to use the package manager npm\(^{22}\) that is provided by Node-js.

Jenkins provides a REST-like remote access API\(^{23}\). The API made it possible to request information about Jenkins jobs and builds, trigger builds and create new jobs. The API is provided for different languages and data interchange formats, but the JSON format was chosen for this project since it is based on a subset of the Javascript language which is the language the application is written in\(^{24}\).

\(^{18}\)https://angularjs.org/

\(^{19}\)https://angular-ui.github.io/bootstrap/

\(^{20}\)http://socket.io/

\(^{21}\)http://expressjs.com/

\(^{22}\)https://www.npmjs.com/

\(^{23}\)https://wiki.jenkins-ci.org/display/JENKINS/Remote+access+API

\(^{24}\)http://www.json.org/
In order to decrease the amount of code needed to make calls to the Jenkins API from the backend the npm module called jenkins-api\(^{25}\) was used to make Jenkins API requests. The jenkins-api module includes a function for creating a Jenkins job from a configuration file. The configuration file is in the XML format. An npm module called xmlbuilder\(^{26}\) which is able to generate XML formatted data from a JSON object. The xmlbuilder was used to create a module that generates a configuration for a default Jenkins job in XML format. The XML data was then used to create the new Jenkins job using the jenkins-api module.

### 4.3 Evaluation

This section presents the results from the questionnaire. The full answers exactly as answered by the attendants can be seen in Appendix\(^{27}\).

On the first question, the answers were a little different. The answers varied from "Up to a few times per month" to "Approximately one time per week" to "Approximately 20% of the times I pull new code", which then depends on how often new code are pulled.

On the second question, the answers varied more. The first attendant answered that normally problems with a git pull are not very time consuming to fix, but what takes time is to find out that something in the repository does not work. The second attendant does not believe that it is very time consuming, approximately 15 minutes per pull, but more of a vexation. The last attendant believes that it takes 5-15 minutes each time, but it can take up to 30-60 minutes on big changes.

Question 3 and 4 considered the quality improvement aspect of the system. All the attendants highlighted that the fact that the system is able to handle releases of applications will increase both the quality of the applications and the productivity of the developers. One of the developers highlight that the ability to patch or rebuild an older release will also lead to increased quality. One of the developers also believe that the fact that tests can be ran centralized will increase the quality of the software. Currently each developer tested the part of the application they are currently working on but centralized testing will give a better insight of the application as a whole.

Two of the developers highlighted the fact that the centralized build will build for both the Android and iOS target, which means that the probability that platform specific errors will be detected is higher. One of the developers also described the value of that the applications are always built automatically and by using the same process, which decreases the risk of build failures and the detection of mistakes that occurred while checking in changes.
Chapter 5

Discussion

5.1 Result

5.1.1 Evaluation

The goal of the evaluation was to estimate the improvement in quality of the applications and the productivity of the developers when using the system compared to not using the system at all. In this section the results of the evaluation, shown in Section 4.3, are analyzed in order to make such an estimation.

Productivity improvement

It is hard to make a good estimation of the productivity improvement, so the estimation is very approximate. Based on the answers from the questionnaire it can be estimated that the developers pull broken code approximately one time every week. When broken code are pulled, one of the developers believe that it takes approximately 15 minutes and another one that it mostly takes 5-15 minutes each time, but sometimes up to an hour. One developer said, however, that it is more time consuming to actually conclude that the error is in the repository. From this information one can estimate that the developers save 15-30 minutes per week by using the system when it comes to pulling broken code.

The developers also described other situations when they believe that the system may increase the productivity, but those answers are less quantifiable. All the developers thought that the release process is time consuming, and will probably be easier with the release system in the continuous integration system. The fact that the system can detect possible platform specific errors will make the developers detect these errors in an earlier stage, which should also improve the productivity. Therefore, I assume a productivity improvement that is more than 15-30 minutes per developer. It is also probable that the system can be used to increase productivity in situations that the developers did not thought of when answering the questionnaire. Therefore I roughly estimate, on average, a productivity improvement of 30-60 minutes per week and developer when using the system compared to not using the system at all. Since there are currently seven developers at the company that means a total productivity improvement of approximately 3.5-7 hours per week.

5.1.2 Quality improvement

The estimated quality improvement is not possible to quantify, since there is no measure for quality. However, the developers described certain situations when they believe that the system could be used to increase the quality and stability of their delivered applications. All the developers believed that the fact that releases are always built in the same way will decrease the risk of releasing faulty applications to the customer, which is a quality improvement. They also believed that centralized testing will lead to higher quality and that all the devel-
opers have better insight in all parts of the project, which will lead to an increase in stability. Overall, the fact that the developers believe that errors are detected earlier will reduce the risk of delivering applications that does not work as expected. These are aspects that will probably lead to increased quality, and even though it is hard to specifically say how big the improvement will be, it is certain that the quality actually will be increased.

