Aalto University

School of Science

Master's Programme in Information Networks

Anastasia Lipiäinen

A Web-Based System for Executing Interactive Scala Programming Exercises in Browser

Master's Thesis

Espoo, November 27, 2016

Supervisor:	Professor Lauri Malmi
Advisors:	Otto Seppälä D.Sc. (Tech)
	Juha Sorva D.Sc. (Tech)

Author:	Anastasia Lipiäinen		
Title:	A Web-Based System for Executing Interactive Scala Programming Exercises in Browser		
Date:	November 27, 2016	**Pages:**	vii + 120
Major:	Information Networks	**Code:**	SCI3047
Supervisor:	Professor Lauri Malmi		
Advisors:	Otto Seppälä D.Sc. (Tech), Juha Sorva D.Sc. (Tech)		

When first introduced to programming, students are often assigned to write mere code snippets that only produce a numerical output with the help of a simple control structure. The students are left unimpressed as they fail to see the real utility that learning to program holds. By assigning students real-world programming problems, such as games and media computation exercises, we showcase real applications for programming.

At the beginning of their first programming course, students are already used to working with modern, interactive, and visually impressive interfaces. However, they cannot be expected to produce the graphical user interfaces required to take the full advantage of media content. Previously, we have distributed interfaces implemented with Swing as program code for local execution; the practise does not allow the easy updating of assignment related code and achieving a usable interface is tiresome with the use of Swing.

To address the issues of the current process, we developed a web-based system that allows the execution of interactive Scala programs in a browser. The system combines student's code together with the instructor prepared code and compiles it to JavaScript code. The design of the system is a novel one and takes the use of modern web technologies. The system was trialled with a media programming course in 2015 with encouraging results. Despite the technical difficulties, the system helped addressing the challenges encountered in implementing and distributing modern graphical user interfaces.

Keywords:	distributed homework submission system, novice programmers, computer science education, Scala		
Language:	English		

Tekijä:	Anastasia Lipiäinen		
Työn nimi:	Web-pohjainen järjestelmä interaktiivisten Scala-ohjelmointitehtävien ajamiseksi selaimessa		
Päiväys:	27. marraskuuta 2016	**Sivumäärä:**	vii + 120
Pääaine:	Information Networks	**Koodi:**	SCI3047
Valvoja:	Professori Lauri Malmi		
Ohjaajat:	Tekniikan tohtori Otto Seppälä, Tekniikan tohtori Juha Sorva		

Usein ohjelmointiopintojen alussa opiskelijoille annetaan tehtäväksi vain lyhyitä koodin pätkiä, jotka tuottavat kontrollirakenteen avulla numeerisen tulosteen. Tällaiset tehtävät eivät vakuuta opiskelijoita ohjelmoinnin opettelun hyödyllisyydestä. Antamalla opiskelijoille tehtäväksi todellisia ohjelmointiongelmia, kuten pelejä tai media-ohjelmointitehtäviä, on mahdollista esittää ohjelmoinnin todellisia sovelluskohteita.

Jo ennen ensimmäistä ohjelmointikurssinsa alkua opiskelijat ovat tottuneita interaktiivisten ja visuaalisesti näyttävien käyttöliittymien käyttäjiä. Heidän ei kuitenkaan voida odottaa itse tuottavan käyttöliittymiä, jotka hyödyntävät media-sisältöjä. Aikaisemmin olemme tarjonneet käyttöliittymät Swing-kirjaston avulla toteutettuna ohjelmakoodina, joita opiskelijat ovat suorittaneet omalla tietokoneillaan. Tämä ei kuitenkaan mahdollista helppoa tapaa päivittää tehtävänantoon liittyvää koodia, minkä lisäksi onnistuneiden käyttöliittymien aikaansaaminen Swing-kirjaston avulla on työlästä.

Ratkaisuksi näihin ongelmiin diplomityössä kehitettiin web-pohjainen järjestelmä, joka mahdollistaa interaktiivisten Scala-ohjelmien suorituksen selaimessa. Järjestelmä yhdistää opiskelijan tuottaman koodin opettajan valmisteleman koodiin ja kääntää nämä yhdessä JavaScript-ohjelmaksi. Järjestelmän toimintatapa on uudenlainen ja sen toteutuksessa on käytetty moderneja web-teknologioita. Järjestelmän koekäytössä mediaohjelmoinnin kurssilla vuonna 2015 saatiin lupaavia tuloksia. Teknisistä vaikeuksista huolimatta järjestelmä auttoi kohtaamaan modernien käyttöliittymien toteutukseen ja jakeluun liittyviä haasteita.

Asiasanat:	hajautettu palautusjärjestelmä, ohjelmoinnin aloittelijat, tietotekniikan opetus, Scala
Kieli:	Englanti

Acknowledgements

Contents

Part I

Introduction and Background

Chapter 1

Introduction

One of the learning objectives in the introductory programming courses taught during the first semester at Aalto University is that after completing the courses students see that programming is useful. The most powerful way to showcase the utility of programming is to provide students with exercises that produce real-world usable programs with impressive user interfaces. Furthermore, the use of graphics has been recognised to be motivating in the computer science education literature. Nevertheless, the difficulty and the workload of the exercises should be contained within appropriate limits so that less proficient students will not struggle too much.

Students are already familiar interacting with modern graphical user interfaces (GUI) at the beginning of their first programming course. However, graphics and GUI libraries are often not suitable for the novice programmer and may be difficult for even the experienced programmer to master. Introducing user interface programming at the beginning of the first programming course is not necessarily desirable although students should be kept motivated from the very beginning.

Previously, we have provided students with GUIs as code that should be executed as a part of their solution. Students need to download potentially several source files and place them correctly within their own code project before they can experiment with their solution. Furthermore, the GUI source code is quite likely indecipherable for most students. An effort to try and

understand the code may take up valuable time that could have been spent concentrating on the actual learning objectives of the assignment. Additionally, if some mistake has slipped through to the source code, it is really hard for students to identify let alone correct and an update distributed later on might not reach all students.

Locally run GUIs have been implemented using Scala Swing which is powerful albeit tiresome to use. Compared with the user interfaces built for the web using HTML, CSS and JavaScript, it is much more difficult to achieve a modern and attractive GUI with Swing. Swing's event handling and nested components are quite similar to JavaScript's events and HTML's nested elements defined with tags. However, Swing's components need to be individually adjusted to achieve the desired visual outcome while HTML elements are easily modified with style sheet files.

1.1   Objectives

To try and address the issues described in the previous section, we attempt to offer for students a system that would host GUIs that are more attractive, easier to build, update, distribute and execute. The objective of this thesis is to design, build, and have a trial run of an interactive web-based tool that provides CS1 students with instructor-prepared graphical user interfaces that combined with their Scala program assignments allow trial executions online.

The tool, later, in order to facilitate referring to it, named EDCAT (short for Execution Doesn't Count As Testing), should provide an interface for uploading the logic code produced by students and return a GUI which uses that logic. The GUIs should be at least as easy to build and distribute as the ones build with Swing and distributed as zip-packages. The teaching staff will have access to all submissions and possibility to submit their own solutions. Finally, EDCAT should support debugging practices at least just as well as a locally executed GUIs similar to those that have been used previously.

Hosting the GUI code on a server allows us to hide it from students so that they can concentrate on the assignments and not on understanding code that is too complex for their level. Similarly, web-based implementation allows us

more easily to create better looking interactive GUIs. Furthermore, because the GUI code is hosted on a server for compilation, students do not need to download it. Thus, is is easy to update the instructor prepared GUI code when needed so that the update immediately reaches every student.

Figure 1.1 summarises the impact of the outlined system on the distribution of GUI code and program execution. From the teacher's point of view the change is minimal; instead of uploading GUI code to a server that students can directly access, she uploads the code to another server that will be in
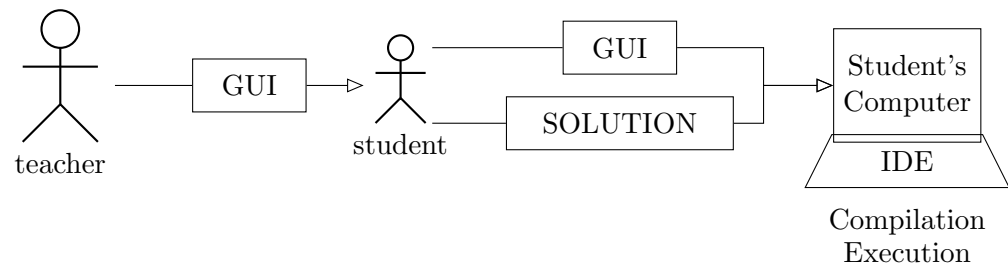
(a) Before
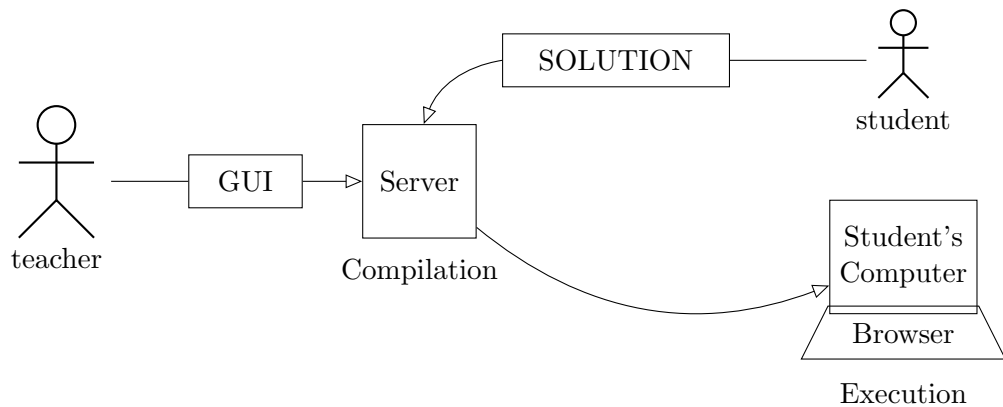
(b) After

Figure 1.1: With EDCAT, the GUI code is available on a server, where it is compiled together with student's solution and returned as a JavaScript program that can be ran in browser.

charge of compiling students' solutions. Students on the other hand need to download and upload less code. Additionally, the program will be executed in browser instead of launching it from the IDE.

Forcing the program execution online provides an opportunity for the teacher to follow how students iterate their solution towards a fully functional program. Additionally, online execution can help with remote tutoring practices as teaching assistants can access students' unfinished submissions easily. Similarly, there is no necessity to download students' solutions for test runs or grading, when the exercises are graded by teaching assistants.

1.2 Structure of the Thesis

This thesis is organised into four parts. The first part introduces the problem scope in Chapter 2 and presents relevant research from the fields of computer science education and web development in Chapters 3 and 4 respectively. The second part presents the solution with Chapter 5 that outlines system goals and restrictions and provides a description of the implemented system and Chapter 6 that describes the technical solution and presents the completed system. The third part is concentrated on evaluating the solution. Students' opinions were accessed through a questionnaire, some additional interviews as well as feedback forms, which are all covered in Chapter 7. Teaching assistants were given a separate questionnaire, which is discussed in Chapter 8. Finally, Chapter 9 covers usage data extracted from server logs and the course management system. The fourth and the last part returns to the initial problem and considers the limitations of the solution in Chapter 10 as well as summarises the findings and suggests further research areas based on this work in Chapter 11.

Chapter 2

Research Context

At Aalto University School of Science there are four computer science courses that use Scala programming language; *Programming 1*, *Programming Studio 1: Media Programming*, *Programming 2* and *Programming Studio 2: Project*. The two courses mentioned first are offered in the autumn semester and the latter two in the spring semester. The students are expected to take the courses during the same academic year, usually their first. The courses were developed as a part of a degree reform and have been offered starting from the study year 2013 - 2014.

