Teaching Practical Software Maintenance: Students' Impression of Learning in a Software Development Course

Daníel B. Sigurgeirsson

Thesis of 60 ECTS credits
Master of Science (M.Sc.) in Computer Science

May 2017

Teaching Practical Software Maintenance: Students' Impression of Learning in a Software Development Course

by

Daníel B. Sigurgeirsson

Thesis submitted to the School of Computer Science
at Reykjavík University in partial fulfillment
of the requirements for the degree of
Master of Science (M.Sc.) in Computer Science

May 2017

Thesis Committee:

Björn Þór Jónsson, Supervisor
Associate Professor, Reykjavík University, Iceland

Marta Kristín Lárusdóttir, Co-supervisor
Associate Professor, Reykjavík University, Iceland

Mats Daniels, Committee Member
Senior Lecturer, Uppsala University, Sweden

Mohammad Hamdaqa, Committee Member
Assistant Professor, Reykjavík University, Iceland

Teaching Practical Software Maintenance: Students' Impression of Learning in a Software Development Course

Daníel B. Sigurgeirsson

May 2017

Abstract

Software maintenance typically represents 60% of software development effort, yet is mostly ignored in computer science education. In this case study, we examine the students' impression of what they learn when participating in a software maintenance course at Reykjavik University which focuses on a large scale software development project. The background of the course is introduced, how the course is set up and what its learning outcomes are. We summarize how we measure the students impressions. The results from those measurements are discussed, and used to suggest improvements to the project and the course.

Kennsla viðhalds hugbúnaðar: sýn nemenda á nám í hugbúnaðargerðaráfanga

Daníel B. Sigurgeirsson

maí 2017

Útdráttur

Viðhald hugbúnaðar fær litla athygli við kennslu í tölvunarfræði, þrátt fyrir að standa fyrir 60% af kostnaði og umstangi á líftíma hugbúnaðarins. Í þessari grein skoðum við upplifun nemenda af þátttöku í hugbúnaðarverkefni í áfanga í Háskólanum í Reykjavík þar sem nemendur vinna saman að smíði og viðhaldi á stóru hugbúnaðarverkefni. Bakgrunnur áfangans er kynntur, hvernig hann er uppsettur og hver lærdómsviðmið áfangans eru. Við skoðum hvernig upplifun nemenda er mæld, ræðum niðurstöður mælinganna, og notum þær til að leggja til breytingar á áfanganum og verkefninu.

Teaching Practical Software Maintenance: Students' Impression of Learning in a Software Development Course

Daníel B. Sigurgeirsson

Thesis of 60 ECTS credits submitted to the School of Computer Science
at Reykjavík University in partial fulfillment of
the requirements for the degree of
Master of Science (M.Sc.) in Computer Science

May 2017

Student:

.....
Daníel B. Sigurgeirsson

Thesis Committee:

.....
Björn Þór Jónsson

.....
Marta Kristín Lárusdóttir

.....
Mats Daniels

.....
Mohammad Hamdaqa

.....
Date

.....
Daníel B. Sigurgeirsson
Master of Science

Dedicated to my family.

Acknowledgements

I would like to thank my instructors, Björn Þór and Marta, it has been a priviledge to enjoy their guidance. I would also like to thank the students which took part in this course and this project, without them this would not have been possible at all.

- Alexander Baldvin Sigurðsson
- Andri Rafn Ágústsson
- Ari Freyr Ásgeirsson
- Arnar Dóri Ásgeirsson
- Arnar Freyr Sævarsson
- Arnar Gauti Ingason
- Arnar Þór Sveinsson
- Árni Fannar Þráinsson
- Árni Freyr Þorsteinsson
- Árni Reynir Óskarsson
- Arnór Ýmir Guðjónsson
- Arnþór Snær Sævarsson
- Aron Bachmann Árnason
- Ásgeir Bjarni Ingvarsson
- Ásgeir Daði Þórisson
- Áslaug Sóllilja Gísladóttir
- Ásrún Sigurjónsdóttir
- Atli Guðlaugsson
- Axel Máni Gíslason
- Axel Örn Sigurðsson
- Bæring Gunnar Steinþórsson
- Baldur Tryggvason

- Barði Freyr Þorsteinsson
- Benedikt Hólm Þórðarson
- Benedikt Logi L Sörensen
- Benedikt Þorgilsson
- Berglind Lilja Björnsdóttir
- Bergþór Þrastarson
- Bjarki Sörens Madsen
- Bjarni Egill Ögmundsson
- Bjarni Ragnar Guðmundsson
- Bjarnþór Sigurðarson
- Björn Alfreðsson
- Björn Ólafur Jóhannsson
- Brynjar Bragason
- Brynjar Smári Bragason
- Brynjólfur J Hermannsson
- Brynjólfur Stefánsson
- Carsten Petersen
- Dagur Arinbjörn Daníelsson
- Davíð Hafþór Kristinsson
- Dorothea Pálsdóttir
- Drífa Örvarsdóttir
- Egill Sveinbjörnsson
- Einar Alexander Eymundsson
- Einar Karl Einarsson
- Einar Logi Hreinsson
- Einar Sigurður Sigurðsson
- Einar Þór Traustason
- Elín Rós Hauksdóttir
- Emil Atli Ellegaard
- Erla Harðardóttir

- Fannar Már Sigurðsson
- Fannar Örn Hermannsson
- Fanney Hrund Jónasdóttir
- Fanney Sigurðardóttir
- Finnbjörn Þorvaldsson
- Friðrik Már Jónsson
- Gadidjah Margrét Ögmundsdóttir
- Geir Matti Jarvela
- Grímur Daníelsson
- Guðjón Geir Jónsson
- Guðjón Hólm Sigurðsson
- Guðjón Jónsson
- Guðlaugur Garðar Eyþórsson
- Guðmundur Ellert Björnsson
- Guðríður Sturludóttir
- Guðrún Sif Hilmarsdóttir
- Gunnar Karl Pálmason
- Gunnar Marteinsson
- Gunnar Þór Stefánsson
- Gunnhildur Finnsdóttir
- Hafrún Sigurðardóttir
- Hafsteinn Hjartarson
- Hafþór Gunnlaugsson
- Hafþór Snær Þórsson
- Halldór Vilhjálmsson
- Haraldur Andri Stefánsson
- Haukur Stefánsson
- Heiðar Berg Hallgrímsson
- Heiðar Freyr Steinunnarson
- Heimir Már Helgason

- Helgi Rúnar Einarsson
- Hildur Andrjesdóttir
- Hjalti Erdmann Sveinsson
- Hólmfríður Guðlaug Einarsdóttir
- Höskuldur Ágústsson
- Hrafn Orri Hrafnkelsson
- Hrafnkell Baldursson
- Hreinn Rúnarsson
- Hugrún Ósk Bjarnadóttir
- Hulda Lárusdóttir
- Ingólfur Rúnar Jónsson
- Ívar Oddsson
- Jakob Arinbjarnar Þórðarson
- Jóhann Brynjar Magnússon
- Jóhanna María Svövudóttir
- Jón Agnar Stefánsson
- Jón Atli Jónsson
- Jón Hallur Haraldsson
- Jón Mogensson Schow
- Jón Örn Arnarson
- Kári Mímisson
- Karl Einarsson
- Karl Kristjánsson
- Kevin Freyr Leósson
- Kjartan Svanur Hjartarson
- Kjartan Valur Kjartansson
- Kristín Þóra Jökulsdóttir
- Kristinn Júlíusson
- Kristinn Svansson
- Kristinn Þorri Þrastarson

- Kristján Harðarson
- Kristján Patrekur Þorsteinsson
- Lárus Konráð Jóhannsson
- Laufey Rut Guðmundsdóttir
- Leifur Björnsson
- Lóa Jóhannsdóttir
- Lúðvík Kemp Guðmundsson
- Margrét S Kristjánsdóttir
- Marinó Fannar Pálsson
- Matthías Skjöldur Sigurðsson
- Natan Örn Ólafsson
- Ólafur Björn Magnússon
- Ólafur Helgi Jónsson
- Ólafur Ingi Eiríksson
- Ólafur Konráðsson
- Ólafur Ómarsson
- Óli Ólafsson
- Ólöf Gyða Risten Svansdóttir
- Ómar Karl Valgarðsson
- Ómar Óskarsson
- Patrekur Patreksson
- Perla Þrastardóttir
- Pétur Bergmann Halldórsson
- Ragnar Adolf Árnason
- Ragnar Borgþór Ragnarsson
- Ragnar Mikael Halldórsson
- Ragnar Þór Valgeirsson
- Rannveig Guðmundsdóttir
- Raquelita Rós Aguilar
- Reynir Örn Björnsson

- Róbert Ingi Jónsson
- Sigtryggur Ómarsson
- Sigurbjörn Kristjánsson
- Sigurður Már Atlason
- Sigurður Snær Eiríksson
- Sigurjón Birgisson
- Sindri Már Sigfússon
- Sindri Sigurjónsson
- Sindri Þór Stefánsson
- Skúli Þór Árnason
- Snæbjörn Þórir Eyjólfsson
- Snævar Dagur Pétursson
- Snævar Geir Geirsson
- Snorri Hjörvar Jóhannsson
- Snorri Örn Daníelsson
- Sólberg Bjarki Valdimarsson
- Sonja Jónsdóttir
- Stefán Ingi Daníelsson
- Steinar Marinó Hilmarsson
- Steinar Þór Árnason
- Svava Dögg Björgvinsdóttir
- Sveinn Þórhallsson
- Sverrir Tryggvason
- Tómas Magnússon
- Unnar Kristjánsson
- Vala Rún Valtýsdóttir
- Valdimar Jónsson
- Valgeir Björnsson
- Valgerður Rós Morthens
- Víðir Orri Reynisson

- Vignir Karl Gylfason
- Viktor Blöndal Pálsson
- Viktor Smári Ágústuson
- Vilhjálmur Karl Ingþórsson
- Þór Adam Rúnarsson
- Þórdís Jóna Jónsdóttir
- Þórður Bragason
- Þorsteinn Eyþórsson
- Ævar Ísak Ástþórsson
- Ævar Þór Gunnlaugsson

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The focus of undergraduate Computer Science (CS) programs is typically to teach students programming, data structures, algorithms, and other similar foundations for a career in the computer industry. Much emphasis is typically placed on the creation of new code, but maintenance of code is rarely mentioned. For example, the ACM standard [1] only mentions maintenance of software a few times. However, writing code from scratch is by no means the only skill required in industry, as research suggests that approximately 60% of the cost of software is due to maintenance: fixing bugs, changing features, adding features to existing code, rewriting modules, etc.[2]. It is therefore important that students get practical experience reading and understanding existing code, but even in those cases where students need to study existing code, they will probably only have to read and understand dozens or hundreds of lines of code. Students will very seldom have to dive into codebases with tens of thousands of lines or more, which is what they most likely must do in industry.