5.2 Methodology

5.2.1 Margin of error in the evaluation and implementation

This section criticizes the methodology used in this project. Since the evaluation was performed by using a semi-structured qualitative method it is, as stated in the report, only an estimation of the actual time and quality improvements that comes with using the system. The evaluation method could also have been affected by the fact that the participants have had insight into the development progress of the system. Also, even though the interpretation of the interview responses is described in detail, the derived estimation of time savings and quality improvements could be interpreted in multiple ways.

The system implementation was mainly based on best practices and how continuous integration systems are commonly implemented. By using such a methodology the resulting implementation are not necessarily the best one. Even though a solution is common it is not necessarily the best. In order to be able to validate how good the system actually is compared to other implementations it would be needed to somehow get quantified output from different system implementations and compare them. However, in the case of productivity and quality improvements the results of introducing a continuous integration system is hard or impossible to quantify. It would also be possible to make extensive qualitative evaluations of different system implementations and qualitatively analyze and compare them.

5.3 Replicability and validity

The decisions made in the implementations have been based on theory, but where discussed and motivated by the author, which means that the outcome of the decision could have been different if made by another author interpreting the theory differently. The decisions were also based on the situation and the needs of the company which means that the outcome should be different in another case. However, that is often the case in a case study. I believe that the methodology is replicable in the sense that it should be possible to follow the same process in a similar case. The validity of the evaluation is hard to determine. The evaluation was performed as an estimation of the improvement when using the system implementation based on the expected use of it. I believe that the time and quality improvements can be roughly estimated, which was also the goal of the study, by using an evaluation technique as the one in this study. However, the actual improvement may differ significantly.

5.4 References

This section criticizes the references chosen as a basis for the theory in this study. On the subject of continuous integration there were a lot of scientific papers to use as a basis for the study. However, there were less scientific papers available to use as a basis for the actual implementation. In the implementation a lot of tools was used, which it was not possible to find relevant scientific sources to verify. Instead, I chose, to a great extent, to use the official documentation for the tools as reference.
5.5 The work in a wider concept

In Section 2.1.4 it was described that the build monitoring system should show who it was that broke a build, and that the system should present the information in a way so that whoever caused the broken build aren’t blamed for the broken build. The build monitoring system in itself could reduce the risk of that happening, but it also depends on company cultural aspects. The company should make sure that all the developers are informed that breaking a build is not necessarily that bad and that they should not feel blamed themselves or blame other developers. If not, such a system could cause more harm than benefit. To inform the developers about these issues is therefore a recommendation before introducing such a system.
Chapter 6

Conclusion

The study lead to the conclusion that the build process required to build applications in VISIARCs framework for cross-platform mobile application development can be automated by scripting certain tasks as described in Section 4.2. The continuous integration and delivery system implementation presented in this report is one working way to realize such an implementation based on theory and requirements from the company. This report shows how such an implementation is possible by using Jenkins and other open source tools.

The evaluation of the system lead to the conclusion that the automated build process and the future use of the continuous integration system will, by estimate, increase the productivity by 30-60 minutes per week and developer. The evaluation also lead to the conclusion that the quality will most probably increase when using the system.

6.1 Future work

As described in Section 5.2 the time and quality improvements derived from the questionnaire are not unequivocal. Therefore it would be interesting to observe the actual time and quality improvements. Since the effects of introducing a continuous integration system is long-term, there was not enough time to perform such an evaluation. It could also be interesting to see the difference of the questionnaire responses after that the respondents have been using the system for a while, to see if the system met the expectations and estimations. A similar approach was performed by John Downs et al. [11] to evaluate an implemented build monitoring system. They evaluated the system by letting the team use the system and evaluate it by using qualitative interviews and observing the changes in behavior. This type of evaluation was not possible to perform due to the time limit of this project. Although, a similar evaluation method could be interesting to perform as future work.

Another evaluation that could be interesting to perform is to evaluate different implementations of continuous integration systems. As described earlier in the report, there are no consensus on how a good continuous integration system should be implemented and configured, which makes it hard to make decisions when implementing one. A comparison of multiple common implementations considering productivity and quality improvements should therefore be useful for future similar projects as the one in this report.
Appendix A

Screenshots of the web application

Figure A.1: A screenshot of the build list in the build monitoring view of the web application.
### Overview

#### All projects

<table>
<thead>
<tr>
<th>Name</th>
<th>Last built</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dispose.js</td>
<td>2 hours ago</td>
</tr>
<tr>
<td>AwesomeApp</td>
<td>2 hours ago</td>
</tr>
<tr>
<td>Förs</td>
<td>3 days ago</td>
</tr>
<tr>
<td>Solidart</td>
<td>31 days ago</td>
</tr>
<tr>
<td>AwesomeApp-ios</td>
<td>38 days ago</td>
</tr>
<tr>
<td>Solidart-ios</td>
<td>2 months ago</td>
</tr>
</tbody>
</table>

Figure A.2: A screenshot of the starting page of the web application.
Figure A.3: A screenshot of the build trend graph in the build monitoring view of the web application. The horizontal axis shows the build number and the vertical axis shows the duration of the build in minutes.