The School of Science has four degree programmes; Computer Science and Engineering, Information Networks, Industrial Engineering and Management, and Engineering Physics and Mathematics. Students in the degree programme for Computer Science and Engineering are obliged to take all of the courses and students in the degree programme for Information Networks all but *Programming Studio 2: Project*, while for the rest only the *Programming 1* is a mandatory part of bachelor level studies.

Clearly the biggest of the courses in respect of the number of students attending the course is *Programming 1* with three hundred students from the School of Science. Since 2014 and the rolling out a MOOC version of the course, there have also been a few hundred students from other schools in Aalto University and some thousand people outside Aalto University taking the course. Other courses clearly have less students; *Programming 2* has

around three hundred students annually, while *Programming Studio 1: Media Programming* and *Programming Studio 2: Project* have around 150 students annually.

Programming 1 introduces the key concepts of programming using imperative and object oriented programming paradigms. The course consists of mainly practical assignments that students can choose to complete alone or in pairs. There are no prerequisite skill requirements for the course and it is often the first programming course for the students. After the course, students should find programming to be useful and be able to discuss the key concepts of programming and the object-oriented programming paradigm. They should be able to apply those concepts to well-defined problems with the use of appropriate programming tools. Students will have an idea of how a computer works and be able to write code in a good style as well as read and use code and documentation written by others.

Concurrently alongside *Programming 1*, students are presumed to complete *Programming Studio 1: Media Programming*, which covers multimedia programming and applies problem based learning in programming context. In *Programming Studio 1*, skills learned in *Programming 1* course are fortified with applied problems from the media programming field. After the course, the students should understand the basic concepts of digital media and the related programming principles. The students are able to write computer programs that deal with digital media in various ways. Special focus is placed on the playback, compression and filtering of natural signals, such as images and sound.

In *Programming 2* students focus on understanding computer architecture and program execution while the course *Programming Studio 2: Project*, as its name suggests, focuses on completing an individual programming project from planning to execution and documentation. At this time, EDCAT is intended to be used primarily in *Programming 1* and *Programming Studio 1: Media Programming* courses while the trial run will be carried out with only the *Programming Studio 1* course. By designing a system that would suit the needs of both of these courses, we hope to reach a general enough solution to meet the needs of any programming course.

2.1   Student Motivation and Technical Skills

Programming 1 is mandatory for all students in the School of Science, which means that the students' technical skills and motivation to learn programming can vary significantly. The students of the Industrial Engineering and Management degree programme for example must take only one other computer science course, which covers the basics of database design and queries. Completing that course does not require much programming even though *Programming 1* is mentioned as a prerequisite for the course. Overall, the studies in the degree programme of Industrial Engineering and Management are not really technical, so for some, the computer science courses are just necessary evil and motivation to spend time on them is minimal.

At the same time, some students of Computer Science and Engineering are highly motivated in performing especially well in computer science courses. They have chosen to focus on computer science and may have previous programming experience through hobbies or work. On the other hand, some students end up studying Computer Science and Engineering after the failed admittance to their primary discipline or as a temporary solution while they decide what they really want to study. However, recent legislative changes in the student selection process might reduce the amount of unmotivated students. From fall 2016 universities are obligated to reserve a quota of study places for persons applying for the first time to a university. Thus, unsure students will less likely apply (and be admitted), because it would considerably weaken their chances to get into a desired university in the future should they decide to change their study programme.

In addition to degree students, *Programming 1* is offered also as a MOOC. It is primarily aimed at the high school students, teachers and students of Aalto outside the School of Science although anyone interested can participate.

To conclude, students in the mentioned introductory programming courses are far from a homogenous group both in motivation and technical skills. The provided tools should be easy to use so that students with lower motivation do not struggle further and highly motivated students do not loose interest because of a troublesome environment.

2.2   Assignments

Programming 1 has approximately ten assignment rounds each consisting from three to five smaller exercises. The exercises can be small programming problems but there are also multiple-choice questions related to the study material. The exercises are submitted to a grading system that immediately provides an automated assessment for the submission. *Programming Studio 1*, on the other hand, has approximately only five programming assignments that result in programs with a specific function, like image filtering tool or Battleship game. All assignments are graded by teaching assistants; verbal feedback and grading are provided about one to one and a half weeks after the submission.

The single greatest limitation set by *Programming 1* to EDCAT is derived from the sheer number of students. EDCAT should be able to process multiple submissions fast enough so that running the code online is as fast as with a locally executed GUI. Additionally, even after eliminating the multiple-choice questions, another limiting factor is the number of individual exercises. Although some reuse could be possible between the exercises, it should be effortless and simple enough to create new GUIs.

The topics of the assignments make up a specific challenge in *Programming Studio 1* course. At least simulating sound and image manipulation should be possible. It should also be thought through, how closed the system should be; in the second year of running the *Programming Studio 1* course one assignment round was dedicated to the use of web resources for which students completed RSS aggregators with positive results. Furthermore, students begin to practice user interface design in *Programming Studio 1*, thus later on an implementation of a simple graphical API for their use would be extremely beneficial. Either way, EDCAT should support the implementation of complex interfaces with several different interaction elements.

2.3 Teaching Environments and Tutoring

In addition to student information and course management systems, computer science courses often have special requirements and employ their own virtual learning environments. *Programming 1* uses an older system called Goblin[1] [17] that is build upon the traditional LAMP (Linux, Apache, MySQL, PHP) stack with an integrated automated assessment tool. The exercise instructions are provided embedded in the course's interactive study material that contains in addition to the multiple-choice problems some simulations and animations. The students should first read the theoretical introduction to the subject and try out the provided example code before they move onto completing and submitting the assignments.

In *Programming Studio 1* the assignments are distributed and submitted through a more recent system called A+ [2], which is implemented using a modern Python web framework Django. A+ is a service-oriented system and it is interoperable with other learning systems. For example, in 2014 the same automated assessment tool that is in use with Goblin was trialled in some of the assignment rounds but proved unsuitable for the nature and objectives of the course. Grading assignments by hand allows tutors to pay attention to code style and the intelligibility of the solutions.

Both courses implement similar tutoring practices; there are several weekly exercise sessions with one to three tutors, additionally help is offered through IRC channel and Piazza[3] discussion boards. Students do not need to register for exercise sessions but can attend them whenever needed or choose altogether not to use them. This results in busy sessions near the assignment deadline, where students might need to wait quite long to get help and of course right after the assignment deadline, the sessions are quite thin of students.

---

[1] `https://greengoblin.cs.hut.fi/`
[2] `https://plus.cs.hut.fi/`, `https://github.com/Aalto-LeTech/a-plus`
[3] `https://piazza.com/`

Chapter 3

Computer Science Education

Learning to program is not easy and related problems are quite manifold. Some of the issues can be tackled with the help of educational software while others are more related to course design. Du Boulay [8] separates the difficulties of learning to program into five partly overlapping areas; *orientation*, problems related to understanding *the notional machine*, *notation*, *structures*, and *pragmatics*.

Problems of *orientation* are related to recognising the advantages of learning to program and what can be achieved after learning the skill. Closely related are the issues of understanding the properties of a computer and how to control one programmatically. A mental model of program execution on a computer — functioning of *the notional machine* — should be formed at the right level of detail. *Notational* problems relate to not only semantics but also the syntax. *Structures*, such as loops or if-else statements, often prove difficult for novice programmers. Finally, mastering the *pragmatics* of programming, including specifying, developing, testing and debugging a program, requires mostly practise.

Currently, the dominant theory of learning is constructivism according to which students actively construct knowledge rather than passively absorb it from lectures and textbooks. Glasersfeld [36] points out that teaching is not about giving students the right answers but it is far more important to teach students *"to see why a particular conception or theory is considered*

scientifically viable in a given historical or practical context" [36]. Theories are considered viable in relation to their context and it is critical to realise that there will always be more than one way to solve a problem.

In order to apply constructivism to computer science education two domain specific characteristics should be taken into account. The beginning computer science students do not have viable conceptions about the computer similarly like the beginning physics students have naive theories about their field. Additionally, the computer provides immediate feedback about whether a student's solution was successful. [4, 5] The computer is not an easily accessible entity, whose inner functionality could be intuitively deduced. As we teach computer science with the help of hands on programming, the student will face in her first computer science course early on the inevitable conflict between her ineffective model of the computer and the behaviour of an actual computer. To some this feedback may be discouraging and contributes to the conception that programming is difficult.

However, Smith et al. [30] argue that misconceptions can have a productive role in acquiring more effective scientific conceptions and that expert reasoning is based on reused and refined prior and intuitive knowledge. The claims are based on the core of constructivism; new knowledge is created through accommodation and assimilation so that prior knowledge is either reasserted or remodelled. Smith et al. [30] conclude that knowledge acquisition is a gradual process and misconceptions can not be substituted directly for expert concepts.

By providing students with motivating opportunities to refine their conceptions and good examples on how those conceptions are relevant in practice, we hope to tackle the difficulty of orientation as well as ensure that students reach the course's learning objectives. Additionally, we hope that EDCAT could provide a simpler approach for the teacher to achieve set goals while easily managing the course and its exercises.

This chapter is organised into three sections. In Section 3.1, we concentrate on the motivation of novice programmers by discussing successful exercise designs and topics. Section 3.2 outlines the relevant technical tools available for supporting novice programmers in their quest for mastering the skills

required in development of computer programs. Finally, Section 3.3 sums up this chapter and analyses the presented research areas from the point of view of implementing EDCAT.

3.1 Designing Motivating Exercise Topics

The programming exercises should most importantly provide an opportunity to practice the basic programming structures covered by the course. Nevertheless, they should be interesting so that students are motivated to work on and finish them as well as possible. However, an appropriate level of difficulty should be maintained taking into account that most of the students have no prior programming experience.

Most of the approaches to motivate the students in CS courses presented in the literature fall usually in one of the two categories; the use of game assignments or the use of graphics or media computation assignments. Furthermore, Sung et al. [33], who themselves use games in computer graphics class, distinguish three separate types of courses that use games; *games development*, *games programming* and *games development client* courses. Because we are not specifically interested in game design and development, we will only consider literature that describes *games development client* courses that integrate games into introductory programming courses' curriculum by using games as programming assignments.

There are many successful examples of using games for teaching students the basic programming concepts in an introductory programming course [3, 16, 20, 22]. It is not really significant what kind of games are used, as long as they fit the course curricula. Although some games work better than others. Becker [3] substituted the assignments of Conway's Game of Life in a CS1 course and a drawing program in CS2 course with *Minesweeper* and *Asteroids!* respectively. She found that students were much more motivated than in previous years of the courses; they put in extra hours and continued to work on their games even after the submissions date. Using games already familiar to the students motivated students to outdo themselves while achieving the learning outcomes set for the course.

Furthermore, games can be used to introduce programming constructs that might otherwise prove too difficult for students to grasp. Hansen [16] successfully used the game of Set to familiarise students with polymorphism and design patters — topics usually seen too advanced for an introductory programming course. Similarly, Lewis [22] introduced design and implementation aspects of a larger program by using a full semester long game project. Furthermore, in the course described by Lewis, the students had the same briefing but were free to choose their own topic for the game within the given technical requirements. The technical implementation of Lewis' project framework will be further discussed in Section 3.2.

However, when choosing games for course material — or equally media computation — the teacher should consider the appropriate difficulty level. Jimenez-Peris et al. [20] have found that by providing more background materials, even the assignments that at first seem difficult can be suitable for entry level programmers. By providing students with interesting assignments at an appropriate difficulty level Jimenez-Peris et al. [20] have succeeded in introducing students a breadth of topics in computer science covered thoroughly in the later courses.

Regardless of the aforementioned successful uses of games in introductory programming courses, they do not provide a silver bullet. Bayliss and Strout [2] found that indeed not everyone is interested in games, and they should not be used on every course.

Use of media computation, or MediaComp, in those cases might prove more suitable; originally it was designed to appeal to non computer science majors but has proven also successful among computer science majors [29]. Simon et al. [29] found that students learned comparable skills to a traditional computer science course. Although the focus is on the more complex use of core constructs and not that much on class or program design. Additionally, the retention rate was significantly higher than in a traditional course.