Since the inception of the undergraduate CS program at Reykjavik University in 1988, students have gained practical experience through two 6 ECTS project courses at the end of the first and second semesters, as well as a 12 ECTS final project at the end of the undergraduate program. These project courses have had little or no prior code to build upon, however, resulting in projects where students have only experienced software development from scratch, but students have not experienced how to maintain software through the projects. This lack of exposure to practical software maintenance has long been discussed within Reykjavik University, with the intention of improving the learning outcomes of the program.

Attempts to address this elsewhere have mostly focused on involving students in open source software (OSS) [3]. At Reykjavik University, a course called "Open Source Software Development" was founded in 2012 by the author of this thesis, and run again in 2013 by a Google employee. Based on the experience from these courses, working with OSS has three major drawbacks. First and foremost, the course teacher is not the owner of the project. The owner is usually the person who decides whether changes proposed by students will be incorporated into the master branch of the software, and this person typically has other interests in mind than the learning experience of the students in the project course. Second, the availability of problems suitable for the particular students to work on is highly unpredictable in such OSS projects, and solving the current issues may be too complex for some of the students. Third, the problem domain of OSS is often something students are not familiar with, and getting acquainted with the problem domain takes time. While there may be cases where the latter two drawbacks are not present, the first problem—the ownership of the project—will always be a significant hurdle.

Based on these observations, the author of this thesis created the course "RU Internship" in 2013, with a focus on practical software maintenance. The design of the course drew from the experiences from the practical project courses and the Open Source Software Development course. At the time, Reykjavik University was using a system which implemented a combination of a Learning Management System (LMS) and a Student Management System (SMS). The system was written in Classic ASP [1] and had a number of design flaws, and it was clear that the system would be replaced within a short timeframe. We therefore decided to use the course to build a prototype for a new system, solving the same user goals. The aim of the course was to gain insights into the requirements for a new system from the experience. The student-developed system was named Centris and was proposed as a candidate for replacing the old system.

In the first iteration of the RU Internship course, the students had to write code from scratch, but as time progressed and the project evolved, students had an ever–increasing codebase to build upon. As anticipated, students were immediately familiar with the problem domain, due to the fact that they were already using the old system, and were (unpleasantly) aware of many of the flaws in that system. Their experience of using the old system meant that they had strong motivation to create a new system without similar problems. And, as the project owner, the course teacher could make all final decisions on which code to include and which not.

In spring 2015, the project had reached the status of a relatively large codebase, containing tens of thousands of lines of code. Having executed the course several times also meant that experience had been accumulated regarding how to teach practical software maintenance, both in terms of practical issues such as how the course was taught, and also regarding various technical questions regarding the codebase. In order to quantify these experiences, the learning outcomes for the course were rewritten for the spring semester 2015, and students enrolled in the course were asked to fill in a number of surveys to describe their experience before, during, and after the course. The following iterations of the course were then modified based on an analysis of the survey outcomes, and further potential modifications have been suggested.

This thesis is intended to take the insights gained in the many iterations of the course, and transfer them to any interested parties, both within the school and outside, for the benefit of current and future students.

The contributions of this thesis are:

- We describe the course, its motivation and history.
- We describe the learning outcomes, and how we measured the students perception of their learning.
- We describe the results of those measurements and lessons learned: which aspects of the course could be improved and which should remain unchanged.

The remainder of the thesis is organized as follows: We present a detailed history of the course in Chapter 2. In Chapter 3 we discuss the methodology of data collection, while the results from the data analysis are introduced in Chapter 4 and the key findings are discussed further in Chapter 5. Finally, we discuss related work in Chapter 6 before stating our conclusions in Chapter 7.

[1] Classic ASP is a web development platform from Microsoft, based on VBScript. It was used a lot 15–20 years ago, but has not been used for new web projects for several years.

Chapter 2

Course History

The "RU Internship" course was created for spring semester 2013. It was set up as a 6 ECTS course, and students were allowed to enroll twice in the course. At Reykjavik University, the semester is currently split up into two periods: a 12–week period, where students are typically enrolled in 4 courses, and a 3–week period, where students can focus on a single course.[1] Initially, the plan was to teach the "RU Internship" course during both periods, but when it became evident that the course format did not suit the 12–week period, only the 3–week periods were used.

In the course, students have focused entirely on a single project. Outside of the course, 12 students have completed 5 final projects directly related to the project. Also, 4 students have worked on the project during the summer in a paid position. In total, 182 students have been involved in the project, and have spent more than 30 thousand manhours working on the project.

To give an overview of the participation in the project, Table 2.1 shows the number of students in each semester. Column 2 ("Weeks") shows the length of the iteration period, column 3 ("New Students") shows how many students were involved with no prior experience in the project, column 4 ("Repeating") shows how many students were involved with prior experience, column 5 ("Total") shows the total number of students involved in the project during that semester, column 6 ("Final Project") shows how many students were working on a final project related to the project, and finally column 7 ("Paid") shows how many students were working on the project in a paid position.

This chapter is organized as follows. First, in section 2.1, we cover the background of the course. Second, in section 2.2, we describe each iteration of the course, to give the reader a clear picture of the evolution of the course. Finally, we summarize the major lessons regarding the context and execution of the course.

2.1 Inspirations for the Course

The course design was inspired by several experiences in the undergraduate program. The most influential courses were:

- The two practical courses at the end of the first and second semester, "Semester Project 1" and "Semester Project 2".

[1] The fall semester of 2013 had a different setup, with a single 15–week period where students were enrolled in 5 courses.

Semester	Weeks	New Students	Repeating	Total	Final Project	Paid
Spring 2013	12	10		10		
Spring 2013	3	6	10	16		
Summer 2013	8	4	1	5	3	2
Fall 2013	15	10	7	17		
Spring 2014	12	13	3	16		
Spring 2014	3	25	5	30		
Summer 2014	8		1	1		1
Fall 2014	12	4	1	5	5	
Fall 2014	3	8	3	11		
Spring 2015	12	2		2	2	
Spring 2015	3	24	3	27		
Summer 2015	8		1	1		1
Fall 2015	3	23	5	28		
Spring 2016	3	27	6	33	6	
Fall 2016	3	16	3	19		
Spring 2017	3	16	3	19		
Total		182			16	4

Table 2.1: Total number of students involved in the project.

- The course "Open Source Software Development" (OSSD) which was taught twice in 2012 and 2013.
- The course "Object–oriented Programming in C++" (OOP) which was taught in 2005–2007.

The practical courses at the end of the first and second semester are built around the fact that students enter those courses with little background in software development, and are basically learning the ins and outs of how to analyze, design, and code a small project from scratch. We believe this to be essential to the development of students in CS, but as previously mentioned, these courses do not train students in software maintenance.

The OSSD course was an experiment which was executed twice, in both cases during the summer semester, when much fewer students decide to spend time studying. The author of this thesis taught the course in 2012, and which was a valuable experience in how to teach software maintenance.

The OOP course is relevant to the design of this course as well, because in the OOP course students built on their background in C++ to take part in a single project, written in C++, and all the students in the course worked towards the same goal, as the students were split up into groups which were working on individual parts of the system. This was the first time this was tried within the CS curricula at Reykjavik University. The project in question was to create a networking multi–player board game in C++, but for various reasons the project never took off, partly because the programming language was not really suited for this kind of software, and neither was the architecture. In hindsight, a web-based platform would have been much more relevant for the project itself, and some other type of project should have been chosen to give students experience in C++ programming. However, the fact that all students were working together towards a common goal turned out to be

memorable for all participants, and is the decisive factor which influenced the creation of the course RU Internship.

2.2 Course Iterations

2.2.1 Spring 2013 (12 week period)

12 students started the course, but two students quit shortly after the course started. The first meeting was held on the 20th of December 2012. A Facebook group was created for the group, and most of the communications took place there, other than face–to–face communications. Work began on the 14th of January. Very soon the project gained the working title "Centris".[2]

Due to the fact that the SMS system had only a handful of users, in contrast to the LMS which is used extensively by students and teachers, it was decided to focus on the SMS system at the beginning. The school had access to the source code of the old SMS/LMS system, but we decided to start from scratch due to the following factors:

- The code was written in Classic ASP, which had not been taught at Reykjavik University for more than 10 years.
- The code was highly coupled, i.e. change in one part of the system could easily affect other seemingly unrelated parts of the system.
- The code had little separation of layers, so a single file could contain SQL code, HTML and VBScript.
- There were no automatic tests available for the system, and it would have been difficult to create such tests.
- The code had multiple security issues, such as SQL injection.[3]

We decided to use an API-first architecture,[4] where all operations were available via a given API, which would then be used by separate clients. Testability of the system was also a high priority. However, in order to increase the chance of a successful transition from the old system to the new, it was decided to start with a copy of the system database, where sensitive information had been removed.

The first few weeks were used to:

- Learn about Git,[5] Web API,[6] JavaScript front-end frameworks, application architecture etc.
- Find out where to store the code, issues, time logging, set up the physical facilities such as a whiteboard etc.

[2]The inspiration for the name came from the fact that the facilities the group had were in the center of the school. "Centr" is also a reference to popular applications at the time such as Tumblr, Flickr, and Twttr, while "is" refers to the Icelandic country code top–level domain.

[3]SQL Injection is one of the most common security vulnerability found in web projects.

[4]API is an acronym which stands for "Application Programming Interface".

[5]Git is a Version Control System, written by Linus Torvalds.

[6]Web API is a type of API which is available via HTTP, usually REST–based.

The first code was checked in to source control on the 21st of January 2013; a run-of-the-mill C# Web API project. At the end of January, the project finally found its place in a repository on BitBucket.[7] Initially, there was only a single repository, containing both the client and server code. A week later, the first client code was checked in: a test HTML/Angular[8] application. At that time, the decision to use Angular was not final, since other frameworks (KnockoutJS, BackboneJS) were still an option, but this was decided about a week later.

In early February 2013, a User Interface (UI) theme was chosen for initial development, based on Bootstrap 2.[9] As it was only meant to be used temporarily, it was replaced one year later.

At first, there were some discussions about using a Content Management System (CMS) for the front end, making it easier for users to change the behaviour of the system. After a meeting with developers at Betware[10] in mid-February, it was decided to put the CMS ideas on the shelf, and focus on an Angular Single-Page App[11] architecture.

On February 21st, the first meeting with expected users was held. The users came from 4 different academic departments (School of Computer Science, School of Law, School of Business, and School of Science and Engineering), plus the Teaching department (ísl. Kennslusvið). After that meeting, each school department made a list of desired features (compared to the old system they were using). Those features were either not present at all in the current system, or were incomplete or buggy. This list included features such as:

- Generating lists of students in various ways (filter by department, majors, gender, registration semester, completed ECTS, etc.).
- Attaching notes to individual students.
- Keeping track of which students had received scholarship awards.
- Maintaining multiple versions of the curriculum (course templates, majors).
- Ensuring that classrooms will not be double booked.