Figure A.4: A screenshot of the test trend graph in the build monitoring view of the web application.
Appendix B

Questionnaire

This appendix shows the content of the questionnaire as sent out to the developers. The questions are marked with QX where X is the question number. The first question where the attendant is asked for their name is not numbered since the answer is not present in this report.

B.1 Description

Var vänlig läs instruktionerna noggrant!


För att öppna upp för möjligheten att fråga om kompletterande information på dina svar, kommer du behöva uppgå ditt namn. Du kommer däremot vara anonym i rapporten till den mån att ditt namn inte visas. Det enda som publiceras är att du är anställd på företaget samt dina svar.

Fråga gärna om du har några funderingar kring frågorna i enkäten.

Tack på förhand!

B.2 Questions

Vad heter du?

Q1: Ungefär hur ofta under utvecklingen skulle du uppskatta att mjukvaran (i ert fall apparna) är i ett icke fungerande tillstånd, d.v.s. att du tar emot ej fungerande kod vid en "git pull"? Det uppskattas om du motiverar ditt svar, diskuterar frågan och/eller ger exempel relaterade till frågan.

Q2: I fallet ovan, ungefär hur lång tid skulle du uppskatta att det tar att identifiera när problemet uppstod, varför det uppstod respektive hur lång tid det tar att modifiera koden så att programmet på master-branchen återgår till ett fungerande state? Det uppskattas om du motiverar ditt svar, diskuterar frågan och/eller ger exempel relaterade till frågan.
B.3 Answers


Q4: Tidigare under examensarbetet har du deltagit vid en demonstration av byggsystemet och dess funktionalitet. Om du minns, beskriv ett eller flera exempel på situationer i ditt arbete där ett sådant system skulle vara användbart.

Attendant 1:

Q1: Upp till ett fåtal gånger per månad. Den vanligaste orsaken är att någon glömt att checka in en nödvändig fil, eller att ny funktionalitet inte blivit testad på både Android och iOS.

Q2: Det går oftast ganska fort att rätta till. I fallet att en fil glömts går det väldigt fort att rätta till. Om tex projektfilen på iOS inte har blivit uppdaterad är det också oftast enkelt att fixa. Det som brukar ta mest tid är själva konstaterandet att det är något från repositoriet som inte fungerar och inte något jag själv gjort.

Q3: Det kommer att underlättas mycket vid releaser, både vid själva byggandet av releasen då vi kan vara säkra på att det är exakt det som är i vårt repository som byggs, samt om vi vill gå tillbaka och testa eller patcha en tidigare release. Förhoppningsvis kommer byggsystemet även att fånga en del plattformsspecifika problem, som att en utvecklare pushat in kod som fungerar bra på en plattform men som inte bygger på en annan. Detta borde leda till att vi blir bättre på att testa innan införning.

Q4: Vi gör kontinuerliga releaser av ett stort konsultprojekt. Det har hänt att felaktiga inställningar har kommit med i releasebyggar, något som vi skulle ha sluppit med ett bra byggsystem. I andra projekt har vi haft problem med att gå tillbaka och bygga en tidigare släppt version då vi inte tillräckligt bra dokumenterat hur uppsättningen vid releasebyggen såg ut. Även detta skulle kunna ha lösts med ett bra byggsystem.

Attendant 2:

Q1: Jag uppskattar att det händer ca 20% av gångerna jag gör en git pull.


B.3. Answers


Attendant 3:

Q1: Någon gång per vecka. Detta är olika beroende på vad jag jobbar med. Har man inte jobbat i ett projekt ett tag är sannolikheten större att förändringar skett som ger en icke-fungerande bygg.

Q2: Oftast tar det mellan 5-15 minuter men kan mera sällan ta 30-60 minuter vid större förändringar.


Bibliography

[1] Mathias Meyer. “Continuous Integration and Its Tools”. In: *IEEE Software* 31.3 (2014), pp. 14–16. ISSN: 0740-7459. DOI: 10.1109/MS.2014.58 URL: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6802994&disnumber=6802981&backslash=7B%5C&sortType=7B%7Dasc&7B%7DAND%7B%7D28p%7B%5C_%7D15%7B%5C_%7DNumber%7B%5C_%7D3A6802981%7B%5C_%7D29


[4] W W Royce. “Managing the development of large software systems”. In: *Electronics* 26.August (1970), pp. 1–9. ISSN: 03784754. DOI: 10.1016/0378-4754(91)90107-E URL: http://www.pl.informatik.tu-darmstadt.de/fileadmin/user%5C_7B%7C%7DUpload/Group%7B%5C%7DPI/LV%7B%5C%7D7B%5C%7DSE%7B%5C_%7D%20folien%7B%5C%7D%20Wasserfallmodell%7B%5C%7D%20Schwaiger.pdf