Assignments in MediaComp appeal to the students' creative side and showcase how programming is used to solve the real-world computational problems of sound and image editing programs. Additionally, they result in intriguing programs that students enjoy working on and are proud of.

As mentioned, MediaComp assignments have proven especially successful with non computer science majors as Guzdial and Soloway proposed in their first article in 2002 [14] and Guzdial [12] later confirmed with the 2003 paper. Guzdial, based on these and his later works on the same subject, synthesises his findings into five hypothesis and analyses them against other research papers on MediaComp [13].

Guzdial proposes that MediaComp assignments (1) reduce incentive to plagiarise other students' work, (2) result in higher retention rates than on a traditional course, (3) are more interesting to female students, (4) result in as good learning outcomes as in a traditional course, and (5) inspire students to take more computer science courses. [13] Of these hypotheses the second, the fourth and the fifth are especially interesting as we wish all of our students to successfully pass the course having met the set learning goals and motivated to continue onto further computer science courses, including non-mandatory ones.

Guzdial found that indeed in MediaComp courses students' retention rates were higher compared with the corresponding traditional course both in Georgia Tech, his institution, as well as other institutions. The student interviews revealed that students found the MediaComp approach motivating, because the course was tailored, relevant and provided an opportunity to be creative. Furthermore, Guzdial found that the sense of relevance is not uniquely attributable to MediaComp context and can be achieved with other topics as well. [13] Although research has observed higher retention rates in MediaComp courses compared with the traditional programming courses [13, 29], there is no clear reason as to why the MediaComp approach is successful.

The learning outcome hypothesis resulted in inconclusive results in Guzdial's work; the results were contradictory in two consecutive years of the same courses. First it seemed that the learning outcomes were different after the first course between MediaComp students and other students, however the differences evened out after the second course which was the same for all students. [13] However, those results were not repeated after the change of the assessment method. Later the results indicated that MediaComp students

failed to reach similar learning outcomes than other students. However, they also had the least programming experience. The disparity between Guzdial's and Simon et al.'s [29] research can in part be explained by the relative differences in the compared courses as well as the assessment method.

However, even if MediaComp courses do not reach identical learning outcomes as the traditional CS1 courses, as long as the learning outcomes are comparable and the differences dissolve after the subsequent courses, there is still big advantage to choosing MediaComp over a non contextual approach as students will be more motivated to complete the course. Furthermore, as we are aware that a MediaComp course will inevitably nourish some skills over others, we can systematically focus on those areas that naturally get less attention by either modifying the assignments accordingly or by assigning assignments that solely concentrate on those issues.

Because there are students who are not necessarily motivated to learn to program in our course, Guzdial's fifth and final hypothesis is relevant in the context of this research. We want our Information Networks students to see Computer Science as a viable option for their minor subject. Guzdial found that less than 5% of the students in MediaComp course continued on to the following course, specifically aimed at them, and less than 10% of those students continued on to the computer science minor or changed to CS major [13]. The curriculum will most likely have no effect to the interests that students have before beginning their university studies.

In the light of Guzdial's findings, perhaps the most we can do is not to discourage our students who might be inclined towards computer science studies. Furthermore, by introducing the different areas of the computer science field early on, like Jimenez-Peris et al. [20] suggest, we could capture students' interest and motivate them to pursue computer science concentrating on their area of curiosity.

What is common to both the games and the MediaComp approaches, is that students are given some freedom implementing their solution and get to work on relevant problems. Students get to use their own judgement and creativity in defining the rules of game play or designing an image filter while working on a project that results in a usable program. These approaches might

indeed appeal better to non technically oriented students while other students might find them not serious on inauthentic. As our course is mandatory for students with varying technical inclinations, we should provide challenges to all skill levels.

3.2 Educational Software Tools

Many tools have been developed in order to help novice programmers. There are integrated development environments (IDE) designed specifically for students (e.g. BlueJ[1] or DrJava[2]) and web-based IDEs integrated with a submission system. There are tools that demonstrate and visualise code execution and those that analyse code and help to find elusive bugs. There are systems for grading and giving feedback on the exercises. However, there really are few systems that allow web-based program execution comparable to locally executable programs — the kind that would allow interaction and provide a graphical user interface (GUI).

While some program execution visualisation tools might be interesting from the technical point of view, they are outside the scope of this thesis. Even if they execute student written program code online and provide a visual context, they are primarily designed to aid students in code tracing and problems related to the notion of the notational machine. Similarly, we are not interested in web-based IDEs as such nor in automated assessment tools. However, some of them not only compile and run the program against test cases but also return the executable program for student's use and might allow user input during the execution process.

Section 3.2.1 reviews some of the web-based programming environments that allow the creation of interactive programs. In Section 3.2.2, we inspect some libraries that could enable the easy creation of GUIs or otherwise visual content. Closely related is the use of skeleton projects as a basis of programming assignments, which is shortly discussed in Section 3.2.3.

---

[1] `http://www.bluej.org`
[2] `http://www.drjava.org`

3.2.1 Web-Based Programming Environments

W4AP [35] is a programming and execution environment that students access via browser. The user can create and modify program files online or upload locally prepared files, which are then compiled into an executable application. W4AP also provides a monitoring system that tracks the execution of the compiled program in real time and visualises the program state. W4AP supports different kinds of projects and programming languages, including Java. To support multimedia projects the environment uses a specially designed Java library [35]. Because the produced program is an applet, there should be no obstacle in creating an interactive application.

Ng et al. [23] propose a similar system to W4AP where students can access programming activities defined by the teacher, edit files needed for submission inside an editor in browser and submit files for compilation on a server. The result of a successful compilation is an executable JAR file that students can use locally. The focus of the system described by Ng et al. [23] is in distance learning and tutoring. The unique contribution of the system is an interface for leaving a help request to the tutors.

Furthermore, Truong et al. [34] describe a very similar system that provides students with gap filling exercises in Java and returns executable programs. Unlike W4AP or Ng et al.'s systems, Truong et al.'s Environment for Learning to Program (ELP) does not have any interaction with the tutors or analysis of program execution. The motivation is to lower the barrier to begin to program and it also integrates the lecture materials and tutorials to help students to solve the exercises.

Unlike the solutions described above Hitz and Krögeler [18] outline a system that will accept student input prior to running the program remotely. The system supports fill in the gap type of C++ exercises that are embedded in the course material. Although input is supported, the interaction is nonexistent after the initial input values.

Perry's VECR (View/Edit/Compile/Run) [24] environment for C, Java, and Unix shell script programming exercises supports also exercises where students are required to create interactive GUIs with HTML forms. Addi-

tionally, the environment supports the creation of mathematical plots with a specialised library and the use of audio manipulation and output. The technical implementation uses Java applets that use only the simple functionality so that environment would work also with older versions of Java [24].

Yoo et al. [37] describe a solution for Scheme and Racket that can execute interactive REPL programs. WeScheme [37] is implemented with more modern technology and benefits from the latest capabilities of HTML. The compilation is performed on a remote server initially as well as during the execution of the program when input is given in REPL. This requires a constant connection to the server during the execution. WeScheme has a sharing functionality, that allows users linking to their projects anywhere from the Internet; the users can choose whether the source code will be visible or not.

Another modern implementation of a similar system for Python assignments is presented by Edwards et al. [9]. Pythy [9] implements a modern web interface with the underlying installation of `git` for the version management of students' work. It is implemented as a Ruby on Rails application and uses a modified version of Skulpt (introduced in Chapter 4). While the implementation supports media computation like image manipulation, the created problems rely on REPL for interaction.

3.2.2 Graphics and GUI Libraries

The most commonly used programming languages for CS1 courses (e.g. Python and Java) have powerful albeit complicated GUI libraries. Furthermore, these days students starting their first programming course are mostly familiar interacting with the computer through GUIs and working with command line can be seen as difficult. Many teachers have thus provided their students with easy to use but powerful ways of building GUIs.

RacketUI [15] enables automatic web form generation based on students' program code written in Racket. After importing the RacketUI library, students only need to create a `web-launch` macro and specify labels for the function and its parameters. Using RacketUI seems quite easy, but there is little control over how the resulting GUIs look.

BreezyGUI [21] is a toolkit for building realistic GUI applications with support for event-driven interaction. BreezyGUI is implemented on top of Java AWT and hides its intricacies from students. The ideology is still very much the same, so when students are ready they will probably have little trouble moving onto the real Java GUI libraries.

QuickDraw [31] offers a different approach with system and language independent multimedia application that works as a rendering engine. The functionalities provided by QuickDraw can be used by connecting an application via pipe to the QuickDraw application. QuickDraw provides both 2D and 3D graphics, audio playback and a text to speech engine which can be used through one sided pipe; Input capabilities require a bidirectional pipe. The student's program should print out QuickDraw commands and pipe the output to QuickDraw application. The syntax of QuickDraw is simple — command names are descriptive and the amount of needed parameters is kept small by separating style and primitive commands. [31]

A more straightforward solution has been introduced by Roberts, who has created simple graphics libraries first in C [27] and then in Java [26]. The graphics libraries are constructed especially for novice programmers to use in CS1 course. They provide the basic drawing functions and hide the potentially confusing event loop based paradigm employed by typical graphics libraries. Especially Roberts' Java graphics library hides much of the required boilerplate and provides a conceptually strong object-oriented approach to programming with graphics. [26, 27]

Bruce et al. [6] created a graphics library that simplifies the use of Java's graphics functionality. The aim was to use the object-oriented approach from the very beginning of the CS1 course and familiarise students with the use of objects and methods. The library defines a number of objects that are placed on a drawing canvas that is controlled by a class that is an extension of a Java applet; The complete programs are indeed event-driven Java applets.

Similarly, Shultz [28] proposes that the use of 3D graphics could be possible in CS1 and CS2 courses with the help of an appropriate simplified library. Learning 3D programming involves understanding complex concepts, which Shultz proposes concealing so that students can concentrate on relevant issues.

Compared with QuickDraw's approach, Shultz's proposed 3D graphics API is much closer to the APIs used by the industry.

3.2.3   Use of Skeleton Projects

The practice of providing the students with a project skeleton that they should supplement with the implementation of some classes or methods has been around already for some time. Grissom [10] summarises the discussion on teaching students I/O in Java. Depending on the approach students can be given a class that encapsulates the needed functionality or a program with a GUI that students need to modify. Grissom himself proposes supplying students with a package they can integrate into their program. The use of the teacher prepared code allows students not to worry about aspects that are not yet relevant in their learning process.

Reges' [25] use of skeleton projects is very similar to what we have been doing in our course. The projects distributed to students implements mostly just the GUI code while students are in charge of implementing the logic of the program. Reges' findings are also quite similar with our experiences — students like working with full programs although preparing the needed GUIs is time-consuming and students might sometimes misuse the teacher prepared codes.

Buck and Stucki [7] found similar difficulties as Reges while using skeleton projects to enhance the outcomes of student work. They have concentrated on the project design so that students do not need to work with GUI code directly and that the projects do not teach bad habits of program design. Astrachan and Reed [1] have similar aspirations by providing their students with complete applications that the students are asked to re-use and re-work. Students learn that well designed programs are easy to modify and get to work on full applications.

Sung et al. [32] have designed game themed assignments that contain material both for the students and the instructor of a course. The assignments are distributed to the students as skeleton projects which they complete by following step by step instructions to achieve a fully functional program.

The students get to concentrate on the core programming concepts while the graphical functionality is already implemented. The materials are in theory freely available [3], but unfortunately some resources are missing and the projects cannot be accessed.

As mentioned in Section 3.1, Lewis [22] uses a framework on top of which students implement their game project throughout the course. The students are provided with a JAR file containing the full framework without source files and first few class skeletons for their implementation. This is not a skeleton project in the same sense as students are given the supporting functionality as a compiled library; they get to learn how to use Java documentation and existing libraries.