The group set out to implement those features, some of them were completed during this semester, others took longer to implement.

In early March, a rudimentary language support was implemented in the client.

Some time was spent on worries that the client would become too large, and that it would take too much time to load all modules at startup. A considerable amount of time was spent on finding solutions to this. Afterwards, these worries were probably unnecessary. At the end, it became obvious that any solution to this (loading a given module only when it was really needed) would be a hack, since there was no proper support for lazy-loading modules in Angular at the time.

Another thing that in hindsight should have been done differently was to enforce certain coding standards early on, preferably by using tools to help with this. It soon became

[7] BitBucket is a website which offers free Version Control hosting, with support for Git, Subversion and other Version Control Systems.

[8] Angular is a client–side web framework written in JavaScript.

[9] Bootstrap is a CSS/JavaScript framework, maintained by developers at Twitter.

[10] Betware is an Icelandic software company, one of its owners at the time was also a faculty at Reykjavik University.

[11] A Single-Page App is a web application where only the first page is retrieved via standard HTTP, while the data for other pages is retrieved asynchronously using AJAX.

Programming language	LOC
JavaScript	3,600
HTML	1,800
CSS	200
C#	7,300
Total	12,900

Table 2.2: Lines of code after the first semester.

obvious that the students were using different coding practices, and this needed to be standardized. A part of the problem was simply that because a new technology was being used, no one knew what the best practices were.

Also, at first there was no UI standard. so it was not clear what practices should be used regarding page layout, what controls to use, etc. This resulted in a number of components being added to the project, including components with the same behaviour. Again, this was something that probably needed time and experience to resolve.

2.2.2 Spring 2013 (3 week period)

A new sprint began in April 2013. 16 students were enrolled, of which 6 were new, and the group could now focus entirely on the project, as no other courses were running concurrently.

At the end of the 3 week period in April/May 2013, the "cloc" program was used to count how many lines of code had been written. The results are shown in Table 2.2 (Note that some of these numbers may include auto-generated code).

In late May/beginning of June 2013, a Jenkins Continuous Integration (CI) server was installed. At around the same time, JSHint[12] was introduced to ensure all JavaScript code would pass through certain requirements. This was also very helpful, and should be one of the first tools to install when creating a new project, since the cumulative list of errors this tool reported initially contained 1000s of issues (missing semicolons, undeclared variables and so forth).

2.2.3 Summer 2013

In summer 2013, 2 students worked on the project in a paid 50% job, working on features in the SMS project.

Also, 3 students worked on a related project: a Graduation module called "Genesis". This module was actually deployed later by Reykjavik University, and used in production.

2.2.4 Fall 2013 (15 week period)

Fall 2013 saw 17 students enrolled in the course. 10 were new, i.e. had no prior experience of the project.[13] The group was working on the SMS, LMS and Graduation projects, plus an iOS app for students.

In October 2013, the project was changed such that it was no longer possible for students to submit changes to the "master branch" of the project. From here after, all changes

[12] JSHint is a tool which parses JavaScript code, and points out compile–time errors in the code.

[13] 3 of the new students had insufficient background due to a mistake in course registration. This meant that those students were struggling more than other students, and benefitted not much from the course.

were done using "pull requests" (PR).[14] This resulted in much better control over what code exactly would end up in the master branch of the code. In hindsight, this should have been done much earlier.

In the fall of 2013, the first version of a ElasticSearch[15] server was added to the project.

At the end of this semester, it was becoming quite clear that working on this project in competition with other courses was not the optimal setup. It was also clear that students needed a certain technical background. In order to support this course better, and also to strengthen the web development emphasis line of the undergraduate program, certain changes were made. The course "Web programming II" was overhauled, and was taught in the new format in spring 2014. A new course, called "Web services" was created, and was first taught in fall 2014. The former course gave students the required technical knowledge for working on the client side of the project, while the latter course focused on the server side, and how to write Web APIs. Both courses were built to teach various techniques, i.e. their focus was not only on material directly related to the Centris project, although the experience from the Centris project was very influential in the design of those courses.

2.2.5 Spring 2014 (12 week period)

Spring 2014 saw a new group of students arrive, 16 students were working on the project, of which 13 were new. The focus was on the SMS project and the Graduation project. This was the last time the course was taught during the 12 week period, concurrent with other courses. At the same time, the first version of the course "Web Programming II" was taught using a new curriculum.

In February 2014, the project was split up into more repositories: one for the backend (i.e. the API) and one for the frontend.

2.2.6 Spring 2014 (3 week period)

In April/May 2014, a group of 30 students worked on the project. 25 of those had no prior experience of the project. Most of the students in that group had completed the new "Web Programming II" course, which meant that their preparation was better than in previous iterations. The group worked mainly on the SMS and LMS front end projects, but some work was done on the API project as well, and one group created a cross-platform smartphone application. In this round, the first steps were taken to incorporate a RabbitMQ server[16] into the system.

Shortly before the course started, the frontend was split up into two separate repositories: one for the LMS part and one for the SMS part. Steps were also taken to make the project more open source friendly, by moving documentation from files in Google Drive to markdown documents that were part of the project at Bitbucket.

Even though it was good to have a large group of students working on the project, it was probably too large for a single instructor. It was clear after this semester that the optimal size would be somewhat lower, or 20–25 students.

[14] A separate branch is created for a given feature, and when that feature has been completed, a pull request is issued to request that the changes from the feature branch will be incorporated into the master branch.

[15] ElasticSearch is an in–memory database, which allows for very fast lookups.

[16] RabbitMQ is an open–source tool which implements AMQP, and is currently one of the most popular message queue implementation.

2.2.7 Summer 2014

In the summer of 2014, one student was hired to work on the project in a full-time position. The student spent most of his time doing code cleanup, i.e. ensuring the SMS project was ready for end-user testing.

2.2.8 Fall 2014 (12 week period)

In fall 2014, there was no "regular" activity going on, but two groups were working on the project as a final project. Both groups were working on functionality which was not a part of the core SMS/LMS projects: one group was working on scheduling application using third party tools, while the other group was working on a tool which allows users to specify what happens when certain events are posted to the message queue. Both were standalone products, although they were both related to the Centris API.

2.2.9 Fall 2014 (3 week period)

In December 2014, 11 students were enrolled in the course, 8 of those had no prior experience of the project. The main focus was on the SMS project, both the frontend and the backend.

At this point in time, other courses had been modified based on the observations from the first iterations of RU Internship, and had been taught once each using the new design. It was therefore possible to change the prerequisites for the RU Internship course. Students could now enroll in the course if they had completed the course Web Programming I, which covers the basics in HTML, CSS and ASP.NET MVC using C#, and one of the following:

- Web Programming II, which covers client side web development: HTML5, CSS3, JavaScript and Angular
- Web Services, which covers server-side web service programming using C# Web API, and NodeJS

As this iteration was the first one where students had the opportunity to have completed these prerequisite courses, they were generally better prepared than in previous iterations, and had to spend less time getting acquainted with the technical aspects of the code.

2.2.10 Spring 2015 (12 week period)

In January 2015, the SMS project was tested by potential users and was well received.

Also, 2 students did a final project which involved creating a discussion module which could be used in several places in the project.

2.2.11 Spring 2015 (3 week period)

In this iteration, 27 students took part, 24 which had no prior experience. The focus in this iteration was first and foremost on the LMS part, but with occasional tasks originating in the SMS project. In fact, it was after this iteration that the LMS part finally started to look like a proper application.

The data gathering took place in this iteration. This is covered in detail in Chapter 3.

2.2.12 Summer 2015

A single student worked on the project for 2 months in a full–time position, mostly focusing on tasks belonging to the LMS project, and the APIs required for those tasks.

2.2.13 Fall 2015 (3 week period)

At the end of the fall 2015 semester, 28 students took part in the course, 23 for the first time. The focus was now almost entirely on the LMS part of the system.

The backend of the project started moving towards a "microservice" architecture,[17] with a separate API written in NodeJS for a particular service in the system.

2.2.14 Spring 2016 (12 week period)

It was not the initial plan to teach the course in the first 12 weeks of the semester, but due to the fact that at this point the school had started the selection process for a new LMS system, and that we wanted to speed up the development, a handful of students were allowed to work on the system during this period. The incentive to finish those tasks turned out to be high enough in this case, and students were generally highly motivated, knowing that the system could possibly be chosen to be used by the school.

2.2.15 Spring 2016 (3 week period)

27 students were enrolled in the course, 21 which had no previous experience. In this iteration, the course did an experiment with switching over to Jira as a project management software. As it turned out, the learning curve for Jira is a bit high, and due to lack of time the instructor never really had the time to fully incorporate it into the schedule. Some time was also spent to create a Continuous Integration (CI) pipeline on Azure. The documentation for the project was also improved significantly, when a series of videos which explained various concepts and processes were created. These videos were well received by students.

Two students did a final project with a goal of writing a standalone project to be used by the student organizations, but which would be able to use data from Centris. Another group of 4 students did a final project which added to Centris the ability to predict which students would be in danger of dropping out of their studies.

In this semester, the client repositories, i.e. both for the LMS and the SMS, were made public on BitBucket, and therefore officially became open source software.

2.2.16 Fall 2016 (3 week period)

19 students enrolled in the course, 16 with no prior experience. The main focus of this iteration was on improved testing and gradual improvements of all parts of the system, both LMS frontend, SMS frontend, and the API.

During the summer, the school decided to go another route in its choice of an LMS system, and this influenced the course quite a bit, since the possibility that the system would be used at the school was no longer an option, and the possibility that the system would be used elsewhere was only distant.

[17] In a microservice architecture, there are multiple small services instead of a single monolithic service.

Programming language	Comments	LOC
JavaScript	15,400	86,000
HTML	1,000	20,300
CSS + LESS	1,800	52,300
Language files		13,100
C#	28,900	76,400
Java	1,000	5,800
Total	48,100	253,900

Table 2.3: Lines of code after the spring 2017 semester.

2.2.17 Spring 2017 (3 week period)

In spring 2017, 19 students were working on the project, 16 which had no previous experience of the project. In this iteration, the remainder of the A requirements for the LMS project were cleaned up.

In this iteration, we had for the first time mostly students which had learned how to use Angular 2.x, while the project used Angular 1.x. The difference between those two versions of the framework is substantial, but it turned out not to have too much of an impact on the productivity of the students, as the project had already moved to an architecture similar to the one recommended in Angular 2.x applications.

At the end of this iteration, we ran the cloc program again on the code repositories. The results are shown in Table 2.3.