The common finding in use of skeleton projects is that they provide a context of a real program. The instructor can show how a bigger program is build and allow students to concentrate on only the narrow learning objectives set for the week. By writing a package that encapsulates some difficult functionality or by implementing the GUI, the teacher helps the students to create programs above their skill level.

3.3 Summary and Conclusions

Both the games and the MediaComp approach are successful solutions when we want our students to work on motivating real-world problems. However, the use of these approaches often requires visually impressive programs that our students are not yet equipped to fully implement on their own. This leads to the use of simplified graphics and GUI libraries, that are manifold and readily available. However, they might be deemed inauthentic by some students. Thus, we must keep in mind in our exercise design that we include real challenges for all skill levels.

The advantage of using a simplified library is that students are allowed to concentrate on the relevant programming concepts instead of understanding the intricacies of the implementation of a windowing system or rendering

---

[3] `http://depts.washington.edu/cmmr/Research/XNA_Games/index.php`

context. The same advantage can be achieved with the use of skeleton projects, that students are asked to complete with their code. Students get to work on full programs from the beginning of the course instead of unimpressive snippets. By using skeleton projects, we are also more free to utilise the real constructs and libraries of the language employed by the industry.

However, the use of skeleton projects is not without its own difficulties. Students may either accidentally or on purpose modify the supplemented code intended to be used as defined, and end up with a different program that the instructor intended. We should clearly specify what students need to modify and what they should not touch. Additionally, documenting and explaining the supplemented code or library is very important.

EDCAT is intended to provide students with modern interactive GUIs that are executed in browser; what has been previously described in the literature has no real examples of similar systems. The use of REPL in communicating with a program is neither a modern nor an appealing design choice — as is not the use of Java applets. Additionally, we do not want to create a new system but utilise the existing ones combined with some new technologies.

Chapter 4

Web Development Technologies

Originally, the web was developed at CERN for sharing static scientific documents. Its foundation in HTML, URL and HTTP is still present although new technologies are becoming increasingly popular allowing the creation of dynamic applications. Furthermore, the developments in software engineering have affected web application development. The adoption of the Model-View-Controller design pattern for the web has spawn various frameworks based on different programming languages, such as Ruby, Java or Python [19].

There are also prominent frameworks written in Scala, such as Lift[1], Play[2], Scalatra[3] and Spray[4]. However, the purpose of this thesis is not to build a self sustained system that requires continuous maintenance but to use existing course systems with minimum modification. Thus, these frameworks will not be introduced even though the lightweight framework Scalatra might be interesting for students to learn.

The obscuring of the traditional division of client side and server side tasks as well as the transition of desktop applications' to web have resulted in even more dynamic and efficient web applications. Browsers have become more powerful tools that can execute sophisticated client side processes and render vector graphics on the go. [19]

---

[1] `http://www.liftweb.net/`
[2] `https://www.playframework.com/`
[3] `http://www.scalatra.org/`
[4] `http://spray.io/`

Previously, visually impressive websites have used Java applets or have been built with Adobe Flash[5]. Flash is proprietary software and solely utilises ActionScript. Porting software written in Scala would require extensive effort and produce an application dependent on a browser plugin, which are notoriously vulnerable to security flaws. Wrapping exercises into Java applets would be technically easy as Java and Scala are interoperable. However, running Java in a browser is no longer considered secure and for instance the latest versions of the highly popular browser Chrome no longer support Java plugins. Additionally, there are some size issues in Java Applets that must contain all class files; usually class files are shrunken and jar-packaging is used. To summarise, even if Adobe Flash or Java applets could provide a working solution, they would not result in a modern or an appealing result.

HTML5 is considered as a replacement for Flash and together with JavaScript it enables building applications that work similarly in all browsers. Because HTML5 can not provide interactivity on its own, we will look into solutions based on JavaScript. In Section 4.1 we explore options that execute code on the client side, while Section 4.2 is concentrated on solutions for server side code execution.

4.1 Client Side Code Execution

In recent years, programming has become more accessible for beginners with the development of new easy to use and available tools. Writing and executing program code written in a compiled programming language no longer requires installing a programming environment but can be done entirely online. Even compiling and executing of code is possible online with libraries relying heavily on JavaScript.

Skulpt[6] provides a JavaScript implementation of Python that runs in-browser. User-written Python code is first compiled into bytecode, and then decompiled to JavaScript. However, Python and JavaScript objects, functions, etc. are not equivalent. Hence, the JavaScript code generated by Skulpt is

---

[5] `https://www.adobe.com/products/flashruntimes.html`
[6] `http://www.skulpt.org/` and `https://github.com/skulpt/skulpt`

not proper JavaScript representation of Python and it can not be directly called from other JavaScript applications. Furthermore, Skulpt has been used together with the Online Python Tutor[7] [11] for teaching purposes in digital textbooks and in online programming courses.

A relatively new addition to JavaScript-based implementations is Scala.js[8], which is quite similar to Skulpt. Scala.js is a Scala to JavaScript compiler that transforms Scala code into intermediate `.sjsir` files. To produce proper JavaScript, the `.sjsir` files are linked and optimised into a single JavaScript file that can be incorporated with HTML. Unlike Skulpt, Scala.js provides much better interoperability with JavaScript code. Currently there are no documented examples of use of Scala.js for educational purposes.

Unlike Skulpt or Scala.js, Repl.it[9] supports several programming languages (e.g. C, C++, Java, Python, Scheme). Skulpt provides an in-browser environment for exploring programming languages and enables also sharing code snippets through URLs like pastebin. The ideology behind Repl.it is thus quite different from Skulpt and Scala.js — Repl.it provides, as its name suggests, only a *read-eval-print-loop* and is not intended as a library for use in private projects. For each implemented programming language Repl.it contains a language interpreter written in JavaScript, which translates the code to executable JavaScript. The project is no longer actively maintained, as the developers are moving to a sandboxed solution.

4.2   Server Side Code Execution

A more comprehensive solution to executing code online is to set up a sandbox or a virtual machine. Compared with client side execution restricted server side environments can offer higher performance and better control over security. On the down side, when the code is executed on a server, the user sees only the end result of an execution. Implementation of interactive functionality is limited and possible only if the system provides a way for user input, which

---

[7] `http://pythontutor.com/`

[8] `http://www.scala-js.org/` and `https://github.com/scala-js/scala-js`

[9] `http://repl.it/` and `https://github.com/replit/repl.it`

is generally standard input and has to be defined in advance for the entire duration of a program execution or the code needs to be compiled multiple times like in the solution that Yoo et al. [37] propose.

Codepad[10] is an online compiler and interpreter for about a dozen programming languages (e.g. C, C++, Python, Scheme) and operates also as a pastebin application. Codepad uses chroot jail for compiling and executing the resulting programs, which strictly limits the resources. Additionally, the run is supervised with `ptrace` and many system calls are discarded.

Ideone[11] created by Sphere Research Labs[12], like codepad, is an online compiler with support for several programming languages (e.g. C, C++, Java, Python, Scala, Scheme) and pastebin functionality. Compared with codepad, Ideone emphasises debugging — the code can be edited after execution, without altering the shareable URL. Sphere Research Labs offers Ideone and its API (called Sphere Engine) as a paid service for businesses and its precise technical implementation is not portrayed on their web sites. However, the high level solution is presented — the code is compiled and run on a remote server and the program's output is returned to the browser.

The general tendency seems to be that services that offer remote code execution on a server (e.g. Codecademy Labs[13], Coding Ground[14], lynda.com[15], Runnable[16]) also cater for commercial purposes. Therefore, in addition to the fact that service indeed compiles and executes code on the server side, no other technical facts are revealed.

---

[10] http://codepad.org/
[11] https://ideone.com/
[12] http://sphere-research.com/
[13] http://labs.codecademy.com
[14] http://www.tutorialspoint.com/codingground.htm
[15] https://www.lynda.com/
[16] http://runnable.com/

Part II

Design and Implementation

Chapter 5

System Design

This chapter clarifies the design goals for EDCAT and discussed the restrictions related in achieving those goals. Section 5.1 covers all relevant issues that should be considered in implementing EDCAT; Subsections 5.1.1 and 5.1.2 outline what options are available for the basis of EDCAT and analyse how the course's desired learning outcomes affect the end result. Considering the user requirements we evaluate the existing systems that could be used for the solution. Additionally, we consider what kind of requirements arise from the exercises, which are interactive and use media content. Finally, the design decisions are stated in Section 5.2.

5.1   System Goals and Constraints

We aim to create a tool that distributes GUIs that are interoperable with Scala programming assignments and enable the trial execution of those assignments. The assignments are interactive and use media content to challenge the students to apply their freshly acquired programming skills to real-world problems. Because there already are a few different systems in place as Aalto University, on top of those previously used and ran down, the aim is to integrate EDCAT with one of the existing systems. The design should be such that EDCAT could be easily taken to use with little modifications with another system.

The general requirements for EDCAT include ease of use and usefulness in assuring that implemented functionality works as desired. The system must not be more difficult to use than any other exercise submission system and must not cause any extra work to students compared with for example the last year's course. The system should provide sufficient information for whether the student's solution works. However, the system must not explicitly mark the solution as correct or incorrect. We want for students to use their own judgement in deciding when their solution is complete.

The system will have three different user groups; the teacher, the teaching assistants and the students. The students are offered the minimum functionality that is available to the two other user groups as well. The students must be able to see the exercise instructions and submit multiple solutions to those exercises. A submission must produce a working GUI for trying the implemented functionality in action. The solutions must be browsable and don't need to fully implement the functionality described by the exercise instructions to work. It should be clear for students as to which of their solutions will be graded.

The teaching assistants must have access to students' submissions in order to provide feedback and grade for them. Previously, the feedback and grading information has been sent to students via email. However, if teaching assistants could leave feedback and grades through the system, it would decrease the channels through which information is given to students.

The teacher user has the same functionality as the two described previously. Additionally, the teacher must be able to publish and manage exercises. The exercises can change after publishing and the teacher must have easy access to update instructions and resources. Implementing a GUI for the system should not require more work compared with a locally run GUI.

5.1.1 Existing Systems

As already discussed in Chapter 2 the previous year's course used A+ system while the *Programming 1* course, which runs parallel to this course, uses Goblin. Additionally, Goblin's grader EXPACA has been previously used

with A+ in *Programming Studio 1* course. EXPACA is designed to provide a grade for the assignment by comparing student's solution to a model solution. Both solutions are executed against a test class that awards point for passed actions. Finally, the grading returns a report of the execution as soon as it finishes or some mandatory requirement is not met. Out of the box EXPACA supports only this kind of grading and would require extensive rework in order to support returning anything else. Because Goblin is designed to use only EXPACA for grading and will probably be abandoned in the future, there is no sense in building the implementation for EDCAT on top of Goblin and EXPACA.

A+ system provides modern course management features that cover many of the set user requirements. The teaching assistants' requirements are already met without any additional configurations. Similarly, the teacher's requirements are met in every respect with the exception for GUI implementation; A+ does not set any limitation to implementation as long as it can be viewed in browser or started from the browser. The students' requirements are similarly met with the exception for those related to implementing the solution with no additional effort compared with the previous year. Additionally, whether the GUI provides sufficient information on student's success is not related to A+ directly.

Furthermore, A+ is designed to be extended by other services — for example grading software. There is a separate grader framework[1] suggested to be used together with A+. The grader can be hosted separately and should work with minimal effort together with an instance of A+. The grader has support for Python and Java submissions but it is too designed to be extended thus support for Scala could be easily added. By default the grader returns the feedback for the submission using one of the templates that can include any output resulting from the grading process. It would require little effort to implement an appropriate template that could include some embedded JavaScript that can be executed in browser.

---

[1] `https://github.com/Aalto-LeTech/mooc-grader`

5.1.2 Exercise Design

The exercises should introduce students to real-world programming problems that use sound and image and result in interactive programs. The course has five assignment rounds of which the last two involve some GUI programming and one of those exercises is concentrated on building a program that uses some networked content. The first three rounds should be easy to implement for browser execution but the latter two will be excluded from the scope of this project. Exercises from the previous year will be reworked to be used with EDCAT where suitable. In *Programming Studio 1* course we have encouraged students to extend the functionality of the exercises beyond what is required. The new implementation should support to some extent free expandability.

The first assignment round works as an introduction to programming and basic control structures. Previously, small game programming assignments have been proven practical and will be used also this year. The games of rock-paper-scissors and tic-tac-toe should be simple enough for a novice programmer to successfully implement. The second and third assignment rounds have previously been concentrated on sound and image filtering respectively. These assignments will be implemented almost as is also this year. The functionality that is implemented by the student is relatively simple but the GUI implementation could potentially prove challenging.

The GUIs and exercises should be designed so that students can return partially implemented solutions and receive meaningful feedback. Continual executions have been encouraged also in the previous iterations of the course and most of the used exercises support this naturally. The GUIs have been implemented in the past using Swing; using for example Scala.js, introduced in Chapter 4, should make it easy to port the existing exercises to EDCAT.

5.2 Design Decisions

A+ and the aforementioned grader are the most suitable for the implementation of EDCAT considering the set user requirements. A+ takes care of user management and defines only submission limits and deadlines for the exercises

while other information is stored within grader software. The grader software will need a server that can be accessed when exercises should need updating. The course configuration files should be version managed for example by `git` and thus can be easily updated.

Structure of EDCAT is pictured in Figure 5.1; the teacher sets up exercise limits in A+ and manages the materials on a server running the grader application. The student still accesses only A+ system, but no longer needs
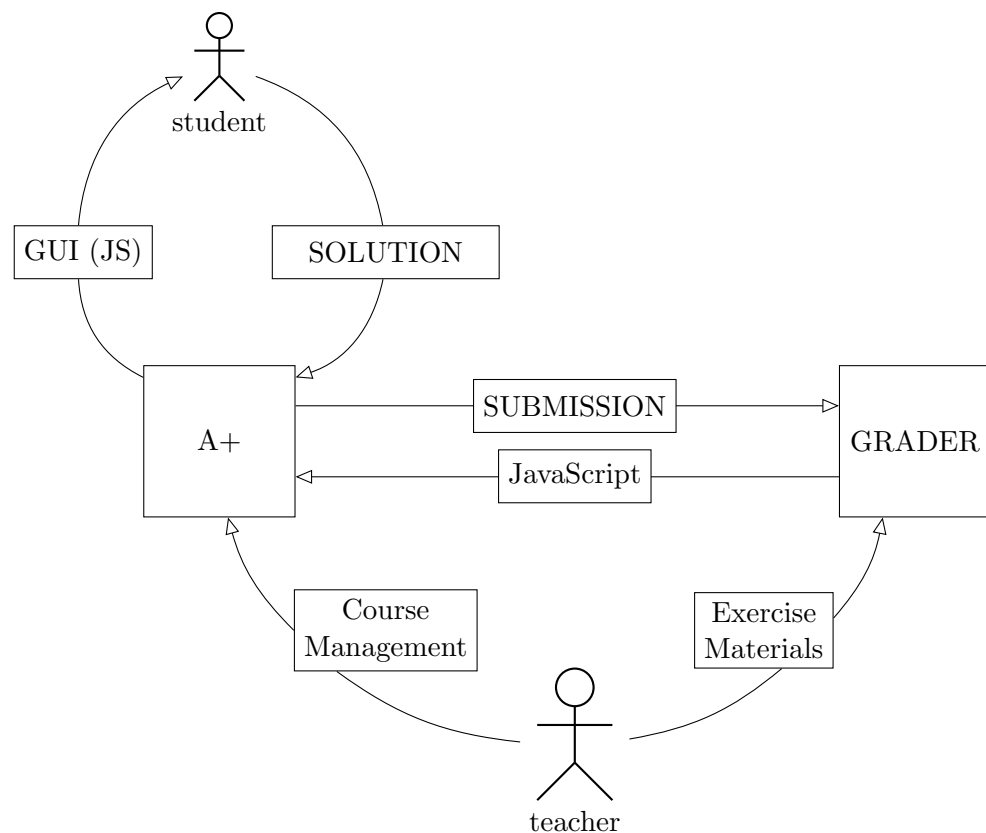

Figure 5.1: The student interacts with A+ system, which is responsible for sending the submission to grader and receiving the resulting JavaScript GUI. Through A+ interface, the teacher manages the course; like sets up the submission deadlines and limits. Through a connection to grader server, the teacher distributes the course material like assignment instructions and files needed for compilation.

to download exercise materials to their own computer. Students' solutions are submitted to the grader server, which in turn returns JavaScript code that the student can execute in their browser.

The students will have two submission areas for each assignment — one for executing the implementation and one for leaving the solution for grading. The latest submission to the grading area will be graded and given feedback to. The feedback will be posted through A+ system, which will cause a notification visible to the student. However, the cumulation of points will be followed separately and updated elsewhere. Because the assignments will be extendable, the students can be awarded more than the maximum points. However, granting more than maximum points is not possible in A+ and we want to keep maximum points value representative of the maximum points granted for the required functionality.

The GUIs for the assignments will be implemented using Scala.js, which allows compiling Scala code into JavaScript. The grader software will be responsible for compiling the JavaScript code and returning it to A+. The media files will be hosted in a separate location with access from A+.

Chapter 6

Implementation

EDCAT was implemented by combining a modified grader software used together with A+, which is a distributed e-learning platform and course management system. Both software are built with Python and fulfil distinct functionality; A+ provides the infrastructure for managing the exercises and students' submissions, and the grader is used for defining the exercises for the course and can be used to grade them. A+ provides a web interface built with Django while the grader is managed entirely through command line directly on the server.

First the user interface and its functionality is discussed in Section 6.1 after which we present the details of technical implementation in Section 6.2.

6.1 Functionality

The core functionality of A+ provides the means for a student to view exercise instructions, to submit exercises, and to see any of their own previous submissions. Maximum and earned points as well as the number of used submissions and the submission deadlines are shown both on exercise pages and course's main page as seen in Figure 6.1.

For the teacher or a teaching assistant A+ allows viewing all submissions, which they can resubmit, leave feedback for or adjust the grading of. The functionalities restricted to the teacher include for example modifying course's
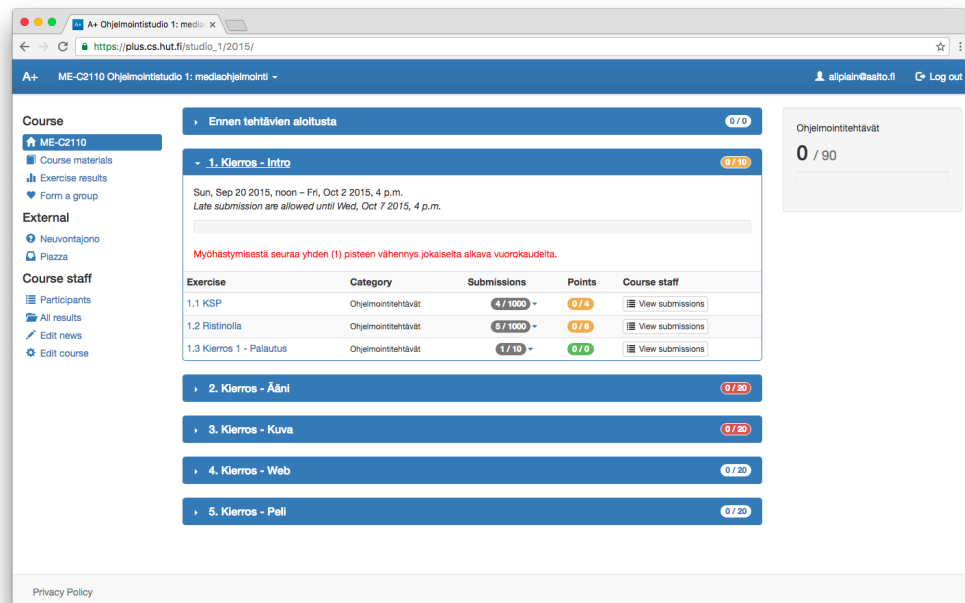
Figure 6.1: The front page of a course with one module expanded. This screen capture was taken with the teacher privileges and shows links related to managing the course and submissions. The students are not able to see the content displayed under the *Course staff* headings.

validity period, granting teaching assistant rights, and creating and modifying exercises. The functionality related to course management is discussed in more detail in Section 6.2.

The instruction pages function as submission forms for the exercises; at the bottom of each instruction page is a simple file upload form with fields for each required file. If the student tries to submit less than the number of required files, she is immediately redirected to the submission page with an error message that states that all files are required. If there is some problem with the submitted files, duplicate files or wrong files like for example *.class* files, the submission results in a compilation error.

When the student successfully submits all the needed files, she is redirected to a submission page. If the grading of the submission takes some time, the student is presented with a progress bar and a message stating that the
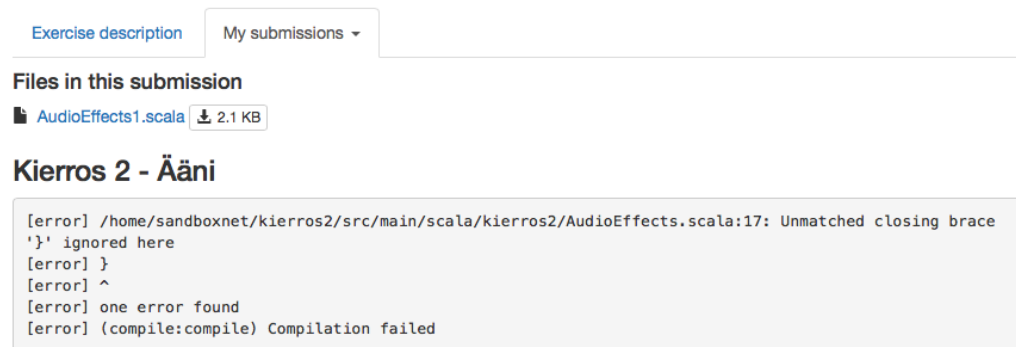
Figure 6.2: A detailed view of submission that resulted in a compilation error.

grading is taking longer than expected. The student is then required to wait and refresh the page in order to see the graded, or in this case compiled, submission. In case the compilation fails, the student is presented with a compilation error message similar to one in Figure 6.2. If the submission compiles successfully, the resulting graphical user interface in JavaScript is presented. The graphical user interfaces used for this course are discussed in more detail in Section 6.1.1.

6.1.1 Prepared Graphical User Interfaces

The graphical user interfaces used for the assignments, shown in Figures 6.3 through 6.6, were relatively simple and easy to seamlessly embed to the base of the A+ system. The first assignment round consisted of two simple assignments — the games of rock-paper-scissors and tic-tac-toe — the GUIs to which are shown in Figures 6.3 and 6.4 respectively. The assignments familiarised students with random number generation and the basics of game programming.

The rock-paper-scissors GUI, pictured in Figure 6.3 in its basic form consist of three elements; an area to show both the user's and the computer selected hand, three buttons for selecting the hand to play, and an area to show the result of the game. The only interactive elements are the three buttons that all function alike. When the user selects a hand to play by

pressing one of the buttons, the picture for user's hand is updated, the method that should return the computer's hand is called, picture for computer's hand is updated, and the end result of the game is updated.