2.3 Discussions

The experience from these four years or so of teaching the course have led us to make the following observations:

- We recommend that the course should be limited to 20–25 students per instructor. Having students in the course with prior experience of the project can actually mean that the number could be raised a little bit, since those students can provide assistance to the junior developers.

- It is less likely that students will get stuck on difficult problems if they work together in pairs, instead of working alone.

- Students need clear deadlines, and if they are working on a project like this without deadlines in parallel with other courses, they will in general not produce much code. This is mostly due to the fact that the other courses take up the time of students. Because of this, it became evident that running the course during the 3 week period, when students are completely focused on this single course, gives the best result. Alternatively, the course could also be taught during a period when the students are enrolled in other courses, but with hard deadlines on their tasks. This can be difficult to achieve due to the fact that the estimated hours needed to complete a given task can vary considerably between individual tasks.

- Students need clear guidelines on how to write the code: the coding standard must be available and thorough, and there must be some UI standard. Otherwise, the code

becomes a mess, with multiple coding styles used, and with multiple overlapping components used.

- Most students will not gain the oversight necessary to fully comprehend the entire project during the short time they are involved in the project, and only a handful of students are capable of making large-scale design decisions. This is similar to the experience of people starting a new job, which often takes weeks or even months to get properly acquainted to. This means that a project of this size and degree needs to have a "chief architect" which effectively makes the design decisions. Students are however perfectly capable of grasping the design of parts of the system, even relatively large parts. Finally, the input of students is very important, both in terms of smaller decisions, and larger ones as well.

We believe that all of these observations could apply to industry as well.

Chapter 3

Methodology

In this chapter, we describe the data collection performed during the spring 2015 semester. We first describe the learning outcomes of the course, which were written before the start of the spring 2015 semester and served as the focal point of the data collection. We then outline the course itself, in terms of the learning assessments, student population and project tasks. We then describe in detail the data collection instruments used in the study.

3.1 Learning Outcomes

The learning outcomes were written in spring 2015, before the course started. They were written with the following goals in mind:

- That completion of the learning outcomes would be necessary as a preparation for a career in software maintenance.
- That the learning outcomes would match what we had concluded would be achievable in the course from previous versions.

A set of learning outcomes were written by the author of the thesis, and then reviewed repeatedly by key members of the faculty, until all stakeholders were satisfied. The final version of the learning outcomes is listed in Figure 3.1. They were then used as the basis for data collection.

3.2 Course Description

3.2.1 Course Setup

In spring 2015, the course was taught during the 3 week period between 27th of April and 17th of May, for a total of 15 days. All students had completed Web Programming II, but only very few had completed Web Services. The focus of the course was therefore on the client side.

The course had 27 students, of which 24 had no previous experience in the project, while 3 students had completed the course once. Students chose themselves to enroll in this course, but only those students which had completed the necessary prerequisites were eligible. The range of the average grade of the enrolled students was from 6.74 to 8.93 on the scale 0–10. Students had all completed at least 3 semesters (90 ECTS), 5 students had completed 4 or 5 semesters (120 or 150 ECTS). 5 of those students were female, 22 were male. The average

At the end of the course, the student should be able to:

Skills:
LO1: Read code, written by others, and describe its functionality.
LO2: Improve code and documentation written by others.
LO3: Apply the various standards and rules set by the project.
LO4: Apply best practices in version management.
LO5: Address issues raised by code reviewers.
LO6: Operate in a group, which is itself a part of a larger team.
LO7: Present (either orally or in writing) your work.

Competencies:
LO8: Identify which parts of a project documentation needs improving.
LO9: Identify which parts of a project codebase needs improving.

Figure 3.1: Learning outcomes for the course.

age of the students was 25.8 years, the youngest student was 21 years old and the oldest was 38 years old. 22 was the most common age (8 students), which is no surprise given the fact that most of these students start their study at the age of 20 and most of them were in their 4th semester. It was our impression that this group of students represented more or less a cross section of the student population at the time.

For logistical reasons, the group had to be divided in two groups: one which was located in a traditional classroom, and another group which was located in a special room belonging to the project.[1] The first room had a traditional setup: rows of tables, two and two tables together, all chairs facing the whiteboard/instructor. The other room had a very different setup: there was one circular table and one long table where students sat facing each other. There was no instructor seat in that room, and a single whiteboard on wheels which could be moved around. This separation between those two rooms was accidental, but turned out to provide us with useful data about the usefulness of each setup.

Also for logistical reasons, the students had to help first-year students in another course also taught by the instructor. Each student had one day out of the 15 which was dedicated to this assistance. This also turned out to be helpful in providing useful data.

3.2.2 Work Organization

The code the students had access to consisted of several projects containing at least tens of thousands of lines.[2] In this course the focus was on 4 of those projects, and most of the effort was in a single project, the LMS project.

Students worked on assignments in pairs, since that arrangement had given the best results in previous iterations of the course. First, the students were asked whether they wanted a particular "buddy" to work with. Those that did not ask for a particular partner were paired together such that no pair had two members with a low average grade. For those students which requested a particular buddy, there was no restriction on the pairing. This resulted in 3 pairs where both students had a relatively low average grade.

[1] The latter room was called "the glass cage", as its walls were entirely made of glass.
[2] Program line count (cloc) data from this time is not available.

Each pair was assigned a single task to begin with. Each task was set up such that it would be doable in a few days at most. Tasks were created by the instructor, and once they were completed, the resulting changes to the code were reviewed.

Students received guidance from the instructor when the task was assigned to them, i.e. they were given hints on how to implement their task, and where to look in the source code. The instructor was also available to guide the students upon requests.

The group did not follow a specific methodology such as Scrum, which meant that there were no daily meetings during the course, although there was one meeting in week 2 were students could discuss the progress. Otherwise, the students used the Facebook group for various informal discussions and questions during the course.

The instructor had several responsibilities in this course:

- Help students get acquainted with the codebase, documentation, etc.
- Write and assign issues to student pairs.
- Assist students when they needed help.
- Review the code written by students.

3.2.3 Course Assessment

The final grade was a pass/fail grade. To pass the course, students had two requirements:

- They had to be involved in at least one accepted pull request.
- They had to complete at least 150 hours of work during the course.[3]

Students were required to log all working hours in an internal system (an instance of the Redmine open-source project management tool) and this log was then used to determine whether the students had passed the latter requirement.

Students did not have access to the master branch of the code, and had to submit a pull request for changes they wanted to make to the code. PRs were reviewed by the instructor and 3 external reviewers, all of which had previous experience in the project, as well as hands–on experience in the given technological domain from their current work. All issues raised by reviewers had to be resolved before a PR could be accepted.

3.3 Data Collection

Students answered a total of 5 questionnaires: one before the course started, one at the end of each week (i.e. three times total), and one at the end of the course. All previous participants in the project were asked to read the questionnaires beforehand, several comments were received and the questions were adjusted accordingly.

The questionaires were posted online. Students had to identify themselves in each questionnaire using their Social Security Number (SSN) in the first question in each questionnaire. Students could answer the questionnaires at any time, and there was no attempt made to ensure they completed them in isolation. All questionnaires were hosted at freeonlinesurveys.com.

Additionally, three methods of data collection were used.

[3]Each ECTS unit should represent 25–30 hours of work, and the course was 6 ECTS.

	Question	Details
1	How much experience have you had in software development?	Students could choose between 3 options, see Table 4.1.
2	In your own words, describe what you think each learning outcome really means:	Students were asked to comment on each of the 9 learning outcomes, which are listed in Figure 3.1.
3	How good would you consider yourself in the following, compared to your fellow students in the School of Computer Science?	For each learning outcome, students could pick one out of 7 options, see Section 4.3.5.
4	Rank the learning outcomes that you expect will be emphasized most in the course.	Students were given 3 drop down boxes, and could select any of the 9 learning outcomes in each drop down. Learning outcomes are listed in Figure 3.1.
5	Rank the learning outcomes that you find most exciting in the course.	The options were the same as in the previous question.
6	What motivated you to enroll in this course? Check all which apply.	Students could check any of the predefined options, or add their own. See Table 4.5

Table 3.1: The questions in the pre–course questionnaire.

- Students were asked to maintain diaries of their work.
- The instructor observed the class and made notes.
- A focus group was held after the course.

We now describe each of these data gathering methods.

3.3.1 Pre-Course Survey

This survey was set up to study the background of the students, and their expectations for the course. There were 7 questions in the questionnaire, of which 4 questions asked students about the 9 learning outcomes in the course. The questions can be seen in Table 3.1.

Students were sent one email at the beginning of the course, on the 26th of April, where they were encouraged to complete the questionnaire. This email was also displayed under the "Announcements" section of the course homepage. A reminder was then posted to the project group page on Facebook one day later.

In total, 25 students of the 27 completed the questionnaire.

3.3.2 Weekly Surveys

The students answered the weekly questionnaire 3 times, i.e., at the end of each week. The purpose of this survey was to monitor the progress of the students, whether they perceived that they were making progress on the learning outcomes of the course, and their perception of the project itself. Students were asked to rate their improvement on each of the learning outcomes, and were also asked to rate how much the project had improved. There were

	Question	Details
1	Looking back on the week, to what degree did you improve your knowledge and skills in the following?	Students were asked to comment on each of the 9 learning outcomes. The options they were given are listed in Section 4.2.1.
2	In your opinion, how has the quality of the project codebase improved this week?	Students could pick one out of 5 options, see Section 4.2.2.

Table 3.2: The questions in the weekly questionnaires.

therefore 2 questions students had to answer each week, of which one asked about each learning outcome, and one for the overall course. The questions can be seen in Table 3.2.

An email / course announcement was sent to students before each survey: on Sunday at the end of Week 1 for the first survey, Wednesday in Week 3 for the second survey, and finally on Friday one week after the course had ended, for the last survey.

21 student answered the first weekly questionnaire, 25 students answered the second questionnaire, and 19 answered the third questionnaire.

3.3.3 Post-Course Survey

The students answered a questionnaire when the course had been completed. This questionnaire had 4 questions, 2 of which were about the 9 learning outcomes. The aim of this survey was to learn more about the impression students had of the learning outcomes, and how they compared to their impression before the course started. The questions can be seen in Table 3.3.

An email / course announcement was sent out on Friday, a few days after the course had been completed, and a notification was posted to the Facebook group 4 days later.

19 students answered this questionnaire.

3.3.4 Diaries

During the course, each student kept a diary where they noted what they had done during each day. The diaries were stored in shared documents on Google Drive which were open for all students in the course as well as the instructor. Students were reminded periodically to fill in the diary, usually via a post to the Facebook group of the course.