In order to carry out the assignment, the students are supplied with a project template that contains a skeleton for a Scala class to implement their solution to and a Scala class with a complete implementation of enumeration
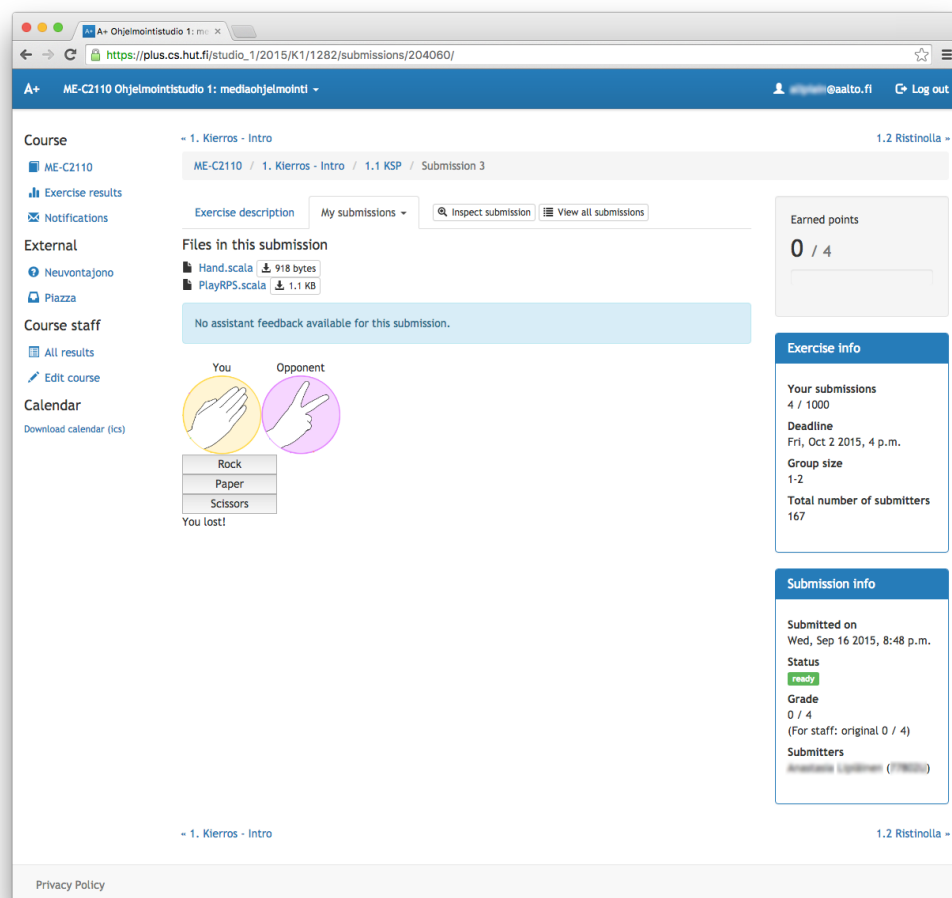
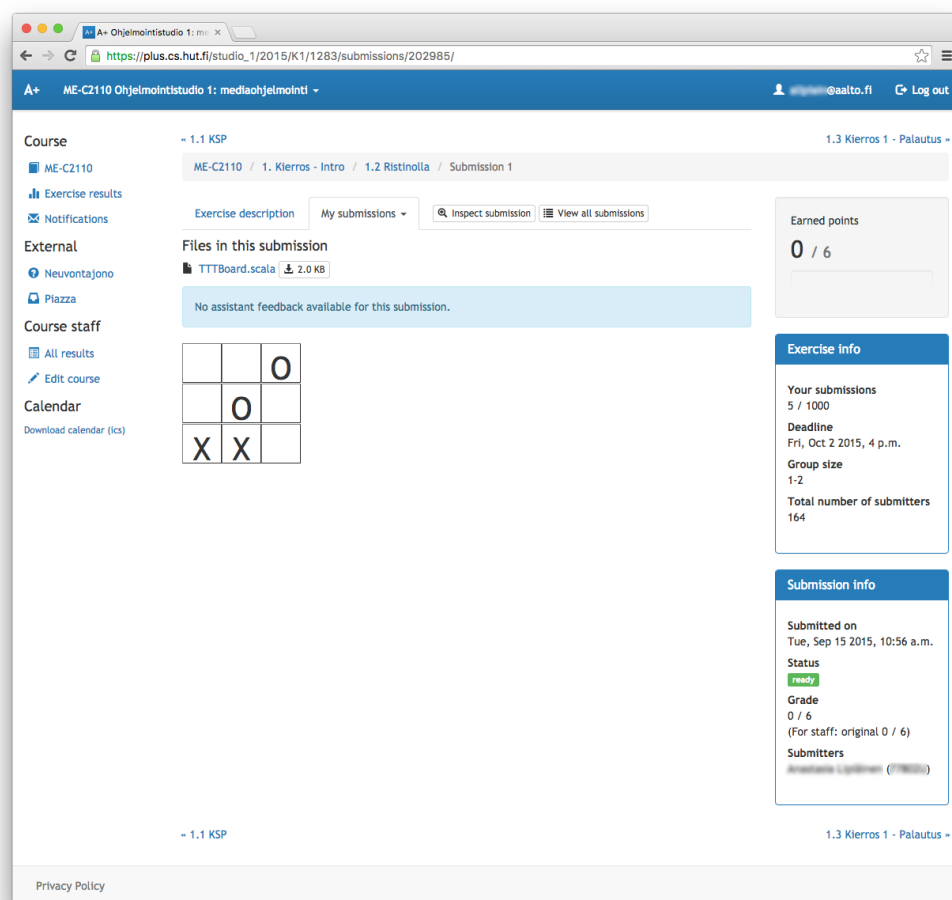

Figure 6.3: The view of a successful submission to the assignment 1.1 with the resulting graphical user interface as JavaScript. The user has selected to play *the paper* while computer opponent has randomly selected *the scissors*. Below the buttons for the possible plays the user is shown the end result of the game – in this case defeat.

that models the possible plays of the game, in this case the hand gestures. For the students, the assignment is to implement the method that can randomly generate a hand for the computer player to play and the method that returns the end result of the game.

As a voluntary expansion on the mandatory work, the students were asked to extend the game with two additional plays of a lizard and Spock, which required adding some lines to the enumeration class and adjusting the logic

Figure 6.4: The view of a successful submission to the assignment 1.2 with the resulting graphical user interface as JavaScript. The user plays the O marks while the computer opponent the X marks.

in the method that returns the outcome of the game. The GUI was built so that it automatically generates buttons for all hands without requiring modifications to the GUI class.

The tic-tac-toe GUI, shown in Figure 6.4, is even more simple than the rock-paper-scissors GUI. The grid of the game is build with nine buttons that show the played mark as text. When the game ends with either one of the players winning or every cell in the grid is filled, the end result of the game is showed beneath the game area similar to the rock-paper-scissors GUI.

The user gets to place the first mark by pressing the desired cell in the grid, which updates the text of the button to correspond to the user's mark. Then, until the game ends, the user and the computer get alternating turns. When it is the computer player's turn, the method that should return the coordinates of an empty cell to place the computer player's mark in is called. When the game ends, the method that returns the end result is called and the result is showed.

The student's assignment is to implement methods that allow querying and assigning a mark to the grid, which are used for drawing the grid and placing players' marks. Additionally, the student needs to implement methods that return truth-value for querying if the grid is full and if the game has ended, and most importantly the methods that return coordinates to place the computers mark in and the mark of the winner of the game. For the optional expansion students were asked to improve the AI of the computer opponent for which the required solution was to randomly return any of the empty cell coordinates. Similar to the first assignment, students were provided with the project template to implement their solution to. In the project template, there is a skeleton of a Scala class to implement the required methods to and an enumeration representing the marks, which allows successfully compiling the project locally.

The solution for the second assignment round results in a sound manipulation program that uses student created filters to achieve various effects. The GUI, shown in Figure 6.5, consists of four elements; the waveforms for the left and the right channels of the sound, a button to play the sound, the drop-down menu for selecting the sound, and the required input fields for
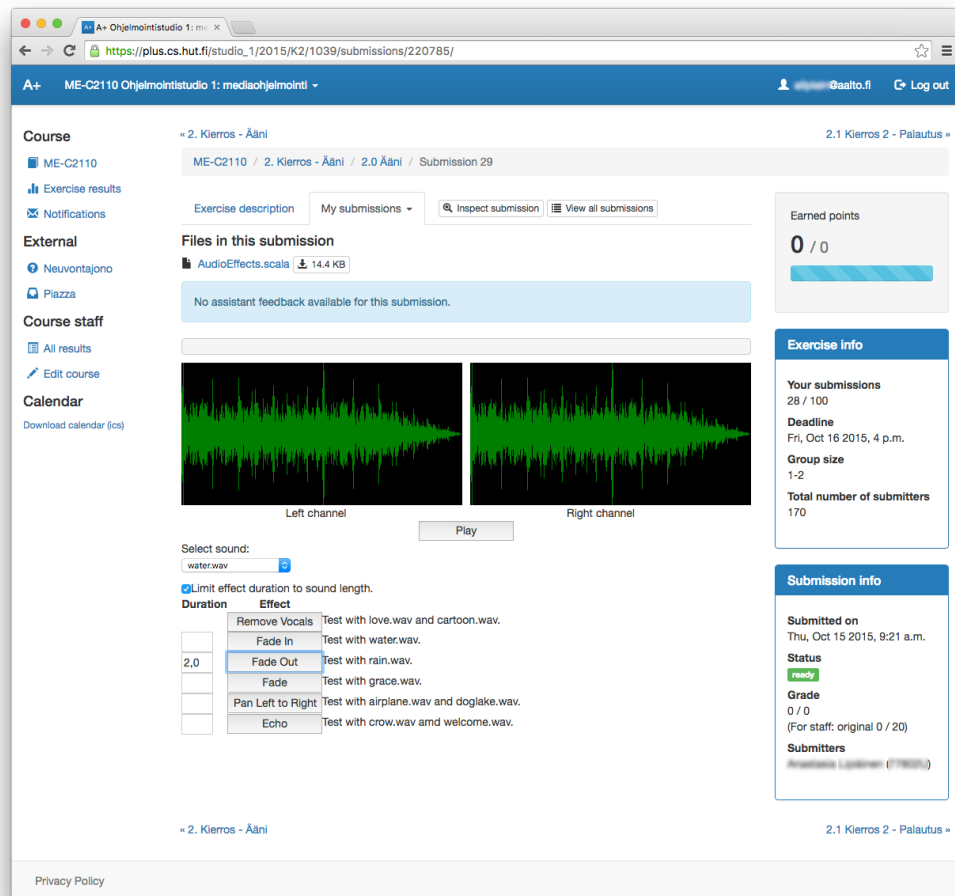
(a) The selected sound's waveform is shown at the top with all the interactive elements below.
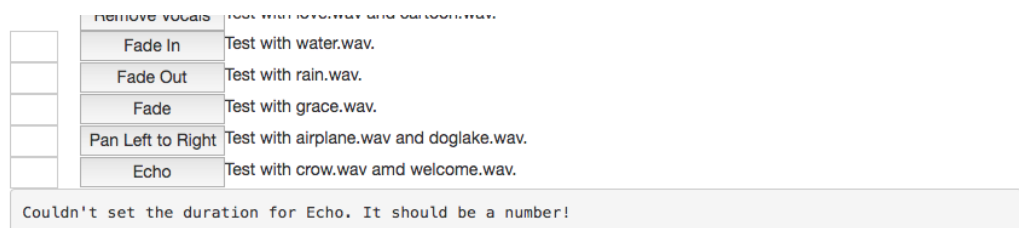
(b) An error message shown to the user after pressing the *Echo* button without input to the accompanying field.

Figure 6.5: The GUI in JavaScript with a highlight of an error message for the second assignment.

using the implemented filters. The GUI is relatively static; the only updated elements are the waveforms and an error message is shown if the user inputs an invalid value to a field.

The waveforms are changed when the user selects a new sound or modifies the sound with a filter. The waveforms also feature an animated progress indicator, which is updated when the sound is playing. The error message, showcased in Figure 6.5, is shown as preformatted text below the static UI. The input to the duration fields is expected to be given as seconds and allow only numerical input greater than zero with a comma as decimal mark. The fields can also be set to accept a value at most the length of the selected sound by selecting the checkbox marked *Limit effect duration to sound length.* Submitting an invalid value will not have any side effects. Furthermore, when the user takes the use of field's spinners the value can be set to the precise value of sound's length.