The students were given the following instructions:

> For each day, write down in a single paragraph or two what you did today.
>
> In particular, include any of the following if applicable: Were there any “a-ha!” moments, and if so, what caused it? Was there anything that made you particularly frustrated? If so, what? What was the complexity of the tasks you had today? Was there any noticeable improvement in your understanding today, or did the code improve significantly? What gave you the best support in solving the tasks? This could include Assistance from teacher, Assistance from other students, Project meetings, Project documentation, Project videos, Videos for other courses (Web Programming II, Web Services, etc.), Code review, Comments from reviewers on Pull Requests, External resources (stackoverflow, blogs etc.)

	Question	Details
1	What would you like to have learned in other courses before you enrolled in this course?	Students could enter a free–text reply.
2	What made the most difference in the course, i.e., what helped you the most in learning what you did learn?	Students could enter a free–text reply.
3	Name 1-3 things you liked about the course and should be kept unchanged.	Students could enter a free–text reply.
4	Name 1-3 things you didn't like about the course and should be improved	Students could enter a free–text reply.
5	How good would you consider yourself now in the following, compared to your fellow students in the School of Computer Science?	For each of the 9 learning outcomes (see Figure 3.1, students had to pick one option out of 7, see Section 4.3.5.
6	In hindsight, how good do you think you were when the course started, compared to your fellow students?	Students had the same options as in the previous question.

Table 3.3: The questions in the post–course questionnaire.

3.3.5 Observations

During the course, the instructor observed the behaviour of the students. This was done in an informal way, and the instructor was also busy with other duties. A couple of interesting points were collected, that are discussed in Chapter 4.

3.3.6 Focus Group

A focus group was held after the course had been completed. The aim of the focus group was to gain insights into the impressions of the students via different methods, and to give the students the opportunity to bring up certain topics which were not mentioned in the surveys.

All students were given the opportunity to join, but only four students took up the offer. The reason why there were not more participants is probably because most students had started their summer jobs at the time when the focus group was held, and were therefore busy. The focus group was held on a Thursday afternoon between 17:00 and 18:30, approximately 3 weeks after the course had been completed. There were 9 questions which the discussions centered around (listed in Figure 3.2. Each participant was given a chance to answer each question, but the participants were otherwise allowed to express themselves freely. The seating arrangement was informal, i.e., participants were sitting and facing each other, with no table in the center. The conversation was recorded and participants were informed that the recordings would only be used as a part of the data collection and would be destroyed afterwards. The students participated equally (more or less) in the conversations.

- What was the main thing you learned?
- Did you learn what you expected to learn?
- Did you feel you were as productive as you expected? a) individually, b) the group
- Did you expect the difference between working in a big group vs. working in a small group to be small? What is your current opinion on that?
- How much do you think students use external help, such as asking questions on stackoverflow.com, posting issues on github project pages, etc.?
- Was it common that you became stuck in given assignments? What was that experience like? Did you learn from being stuck?
- Was there anything in particular that helped you getting "unstuck"?
- What did you think of answering the surveys and keeping the diary?
- When answering the question "In hindsight, how good do you think you were when the course started, compared to your fellow students?", how did you compare yourself to other students?

Figure 3.2: Questions asked during the focus group.

Chapter 4

Results

In this chapter, we present the results from the data gathering methods, and discuss some interesting points seen in results from individual instruments. In Chapter 5, the overall results are then summarized and discussed.

4.1 Pre-Course Survey

4.1.1 Question 1

How much experience have you had in software development?

The results are shown in Table 4.1. The table shows that 3 out of every 5 students only had experience from school work. This is not surprising, as most students have little experience in software development. Also, we conjecture that those students which seek more experience are probably more likely to enroll in this course, since it will provide them with experience which resembles situations in industry.

4.1.2 Question 2

In your own words, describe what you think each learning outcome really means

The purpose of this question was to ensure that their understanding of each learning outcome was at least similar to the understanding of the authors. The students answered in Icelandic. We examined those answers and gave each answer a grade between 0 (the student did not understand the learning outcome at all) and 1.0 (the student fully understood the learning outcome). The average grade for each learning outcome was then calculated. The results are shown in Table 4.2.

Previous experience	Percentage
My only experience comes from school	60%
I've done a bit of programming outside school	28%
I've had 1 - 2 summer jobs	12%

Table 4.1: Previous experience of students.

Learning outcome		Total
LO1	Read code, written by others, and describe ...	0.98
LO2	Improve code and documentation written by ...	0.91
LO3	Apply the various standards and rules set by ...	0.90
LO4	Apply best practices in version management	0.86
LO5	Address issues raised by code reviewers	0.91
LO6	Operate in a group, which is itself a part of ...	0.99
LO7	Present (either orally or in writing) your work	0.98
LO8	Identify which parts of a project documenta ...	0.85
LO9	Identify which parts of a project codebase ...	0.88

Table 4.2: How well students understood the learning outcomes, in the original order.

The learning outcomes were mostly well understood. Two students did not understand **LO4** at all, and two students did not understand **LO8**. Other answers indicated either full understanding, or students mostly understood what each learning outcome meant.

It was interesting to note that in **LO5**, many students mentioned that it was important to be able to accept criticism without taking it personally, i.e. not letting their ego get in their way.

4.1.3 Question 3

How good would you consider yourself in the following, compared to your fellow students in the School of Computer Science?

We asked the students to rate themselves in any of the 9 learning outcomes. A similar question was asked at the end of the course. The results from those two questions are compared in Section 4.3.5.

4.1.4 Question 4

Rank the learning outcomes that you expect will be emphasized most in the course

Students were given the option to rank the 3 learning outcomes they believed would be emphasized the most in the course. The students answered this question as shown in Table 4.3.

The number for the "Total" column was calculated such that votes for the Most emphasized column were multiplied by 3, the votes for the Second most emphasized column were multiplied by 2, and the votes for the Third most emphasized column were used as is, and those results were then added together.

The results show a clear distinction between learning outcomes 1, 2, 3 and 6 on one hand, and 4, 5, 7, 8 and 9 on the other.

4.1.5 Question 5

Rank the learning outcomes that you find most exciting in the course

The students answered this question as shown in Table 4.4. Similar methods were used to calculate the numbers for each learning outcome as in the previous question.

It is rather interesting to note that students do not find the thought of having to read code written by others very exciting. Improving code is, on the other hand, the highest rated

Learning outcome		1st	2nd	3rd	Total
LO1	Read code, written by oth ...	7	7	2	37
LO2	Improve code and docu ...	5	9	2	35
LO6	Operate in a group, which ...	5	2	8	27
LO3	Apply the various stan ...	5	3	5	26
LO4	Apply best practices in ver ...	1	2	3	10
LO5	Address issues raised by ...	1	0	3	6
LO7	Present (either orally or in ...	0	1	2	4
LO8	Identify which parts of a ...	1	0	0	3
LO9	Identify which parts of a ...	0	1	0	2

Table 4.3: What students thought would be emphasized the most, in order of calculated grade.

Learning outcome		1st	2nd	3rd	Total
LO2	Improve code and docu ...	10	4	3	41
LO6	Operate in a group, which ...	8	3	4	34
LO3	Apply the various stan ...	3	5	3	22
LO4	Apply best practices in ver ...	2	6	1	19
LO5	Address issues raised by ...	1	2	4	11
LO9	Identify which parts of a ...	0	3	4	10
LO1	Read code, written by oth ...	1	1	1	6
LO8	Identify which parts of a ...	0	1	2	4
LO7	Present (either orally or in ...	0	0	3	3

Table 4.4: What students thought would be most exciting, in order of calculated grade.

learning outcome. In general, students seem to think that the learning outcomes which will receive the most emphasis are also those they find most exciting, apart from **LO1**.

4.1.6 Question 6

What motivated you to enroll in this course? Check all which apply

When answering this question, students were given the options in Table 4.5. This list was written with the aim to include options which were considered plausible reasons for students to enroll. However, students could also add new options, but none deemed that necessary.

The results clearly indicate that students seek more experience in software development. The results also seem to indicate that the reason why students chose this course were genuine, i.e., they did not enroll in the course just because there were no other interesting courses available.

4.2 Weekly Survey

Students answered the following survey at the end of each week.

Reason	Votes
I need more experience in software development	25
I'm interested in the topics of the course	21
I'm interested in being a part of the team that writes the next MySchool	20
I need more experience working in large teams	18
I'm interested in the use of computers in education	18
I'm interested in improving the teaching environment in the school	17
Because the teacher made me interested in the course	17
My friends recommended the course	10
Because there were no other interesting courses available	1
Other (Please Specify)	0

Table 4.5: What motivated students to enroll in the course.

4.2.1 Question 1

Looking back on the week, to what degree did you improve your knowledge and skills in the following?

For each of the learning outcomes, the students were asked to rate their improvement, by selecting one of the following:

(a) Did not apply this week

(b) I have not improved in this area at all this week

(c) I have improved somewhat this week but I'm still not competent

(d) I have improved somewhat and feel competent

(e) I feel like I have mastered this topic

Figure 4.1 shows the results. Each learning outcome has 3 horizontal bars: the topmost bar represents week 1, the middle bar represents week 2 and the bottom bar represents week 3. Within each bar, the five choices are colored with different colors.

It is interesting to note that in 3 cases (in **LO1**, **LO5** and **LO6**), the number of students which said that they "had mastered the topic" dropped between weeks 2 and 3.

The impact of **LO3** seems to have been the most in week 1, as most students completed their first tasks in week 1, and were forced to ensure that the changes they made to the code were in conformity with the coding standard of the project. In weeks 2 and 3, most students seem to have adjusted to the requirements of the course in this regard.

Since students did not have to present their work until at the end of the course, **LO7** is the one with the least change. It is not entirely impossible that this learning outcome will be altered or removed, in part due to the fact that students are required to present their work in other courses as well.

In general, students seem to perceive that they did improve in most of the learning outcomes in the course.

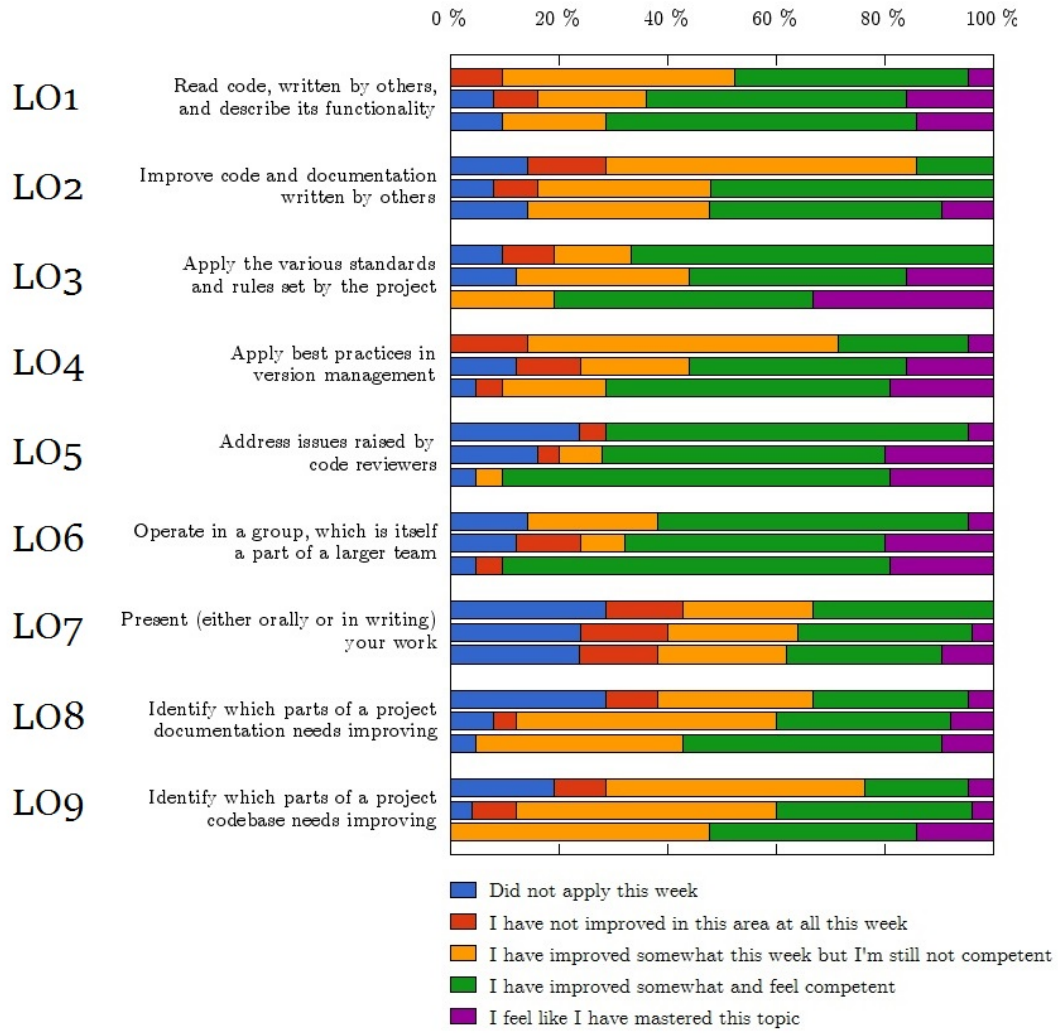

Figure 4.1: Improvements in learning outcomes.

4.2.2 Question 2

In your opinion, how has the quality of the project codebase improved this week?

Here, the students were asked to rate the improvement of the code, by selecting one of the following options:

(a) It is worse now than what it was

(b) It is the same

(c) It has improved slightly

(d) It has improved quite a lot

(e) It has improved very much

The results are shown in Figure 4.2. Thankfully, the students seemed to think that the code was improving. It is also interesting to note that the biggest improvement seems to have taken place in week 2. This may indicate that many students spent less time in week 3 to write new code, but were instead more focused on cleaning up, finishing their tasks, and

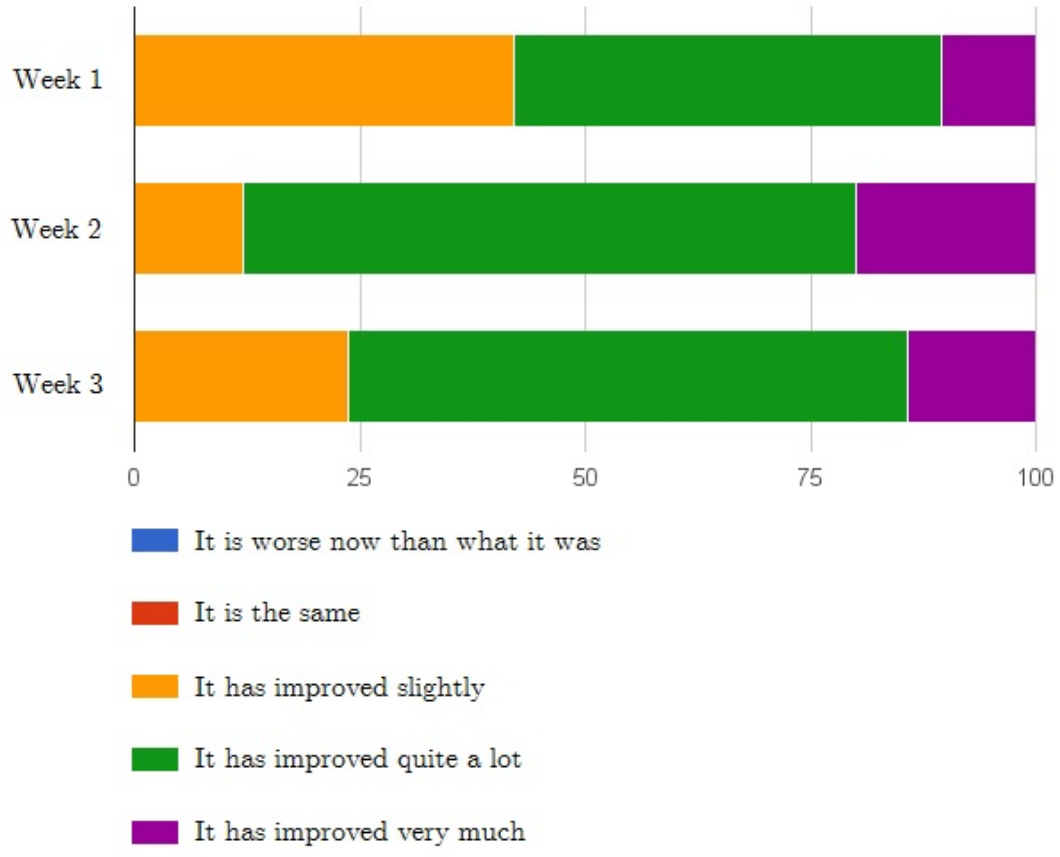

Figure 4.2: How students perceived how the quality of the code improved.

generally ensuring they were leaving the project in good shape. This may have impacted the perceived change, especially in the number of finished features in the project.

4.3 Post-Course Survey

Below is a list of the questions asked in the questionnaire students answered after they had completed the course, and the results for each question.

4.3.1 Question 1

What would you like to have learned in other courses before you enrolled in this course?

In this question, 8 students wanted to learn more about programming a Web API, although many of them mentioned that they knew they were going to learn about that in later courses (and most of them did not have the opportunity to finish that course beforehand), 1 student mentioned learning more in JavaScript and Angular, 1 student mentioned learning more about unit tests, and 1 student wanted to learn more how to read code.

Students were generally happy with the preparation they received in Web Programming II, even though 3 of them said that they would have wanted more time to properly do the assignments in that course. The only other course that was mentioned as good preparation was

Software Engineering (which covers practical aspects of software development). 7 students mentioned that the required prerequisite courses were a good preparation for this course.

4.3.2 Question 2

What made the most difference in the course, i.e., what helped you the most in learning what you did learn?

Note: it is quite possible that this question could be worded a bit differently. At least 2 students misunderstood it.[1] Students mentioned the following:

- Asking the instructor (6)
- Asking other students (5)
- Reading code from other people (4)
- Pair programming (3)
- Patience of the instructor (2)
- Google (2)
- Prerequisite courses (2)
- StackOverflow[2] (1)
- Code review comments (1)
- Suggestions from ReSharper[3] (1)
- Good management(1)
- Getting the time needed to complete a task (1)
- Tackling new project/problems(1)
- Getting issues assigned (1)

4.3.3 Question 3

Name 1-3 things you liked about the course and should be kept unchanged

Students named the following:

- Pair programming (9)
- Cooperation between the instructor and the students (7)
- Having issues to work on (5)

[1] One student explicitly stated that he did not understand the question, but then mentioned the curriculum in the prerequisite courses as something that helped him. Another student seemed to think we were asking how this course was different from other courses, and talked about the experience of entering a large project.

[2] StackOverflow is a website for developers, where they can post questions, other developers will answer, and the best answers are upvoted.

[3] ReSharper is a software from JetBrains, it is a plugin in Visual Studio, and can be used to enforce coding guidelines for projects such as the C# Web API project in Centris.

- Having more creative freedom (4)
- Cooperation between students in the group (2)
- Management of the project (2)
- "The glass cage" (2)
- Stress-free atmosphere (2)
- The freedom to choose who to work with (1)
- Code reviews (1)
- Interesting project (1)
- Feeling of success when tasks are completed successfully (1)
- Keeping diaries (1)
- Assisting 1st year students (1)

4.3.4 Question 4

Name 1-3 things you didn't like about the course and should be improved

Students named the following:

- Having the group in two places - the entire group should be located in the same room (2)
- Assessment of the course (but accepting that it will be hard to find a better course assessment)(2)
- Assisting 1st year students - not enough preparation (2)
- Clearer documentation (but acknowledging that asking the instructor was usually adequate) (1)
- Tasks were not always clear enough (1)
- Tasks could have been more detailed, i.e. having more smaller and focused tasks (1)
- Not enough tasks to work on (1)
- Having to compare to other students in the surveys (1)
- Feedback on code in pull requests (1)
- No daily (or weekly?) meetings (1)
- Instructor not always available (1)
- Nothing (6)

These results seem to indicate that students were generally happy with the course. There are 3 items which were mentioned by more than one student. The issue with the location was addressed in the next iteration of the course.

Learning outcome		Pre	Post	Hindsight
LO1	Read code, written by others, ...	4.58	5.72	4.44 (-0.14)
LO2	Improve code and documenta ...	4.12	5.33	4.17 (+0.05)
LO3	Apply the various standards and ...	4.60	5.56	4.06 (-0.54)
LO4	Apply best practices in version ...	4.46	5.39	4.17 (-0.29)
LO5	Address issues raised by code re ...	4.67	5.44	4.06 (-0.61)
LO6	Operate in a group, which is it ...	5.04	5.61	4.44 (-0.60)
LO7	Present (either orally or in writ ...	4.44	4.67	3.89 (-0.55)
LO8	Identify which parts of a project ...	4.40	5.11	4.11 (-0.29)
LO9	Identify which parts of a project ...	4.16	5.06	4.06 (-0.10)

Table 4.6: How students rated themselves in each learning outcome, before and after the course. The difference between the average rating before the course and the average rating in hindsight is shown within the parenthesis in the last column.

4.3.5 Question 5

Students were asked how they would rate themselves in each learning outcome, compared to their fellow students. This question was asked both in the survey they answered before the course started, and after they had completed the course. In the post survey, they were also asked to rate themselves in hindsight, i.e. to assess how qualified they believed themselves to have been before the course started.

In all cases, students could select one of the following:

1. Well below average
2. Below average
3. Slightly below average
4. Average
5. Slightly above average
6. Above average
7. Well above average

Results were calculated such that "Well below average" was assigned the value 1, "Well above average" was assigned the value 7, and the options below were ordered in between. This gave an average number for each learning outcome, as seen in Table 4.6.