Pressing the *Play* button causes the selected sound to be repeated. To keep things simple, there is no way to pause the playback and if the user presses the *Play* button while a sound is playing, the repetition of the sound is started from the beginning while the previous playback is still played. Additionally, applying the filter not only modifies the sound but also repeats it.

Figure 6.5 shows all filters specified in the exercise instructions. The input fields and buttons are dynamically generated depending on the implemented functionality. In addition to the six required filters the students were welcome to implement any of their own that would be used either with only a button or a button and a numeric input field. The required filters included one that eliminates frequencies common to both the right and the left channels, four variation on a fading filter, and an echo filter. To facilitate the implementation, the students were again supplied with a project template attached with all the libraries required for a successful compilation.

The third assignment required students to create filters for image manipulation. The GUI is very similar to the sound manipulation GUI; the differences are that the image does not require a play button and is updated instantly, and the inputs to filters use buttons and sliders instead of number fields and

buttons. The GUI in the third assignment did not have any dynamic error messages and is shown in its entirety in Figure 6.6.

The GUI shows the original image always on the left while the modified image is updated on the right when the user applies the filters. The exercise instructions specify eight different image filters for students to implement. There are two filters that don't need any parameters and are applied using only a button — those to invert the colours of the image and to convert the colours to greyscale. Additionally, there is a button that returns the modified
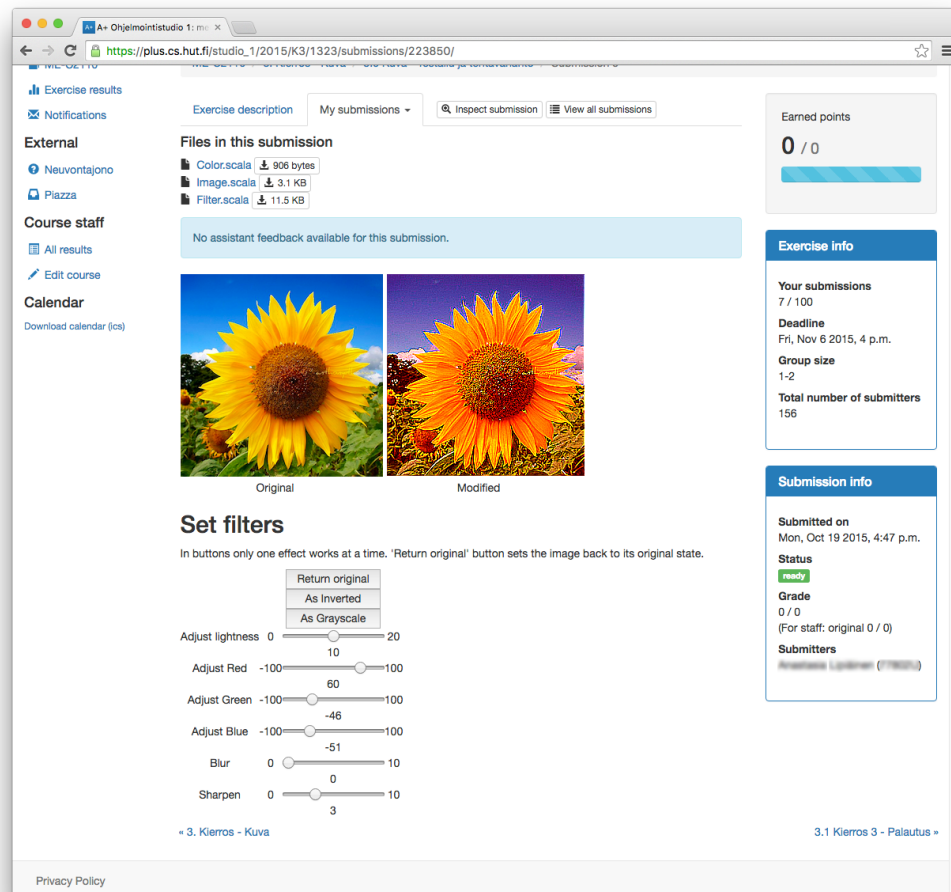
Figure 6.6: The GUI in JavaScript returned for a successful submission to the third assignment. The image's colour and sharpness have been adjusted.

image to its original state. The rest of the filters require input to adjust the intensity of the effect. There is a filter for adjusting the lightness of the image and three filters that adjust the colour components of the image in RGB colour space. Finally, the two more complex filters adjust the sharpness of the image.

As with the previous assignments, the students are provided with a project template to implement their solution into. This time they were required not only to implement the filters but also a class that models the colour in RGB colour space. For additional points, students could implement their own filters that could either take input from a slider or require no parameters.

6.2 Technical Details

The exercises are created in A+ but they are managed through grader software. Furthermore, the actual code that defines the GUIs is written in Scala using Scala.js library. The following subsections explain in detail how the exercises are built and managed from the technical perspective.

6.2.1 Course Management System

The A+ system is hosted and developed by the *Learning + Technology* group of Computer Science department in Aalto University. It is built with Python 3.7 and Django 1.7. and hosted on an Apache 2 server with uWSGI and Postgresql database. The maintenance team creates a course and an instance for the course and sets the teachers for the course, who have rights to fully manage the course. A teacher can set up the teaching assistants, but students can enrol to the course whilst it is visible by searching it through the landing page of the A+ system.

The two most important features for the teacher of the course are creating exercises and viewing student's submissions. A course consists of modules that can contain one or several exercises as well as reading material. Modules have an opening and closing times, during which time all exercises in that module must be submitted unless late submission is allowed. There are four

Figure 6.7: An example of the page to setup an exercise; here are shown the settings for the second assignment. Most of the information, like the exercise instructions, is fetched from service uri. The most important setting on A+ is the amount of maximum submissions.

different kinds of exercise types available intended to be used with different grading methods and service models.

Our course had simple remotely hosted exercises that fetch the exercise definition from provided service URL. One example of an exercise definition is shown in Figure 6.7. The service URL should point to a valid exercise definition file, whose structure is explained in Section 6.2.2. This exercise definition amongst other things has settings for the number of allowed submissions and points granted for the exercise. The number of allowed submissions was set so high that it was virtually limitless and students were not granted any points through the system, because grading was handled by the teaching assistants.

The submissions are searchable and visible to both the teacher and the teaching assistants. You can see an excerpt from the submissions listing in Figure 6.8. The submissions can be searched with student's name or student number, submission date and result status. The submissions containing the submitted files and the feedback returned by the grading service can be individually inspected, as seen in Figure 6.9. Additionally, both the

Figure 6.8: The excerpt from the submissions page with student names and numbers blurred. As visible, few of the assignments have resulted in a compilation error and their status is *error*.

Figure 6.9: The excerpt from view for inspecting a submission. The teacher can also see the same view as the student by pressing the submission ID link. This page allows leaving feedback for the submission.

teacher and the teaching assistants can leave feedback to the submission. This functionality was occasionally used by the teacher to interfere if a student had multiple consecutive erroneous submissions, and by teaching assistants to give verbal feedback for the assignment.

6.2.2 Grader Software

The grader software was installed on a virtual server with Ubuntu 12.04.4 with 512MB of RAM and 10GB storage space. The grader is implemented using Django 1.7 with asynchronous grading queue using Celery 3.1, the grader works both with Python 2.7 and 3.4. The submissions are accepted via HTTP, graded in a sandboxed directory and discarded after grading. On top of standard requirements for grader software, the course required installing JDK 7, Scala 2.11.7 and sbt 0.13.7 for compiling the submissions.

Exercise Definition Files

The course and its exercises are defined in subfolders with configuration files written in either JSON or YAML. The folder for the course contains an index file that defines the exercises as a list where each item or an exercise key must be pointed to a configuration file of the same name. In addition to the course list, the course configuration file contained definition for the name and the language of the course. All files related to the exercise, like additional program code or any other supporting files, are similarly stored in a folder that is named according to the exercise key.

```yaml
1   title: Exercise 1
2   description: The exercise results in a program.
3   instructions: |
4     <!-- Instructions for what students should do. HTML is a good
5          option when the guidelines are complex or extensive -->
6
7   max_points: 0
8
9   # view_type field defines what is submitted.
10  # This definition states that students should submit
11  # two files named Main.scala and Helper.scala
12  view_type: access.types.stdasync.acceptFiles
13    files:
14      - field: file1
15        name: Main.scala
16      - field: file2
17        name: Helper.scala
18
19    actions: # What is done with the submission.
20      - type: grader.actions.prepare
21        charset: utf-8
22      - type: grader.actions.sandbox
23        cmd: [ "precompile.sh", "exercise_key", "Main Helper" ]
24        expect_success: true
25      - type: grader.actions.sandbox
26        cmd: [ "sbt_compile.sh", "exercise_key", "Main Helper"]
27        net: true
28        expect_success: true
29      - type: grader.actions.sandbox
30        cmd: [ "assemble_submission.sh" ]
31
32    # Passes raw grading task output.
33    feedback_template: access/task_direct.html
34
```

Listing 6.1: The exercises are defined with YAML files that contain information shown to the students as well as information about what students should submit and how the submissions should be handled.

All exercises for EDCAT were defined almost identically; each configuration file was similarly structured and the accompanying folders all contained build definition files for *sbt*, the library files for Scala.js., Scala files containing the code needed for GUI, and the HTML file for the response GUI. The exercise configuration files were written in YAML and followed a similar structure of which you can see an example in Listing 6.1.

Of used attributes only `title` and `view_type` were required. Each exercise definition file was additionally supplemented with `description`, `instructions`, and `max_points` attributes. As the grader software was used together with A+, values of those additional attributes were directly relaid to the corresponding options of the exercise configuration inside the A+ system.

The `view_type` attribute specifies the type of the exercise and especially how the exercise is submitted and graded. There are seven predefined types available, including different form and file submission options and it is also possible to define a specific `view_type` for the course. In or course, all exercises were of type `acceptFiles`, which required students to submit one or more files that were then processed on the server and relaid to the asynchronous grading queue for compilation to JavaScript.

The `acceptFiles` type has itself both required and optional attributes that define what files should be submitted, how they are handled, and what is shown to the student. The configuration requires at least the list of expected files and the list of actions that will be performed on each submission. Additionally it is possible to specify the number of required files if not all are required, a message to show after the submission is accepted, whether the client should try to fetch the feedback automatically, and which templates are shown to the student for making the submission and as a response.

In addition to the required attributes of `acceptFiles`, our course utilised `feedback_template` attribute, which specified to use a template that directly shows the feedback generated by actions. In our case, the feedback that actions generated was a HTML file embedded with the JavaScript compiled from submitted Scala code or a stack trace of a failed compilation.

An example definition of the `files` attribute of `acceptFiles` type is shown in Listing 6.1 on lines thirteen through seventeen.For each file, definitions

for the field and file names should be supplied, and optionally information whether the file is required for the submission, which defaults to `true`.

Action list uses specific types of actions, of which there are seven predefined available and which can be defined to fit course specific purposes. The predefined action types have their own set of attributes but there are also attributes common to all action types. The common attributes include override options for awarded and maximum points, title for the action, whether the output should be written as HTML in a template and options to set what should be done if the action fails. EDCAT used `expect_success` attribute that will on failure set the grading state to *Error* and return from executing the actions. In the case of our course, the compilation tasks should be successful in order to continue onto assembling the response HTML.

The actions are executed in the order specified by the defined list seen on lines nineteen through thirty in Listing 6.1. Of the seven available action types only `prepare` and `sandbox` were needed, while other actions included those specific for Python and `git` submissions, those that should be ran with separate grader software, and an action for storing student's submission without any manipulations.