There are a number of interesting points in these results. First, students generally rated themselves in the "Average" category for most learning outcomes. However, **LO6** stands a bit out, since students rated themselves as "Slightly above average". This may have to do with the fact how group work is dominant at Reykjavik University, but that would not explain these results alone, since students were probably using their fellow students for comparison. This particular fact was brought up in the focus group (see below), but the participants in that group did not have any clear explanation for this.

Another interesting fact is that the grade they gave themselves in hindsight was generally lower than the grade they gave themselves before the course started.

4.4 Diaries

The student answers in the diaries were usually short, i.e. 1–2 sentences on a given day. The students described there what they were working on on a given day. There are therefore no numerical results to display for the diaries.

In general, the diary entries stated which tasks the students were working on during the day. Only occasionally did the students note anything unusual, such as what helped them or what annoyed them.

4.5 Observations

The instructor observed that the students were not very active in requesting external help, such as by asking questions on stackoverflow.com, or by posting issues for libraries the students were using on their Github pages. Closely related to this was that students tended to get stuck on some problems. Many of the students seemed afraid to ask for assistance, and seemed to assume they were supposed to solve the problems individually. Both of these topics were discussed further in the focus group.

Another observation was that the atmosphere in the classroom with the more classical setup seemed less relaxed than in the other room, and there seemed to be less discussions in that classroom.

Finally, it was pretty obvious that some students were struggling more than others. In general, students who entered the course with a lower average grade and/or lower grades in the prerequisite courses, seemed to have more trouble understanding how to solve their tasks. When those students were paired with students with a better preparation, they were more likely to be able to get help from that student.

4.6 Focus Group

The following are the main points raised in the focus group:

- When asked what they learned first and foremost, they mentioned:
  - Working in a large group on a large project
  - Having to study code written by others

  This corresponds to learning outcomes **LO1** and **LO6**.
- When asked about if they were happy with the performance of the group, they seemed very happy with how much the group achieved. They also mentioned that this helped them when they got stuck themselves, then at least they did see that others were making progress and therefore the project itself was not stuck.
- Having the working area set up like a classic school room is not ideal, since it does not encourage teamwork. Students preferred one of the following:
  - Round tables, with 5-6 at a single table.
  - Opposing tables, with maybe 4-5 students in a row, and another row facing them.
  - Tables set up in U-shaped formation.

The reason given by the students was that this resulted in more interaction between them, which the traditional classroom setup did not encourage.

- There was quite some time spent on discussing how active students are in getting help when they get stuck with some problem. Students are not taught to ask questions online (such as on stackoverflow.com, in Github projects, etc.) and are even encouraged **not** to ask questions online. This makes them afraid that they are plagiarizing by using solutions/answers given to them online. Students should be taught how to phrase their questions, or how to do "rubberduck debugging".[4] This became obvious both for the group itself, and also for the 1st year students they were assisting in another course. Often, students seem to get stuck in some problem, and they seem afraid to ask for help, perhaps because they are not sure what the problem really is. In any case, this must be addressed.
- There were mixed emotions regarding the questionnaires and the diaries. Some students said that forgetting to log each day in the diary was causing them discomfort after a few days, when they had to think backwards about what they were doing. Others said that it was a nice way to complete each day, to write down what they were doing, thereby getting a better overview over their progress.
- In general, students were very happy with the course, and said that they believed it was a good preparation for them.

[4] When programmers explain their problem to an arbitrary object, often a rubber duck, which often results in them figuring out the solution to their problem, without ever asking another person.

Chapter 5

Discussion

In this chapter, we discuss various questions and themes which occurred in the course, both during data analysis in spring 2015, and during other iterations as well. These questions and themes are either those we find the most interesting, or those that we feel are most relevant when discussing how to prepare students for a career in software maintenance.

5.1 Why do Students Enroll in the Course?

It seems pretty clear that the students which enrolled in the course did so to gain more experience in software development. This does not come as a surprise, given the fact that the majority of students which graduate from the undergraduate program in Reykjavik University enter the computer industry soon after graduation.

As can be seen in question 1 of the pre-course survey (Table 4.1), most of the students entering the course had little or no background in software development, and the main reason why they enrolled was to gain more experience. This was further emphasized by question 5 in the pre-course survey (Table 4.5), where all the students which answered the survey reported that they felt they needed more experience in software development.

5.2 Are Students Properly Prepared?

The results seem to indicate that students are properly prepared. There is a small issue with the fact that the backend of the project is a Web API, and the optimal preparation for it would be to complete the course “Web Services”. However, students are currently not able to enroll in that course until in the 5th semester of their studies. This is due to the fact that this is an elective course which is taught in the fall, and there is simply no room in their study plan for elective courses in their first and third semesters. Effectively, this means that students have a “window of opportunity” at the end of the 5th and 6th semesters to enroll in RU Internship, if they want to have any involvement in the development and maintenance of the backend. In practice, students often enroll in the course at the end of the fourth semester, having completed the “Web Programming II” course, which prepares them for development on the client platform, and there have been plenty of opportunities for students to tackle tasks limited to that part of the system. Also, students have gained certain preparation for backend programming in the course “Web Programming I”, which they complete in their second semester, and many students are capable of using that experience to tackle backend tasks.

It should also be mentioned that while the learning outcomes for the course never explicitly mention the type of software project which the course uses, the prerequisites are specifically tailored towards a web development project. If the course would switch to a different type of project, such as a game development project, the prerequisites would need to be altered accordingly.

5.3 What Should Remain Unchanged in the Course?

We believe that the course has multiple positive points, which should remain unchanged. There were some points which were brought up in the questionnaires which we would like to cover here.

The item which most students mentioned was "Pair programming". This was not surprising to us, as prior observations indicated that this arrangement provided the best results.

Students like having time to really focus on a given feature, and spend as much time on it as necessary. This is in contrast with what they experience during "regular" courses, where the time pressure only allows them a certain amount of time for a given assignment, and they frequently experience having to hand in their solution at a specified point in time, even if they feel they could do better given more time. It can be assumed that both scenarios will occur in industry, so it is not unthinkable that they should also be given the opportunity to experience both in their studies.

Students like having some freedom in implementing features, i.e., where they must figure out themselves how to implement it, as well as experimenting with different possibilities during implementation. Certain projects and assignments in other courses definitely contain this element as well, while others are very limited and bound to a specific technique (for instance, there is limited variability in Quicksort implementations).

5.4 How can the Course be Improved?

The results of the data collection also indicate, however, that there are a number of steps that could be taken to improve the course.

The assessment of the course could be improved. This was mentioned by a few students, both in the post–course survey and also in the focus group. However, it is not entirely clear what the new assessment should look like. The tasks assigned to students are by nature very different, they may emphasize different aspects of the maintenance cycle, and the results may be different in each task. Using metrics such as Lines Of Code (LOC) would be one option, but that approach has its drawbacks. For example, a vital bugfix could be as trivial as editing a single line, while an implementation of a feature could mean adding hundreds of lines of code to the project. However, the former could very well be much more valuable, both for the project and to the development and experience of the student which found and fixed the bug. LOC is therefore not an optimal way to measure progress in learning, and the same can be said about other quantifiable results, such as the number of pull requests, number of commits in the code repository, etc. This does not mean that no such quantifiable results could be used as a basis for the course assessment. We would recommend that the search for a better assessment be continued, possibly by combining quantifiable results with an evaluation from the instructor and/or peers. It could be possible for the instructor to spend time analyzing the changes proposed by each student, and giving them a grade according to

that analysis. It would be interesting to do such an experiment, but then steps must be taken to ensure that the instructor has sufficient time to perform such analysis.

The students were allowed to request to be paired together. The results seem to indicate that this should not be allowed if both students have low average grade, since neither student will be able to help the other, at least if we want the groups to succeed and to produce code. If we have to choose between two groups, one with two students with high average grades and one with low average grades, or two groups were there is one student with high average grades and one with low average grades, then the latter seems to give us students which learn more, while the overall effect on the number of lines of code produced has not been measured. Finally, the results from the last survey indicate that many students like being paired with another student, since 9 students mentioned this as something that should be kept unchanged in the course.

The learning outcomes were rewritten in 2015, and we believe that they represent very well what the course expects of the students. There is a possibility that there should be a learning outcome which would emphasize the student abilities to express themselves when presenting a problem with their code, and their ability to get an appropriate help from others via various channels. However, this is probably something that should also be addressed in other courses, as it should not be only present in an elective course like RU Internship. We believe that this is an essential skill in software maintenance, and as such should be addressed in one of the core courses. One thing might influence this, and that is the fact that plagiarization is correctly frowned upon in universities, but asking for help on a particular detail which blocks the student from progressing with their solution should not be confused with asking someone to solve a particular problem, or taking someone's solution and presenting it as their own. Teaching students the difference between those two should be a part of the curriculum, as well as teaching students how to identify problems, and where to seek assistance.

Other changes in the learning outcomes might be helpful. **LO1** talks about code and documentation at the same time. This could be split up into two separate learning outcomes.

A course like this one should definitely just use a single location/classroom, and should not spread the students to more than one location. This was addressed in iterations after spring 2015, and should be maintained like that. Also, steps should be taken to ensure that the atmosphere in the classroom is not as formal as in regular courses, and that the setup of the classroom should encourage cooperation.

It was pretty clear that the presence of an instructor was probably what helped students the most, and steps should be taken to ensure that an instructor is available who is not occupied teaching different courses.

Steps should be taken to ensure that the project has adequate documentation, and that features have written specifications. Again, this means that the instructor needs to have time to prepare those things, either by doing it by him/herself, or by delegating those tasks to students. We would like to investigate further the use of videos to accomplish this, where individual concepts in the project is explained in a single video.

Finally, there is a major benefit in having the project actively used, such that students are supplied with real–life issues raised by users. As seen in Chapter 6, software maintenance is largely focused on issues and tasks which emerge after delivery. In our work, this could be simulated somewhat, but that will never replace real usage.

5.5 Learning Outcomes

In this section, we will discuss the results with a focus on the learning outcomes.

As can be seen in the results of the weekly surveys (Figure 4.1), students generally reported improvement in the learning outcomes. There are a number of notable results which are worth mentioning:

- There were 3 learning outcomes; **LO1**, **LO5** and **LO6** which saw a decrease in the number of students which replied "I feel like I have mastered this topic" between weeks 2 and 3. The difference was small in all cases. One possible explanation is that students got "cocky" in week 2, but then realized in week 3 that they probably could learn a lot more.
- Learning outcome **LO3** was the only learning outcome where the option "Did not apply this week" got more responses in week 2 than in week 1. We do not offer any explanation for this.
- Two learning outcomes stand out in terms of the number of students which seem confident in their abilities at the end of the course by selecting options d) or e), i.e. **LO5** and **LO6**. In both cases, close to 90% of the students selected either option d) or e). The discussions in the focus group reflected this as well, since students did mention both these as something they had learned in the course.
- Learning outcome **LO8** was noticeable least relevant. In all 3 weeks, almost half of the students either said that this learning outcome was not relevant, or that they did not improve at all. This is probably because the emphasis on this in the course was minimal, there was only one presentation in the course at the end, when some of the students had already answered the survey. Given these results, this learning outcome needs some revisiting, either by changing/removing it, or by changing the setup of the course.

Figure 4.2 shows how students perceived the improvements in the quality of the codebase. It is interesting that the biggest improvement was recorded in the second week, but not in week 3. Quotes from students in the focus group suggest that they were spending more time in week 3 doing cleanup tasks, which may not have an immediate effect on the perceived quality of the project, since not as many features were being added at that time, instead existing features were being refined. It is also possible that the change in quality/quantity was less obvious in week 3, due to the fact that students were in many cases starting from scratch in week 1. Finally, it is possible that the students better understood the size of the project and how much was still unfinished.

5.6 Improving Data Collection

After having completed the data collection, and gathered the results, there were a number of points that in hindsight would have been useful to address, either new questions which could have been asked, or questions which could have been asked differently. We did ask a few of them informally on the Facebook group in spring 2017, at that time there were approximately 150 students still members in the group.

Students were asked beforehand what learning outcome(s) they thought would be emphasized the most, and what they thought was most exciting. It would have been good to

ask again at the end of the course “What was really emphasized most?”, as well as asking if they thought any differently about what they found to be exciting, and note the difference.

Diaries could be more focused. Students should explicitly state what their “feeling” about each day was (was it a success? was it a failure?), if something was helpful, and if so then what, etc. In practice, the diaries turned out to be more like a list of tasks which the students completed, with not much insight into how they were completed, what was stopping them, or what was helpful.

In the weekly questionnaires, when asking about improvement in a given learning outcome, it should be clearer what the reference is, whether students should only talk about the difference in the given week, or whether they should be using the entire course. It was mentioned in some of the options that they should use just a single week, but not all of them. Also, the students should have received a link to the questionnaires in weeks 2 and 3 at the end of each week. It may have impacted the results for week 2, where the students were already 3 days into week 3 when they completed the survey for week 2.

It would have been preferable to analyse the results from the questionnaires sooner, such that those analysis could have been used more to bring up certain topics in the focus group.

It would be interesting to know whether students feel differently working on a project which is just a regular school assignment, where the results will effectively be thrown away when it is completed, compared to working on a project that might go into production, or is already being used by users. This question was asked informally on the Facebook group for the course in May 2017, 10 students opted to answer the question, and those answers suggest that 8 out of 10 students would say that there is a difference, where a project which has the possibility of being put into production will motivate them more. This is something that should definitely be studied further.

The question “Does this course offer a good preparation for software maintenance?” has not been asked in this thesis. We would certainly like to research this further, and we propose that this question should be asked after students have spent some time in industry, when they have gained experience and can reflect on how it compares to their experience from the course. In attempts to answer this question, further steps could be taken. We could use the Kirkpatrick Model [4] and add other measuring tools to address this question.

In general, we would definitely use the same data gathering methods again. We would modify the setup of the diaries, possibly by providing the students with an example of what a typical entry could look like. It would also be informative to ask them to give each day a grade, where the grade would depend on how much they felt they achieved that day.

Chapter 6

Related Work

Software maintenance is defined in the ISO standard 14764 [5] as "the totality of activities required to provide cost-effective support to a software system. Activities are performed during the pre-delivery stage as well as the post-delivery stage." The standard also defines the following four categories for maintenance tasks:

- Corrective, resolving issues from users and fixing bugs.
- Adaptive, adding new features to a product.
- Perfective, improving existing features in a product.
- Preventive, where steps are taken to prevent the possibility that a bug will be introduced to the product, usually by refactoring a given part.

Teaching software development by using hands–on courses is quite common. Marques and Ochoa [6] list several courses which give students hands–on experience by taking part in software projects. The list does not include the size of the project in each course in terms of number of lines of code, and does not explicitly mention which projects focus on software maintenance. However, they mention that the average team size in such projects is 4–6 students, and the data suggests that few projects include dozens of students or more, as we have done.

Teaching the construction and maintenance of a large-scale project is not very common. The ACM Standard Curricula [1] does not specifically address software maintenance; the concept is mentioned in 9 places, but no proper discussion takes place. The Guide to the Software Engineering Body of Knowledge (SWEBOK), on the other hand, devotes an entire chapter to software maintenance, and makes the following statement: [7, p.5–1] "Software maintenance is an integral part of a software life cycle. However, it has not received the same degree of attention that the other phases have."

A number of publications discuss software maintenance in courses. Petrenko *et al.* [8] describe a course where students were asked to improve one of four open source projects. The process they describe appears effective, but there is no mention of whether the changes made by students actually end up in the master branch of the corresponding project. Andrews and Lutfiyya [9] describe a course where students work on an existing project over two semesters. The experience from that course highlights the risk taken by using a project owned by a third party; in their case one of the project owners pulled their project due to internal company changes. They do not mention the size of the codebase of each of the projects, and each group contained at most 6 students, meaning that students did not experience working as a part of a larger team. Szabo [10] talks about the importance of teaching

software maintenance, and discusses a course where students perform maintenance tasks on an existing codebase. In that case, the course used a medium-sized codebase, around 11,000 LOC, which is interesting in itself, but does not address the problem of maintaining a larger codebase. The codebase in question is also not for a system in production, but instead a project written by students, and it does not seem to be intended for general use.

Reading code, written by others, is clearly necessary when maintaining existing code, or as it is stated in SWEBOK: "Programmers spend considerable time reading and understanding programs in order to implement changes" [7, p.5–10]. Tony Clear [3] discusses the importance of reading code a bit in the article "Comprehending Large Code Bases - The Skills Required for Working in a "Brown Fields" Environment", Clear states that it is hard to select a good project for students to work on. He also mentions the difficulty of understanding existing code, and states that it "is a challenging conceptual task, involving not only comprehension of the problem domain, but also how it has been addressed by the software artefact under study." [3, p.2] This is in agreement with our experience from previous courses, which is why we selected the domain of student and learning management for the project.

Abid *et al.* [11] discuss the importance of teaching students refactoring. Their main conclusion is that refactoring is more effective once students have become familiar with both the code and the problem domain. This also agrees with our experience, and underlines the importance of the ability to read code and understand what it does.

Measuring the outcome of training is an important topic. Galloway [4] describes the Kirkpatrick model for evaluating learning. This model defines 4 metrics to measure how successful training is: reaction, learning, behaviour and results. In this thesis the focus is on measuring learning, while behaviour and results are observed by the course instructor.

Chapter 7

Conclusions

Software maintenance typically represents 60% of software development effort, yet is mostly ignored in computer science education. This thesis has described an experiment, conducted at Reykjavik University, of developing a course to teach software maintenance. The results show that there are certain aspects which work well in a course like this. Pair programming is an effective way to prevent students from getting stuck in solving their tasks, especially when at least one student has a relatively good background. Easy access to the instructor in the course or the main architect of the project is necessary, this will make it easier for them to understand various aspects of the code. Care must be taken to ensure that the environment encourages communication, both between students, and between instructor and students. The facilities can play a vital part in this. It is also important to ensure that students have sufficient background when they enroll in the course.

We believe that a course like RU Internship is a necessary part of the curricula in CS. Students want to learn more about software maintenance, and they want to get practical experience that will be useful for them in industry. We also believe that we have demonstrated, over the several years of running this course, that such a course can provide that experience to students. We would like to see further research on how to teach software maintenance, both in terms of data gathering before, during, and after the course, as well as collecting information several years after students complete their study.

Based on the experience from the software development itself, we believe that it is possible to build a large scale product in a course, although it does take time and require patience. Regardless of that, the participation in a large software development project can be an important part of the learning experience. Mistakes will be made along the way; this is unavoidable. But by sharing our experiences, as is done in this report, we can learn from the experiences of others to avoid some of these mistakes and progress towards better training of software maintenance.

Bibliography

[1] The Joint Task Force on Computing Curricula, *Computer Science Curricula 2013: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science*, `http://www.acm.org/education/CS2013-final-report.pdf`, Dec. 2013.

[2] R. L. Glass, *Facts and Fallacies of Software Engineering*. Addison-Wesley Professional, 2002.

[3] T. Clear, "Comprehending Large Code Bases - the Skills Required for Working in a "Brown Fields" Environment", *SIGCSE Bull.*, vol. 37, no. 2, pp. 12–14, Jun. 2005.

[4] D. L. Galloway, "Evaluating Distance Delivery and E-Learning: Is Kirkpatrick's Model Relevant?", *Performance Improvement*, vol. 44, no. 4, pp. 21–27, Apr. 2005.

[5] "Software Engineering — Software Life Cycle Processes — Maintenance", International Organization for Standardization, Geneva, CH, Standard, Sep. 2006, `https://www.iso.org/obp/ui/#iso:std:iso-iec:14764:ed-2:v1:en`.

[6] M. Marques and S. F. Ochoa, "Transferring Software Development Knowledge and Skills in the Academia: A Literature Review", Technical Report TR/DCC-2013-2, Computer Science Department, University of Chile, Tech. Rep., 2013.

[7] P. Bourque and R. E. Fairley, Eds., *Guide to the Software Engineering Body of Knowledge (SWEBOK): Version 3.0*, 3rd. Los Alamitos, CA, USA: IEEE Computer Society Press, 2014.

[8] M. Petrenko, D. Poshyvanyk, V. Rajlich, and J. Buchta, "Teaching Software Evolution in Open Source", *Computer*, vol. 40, no. 11, pp. 25–31, Nov. 2007.

[9] J. H. Andrews and H. L. Lutfiyya, "Experiences with a software maintenance project course", *IEEE Transactions on Education*, vol. 43, no. 4, pp. 383–388, Nov. 2000.

[10] C. Szabo, "Student Projects Are Not Throwaways: Teaching Practical Software Maintenance in a Software Engineering Course", in *Proceedings of the 45th ACM Technical Symposium on Computer Science Education (SIGCSE)*, Atlanta, Georgia, USA, 2014, pp. 55–60.

[11] S. Abid, H. Abdul Basit, and N. Arshad, "Reflections on Teaching Refactoring: A Tale of Two Projects", in *Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE)*, Vilnius, Lithuania, 2015, pp. 225–230.

School of Computer Science
Reykjavík University
Menntavegur 1
101 Reykjavík, Iceland
Tel. +354 599 6200
Fax +354 599 6201
www.ru.is