The `prepare` actions are performed outside sandbox environment usually before the actual grading procedures and include tasks related to modifying files to allow grading. Attributes for moving and copying files are available as well as those for unpacking or setting the character encoding for the submission files. The submissions in our course were all first converted to use UTF-8 character encoding as seen on the line twenty-one of Listing 6.1.

The `sandbox` actions are performed after chroot under a restricted user in a sandbox environment so potentially malicious or malfunctioning code can be executed safely. The functions performed are given in a `cmd` attribute that contains in an array a command for the command line followed by the required parameters. Other available attributes include settings for network usage, as seen on line twenty-seven of Listing 6.1, or limiting the time, memory, disk space, and file usage.

The commands used in our course were all defined in scripts. In the first and second assignment rounds the submissions were first compiled to

JavaScript and then included in result HTML to return to the submitter (see lines twenty-six and thirty of Listing 6.1). For the third assignment round the pre-compilation script, as seen on line twenty-three of Listing 6.1, was added to shorten the time spent on malformed submissions.

Compilation Scripts

As a response to the submission, the `task_direct` template is returned to the A+ system regardless of the success of the submission. The template includes any possible content of standard error stream produced by the compilation process as preformatted text at the beginning of feedback. The content of standard output stream is presented as is, which means that the feedback upon successful compilation can be printed directly to the standard output stream.

The compilation process took use of three scripts of which one was used only in the third assignment round. The `precompile` script was added after the second assignment round in order to speed up the compilation process. The script takes as a parameter the files submitted by a student and the folder that holds the GUI code. It then attempts to compile all program code files together using `fsc` and returns the result of the compilation as exit status, i.e. the information whether the compilation was successful or failed.

If the files do not compile and produce an error with the accompanying error message, the error message will be included in the feedback passed onto A+ system and further scripts are not executed. Precompilation ensures that submission contains valid Scala code and requires spending significantly less time on clearly erroneous submissions instead of compiling them directly with much more time consuming `fastOptJS`.

The `sbt_compile` script works analogous to `precompile` script. First the files are copied to network enabled sandboxed directory that also contains the GUI code. The compilation is done using `fastOptJS` that produces the JavaScript used in the response to the student. Because every compilation is performed in this same networked sandboxed directory, the JavaScript file is then copied to the student's submission folder and the files produced by

the compilation — like `.sjsir` and `.class` files as well as the JavaScript file and its source map — are deleted. If compilation fails, the error message is conveyed to the A+ system and the `assemble_submission` script is not executed.

Finally, the resulting JavaScript code is embedded in a HTML page to be passed to the A+ system. The `assemble_submission` script prints out between `html` tags the contents of the JavaScript and `index.html` file.

6.2.3 Creating GUIs with Scala.js

Most importantly, every GUI designed to be used as a part of an exercise needed to be separate from the program logic written by students. Secondly, the exercises and GUIs needed to be to some extent extendable with students' own functionality. The used methods were explicitly defined in the exercise instructions and in second and third assignment rounds, where students wrote methods for their own filters, the filters were passed to the GUI as an iterable collection with functions as values.

All GUIs except the one for the sound manipulation assignment consist of just one class. The GUI for the second assignment needs more complex functionality which was separated and packaged to its own library. The GUIs are basically Scala code; each written as singleton object that utilises Scala.js, scala-js-dom [1] and ScalaTags [2] libraries. The Scala.js library provides annotation `JSExport`, that marks the object and needed method to be callable from the HTML that the JavaScript is embedded to. ScalaTags library is used for creating the HTML elements of the GUI and scala-js-dom library for access to some additional HTML elements like AudioContext and CanvasRenderingContext2D. A mock-up of GUI class can be seen in Listing 6.2.

Structure of each GUI implementation was similar; the main object had a method that constructed the interface in HTML tags and defined functionality

---

[1] `http://scala-js.github.io/scala-js-dom/`,
`https://github.com/scala-js/scala-js-dom`
[2] `http://www.lihaoyi.com/scalatags/`,
`https://github.com/lihaoyi/scalatags`

for user interaction. The interaction was achieved by defining an inner function that was called when the HTML elements needed to be updated. Each interactive input element defined functionality for click or update event;

```scala
package exercise

import scala.scalajs.js

import scalatags.JsDom.all._

import org.scalajs.dom
import org.scalajs.dom.html

@js.annotation.JSExport
object Main {

  @js.annotation.JSExport
  def main(target: html.Div): Unit = {

    // methods used for operational logic and GUI generation

    // an example of an button definition
    val button = input(
      `type` := "submit",
      value := "button text"
    }
    button.onclick = (e: dom.Event) => {
      // do something
      update()
    }

    def update() : Unit = {
      // Remove all elements and add them anew
      val children = target.childNodes
      for (child <- 0 until children.length) {
        target.removeChild(children.item(child))
      }
      target.appendChild(
        div(
          someTag(
            attribute := "something else",
            numericAttribute := 999
          ),
          otherTag("some content"),
          button
        ).render
      )
    }
    update()
  }
}
```

Listing 6.2: An example of what a GUI class looks like. The object and method that are referenced from the HTML need to marked with JSExport annotation. Otherwise the implementation is typical Scala code.

in addition to changing the inner state of that element the update function was called.

All assignments but one, the tic-tac-toe game, used image or sound resources that were hosted separately from both the course management and grading systems. This means that in order to use those resources successfully, we had to deal with cross-origin HTTP requests. The server hosting media files was configured to accept `GET` requests from all domains. For sound and image manipulation assignments the resources were fetched from HTML code. However, because the rock-paper-scissors game was extendable by adding more plays, the students needed to have access to image representation definitions. The additional images were hosted in the same place as the default ones, so it was quite easy to deduce where to find those pictures needed for the extension.

Part III

Evaluation

Chapter 7

Analysis of Students' Views

Students' views were collected through a questionnaire after the first three assignment rounds. In addition to the questionnaire, some students were interviewed to enhance the results gathered through the questionnaire. Additionally, students' attitudes were accessed after each round through assignment feedback forms. The questionnaire is the primary source for students' feedback and other sources authenticate and enrich the feedback gathered from the questionnaire. The feedback forms allowed to try and address some issues already during the course in addition to verifying the results of the questionnaire and the interviews.

This chapter covers students' views accumulated from all three sources mentioned above. First, the student questionnaire is discussed in Section 7.1. The insights drawn from assignment feedback forms and interviews are presented in Sections 7.2 and 7.3 respectively. Finally, a summary that combines all findings is presented in Section 7.4.

7.1 Student Questionnaire

7.1.1 Data Collection

The student questionnaire was open for all students in the *Programming Studio 1: Media Programming* course. Of one hundred and seventy initially

registered students, one hundred and fifty were actively completing the course after the first three assignment rounds. The questionnaire, which you can find in Appendix A, was given entirely in Finnish and did not include any background information questions.

The questionnaire had a short description about the implementation of the system, where the distinction between the A+ system and the server was emphasised. It was stated that the A+ system is responsible for accepting students' submissions and forwarding them to the compilation server, which returns the resulting JavaScript for the student to try out. The students were asked to concentrate especially on the server related aspects of the service.

The questionnaire concentrated on three themes; usefulness, ease of use, and user experience of the system. Each category consisted of four Likert items on a five level scale from *completely disagree* to *completely agree* and an open field for supplying any comments related to the category. At the end of the questionnaire, students were asked if they had any specific problems while using the system and if they had any ideas on what parts of the service should be developed and what were the most useful ones.

After submitting the questionnaire students were forwarded to another form through which they could submit their contact information in order to receive bonus points for the course and volunteer for a separate interview. The contact information form had fields for the respondents' name and email address and asked to choose if they wanted to participate in the interview *rather alone, rather in a group, does not matter* or did not want to participate in the interview at all.

7.1.2 Results

The student questionnaire received fifty responses, which is about a third of all the potential respondents. All but one of the respondents provided their free-form comments in Finnish and thus all citations in this section are translated. Even though no background information was asked, we can conclude some facts about the respondents from responses to the contact information form. At least forty-three respondents used EDCAT for all of

the first three assignment rounds, one respondent did not use the system for the first assignment round, while six respondents did not provide any contact information and cannot be identified. Forty-three out of forty-four known respondents completed the course successfully.

Usefulness

The respondents were first asked to evaluate the usefulness of the system and the responses to this category are compiled to Figure 7.1. Every statement of the usefulness category scored positively with more than half of the answers being *agree* or *completely agree*. With the broad statement, *I found the system useful*, twenty-five respondents agreed and eleven strongly agreed while only three respondents strongly disagreed and four disagreed. The statement that using the system was helpful while solving the assignments scored quite similarly. The respondents somewhat disagreed more with the statement that the system was helpful in revealing defects in student's program code. Similarly, the statement which asked students to evaluate if the use of the system helped them to understand the functioning of their code received milder responses with respondents choosing *neither agree nor disagree*.

In the free-form comment section, respondents left mostly positive feedback. However, there were five commenters who were generally dissatisfied with EDCAT and six commenters who hoped for a better support for debugging specifically. Most of the dissatisfied commenters provided nothing more than mere complaints. Blame was given especially to systems performance times with such comments as "*the submission system could as well have been replaced with a better working implementation*" and "*– the assessment server completely stopped working under even a small stress, I was unable to work with it*".

However, one of the negative commenters provided more detailed criticism. The respondent blamed EDCAT as being useless for debugging because there were no error messages and it was not possible to test methods separately, because a mistake in one method affected all other methods. Additionally, the respondent was dissatisfied with EDCAT because in the assignments on
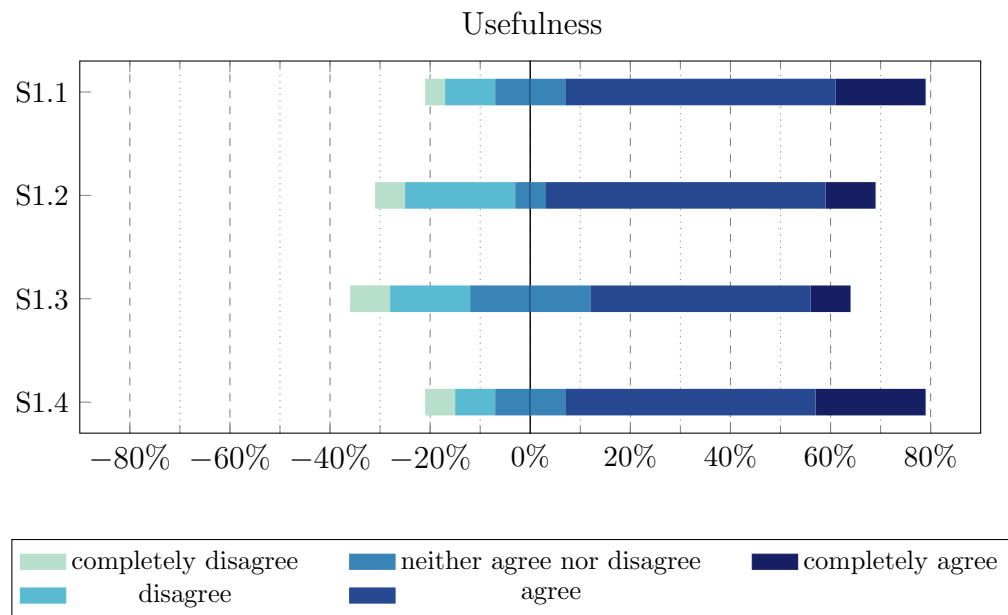
Figure 7.1: The results of the usefulness category in the student questionnaire. The bar to the left of center shows negative reactions — *completely disagree* and *disagree* — and half of neutral responses while the bar to the right of center shows the positive reactions — *agree* and *completely agree* — and half of neutral responses. The amounts are shown as percentages. The Likert items of this category are listed below.

S1.1 Use of the system helped me to solve given assignments

S1.2 Use of the system helped me to find concealed defects in my program

S1.3 Use of the system helped me to understand functioning of the code I wrote

S1.4 I found the system useful

sound and image manipulation there was no other way of knowing if the code was correct than superficial evaluation. The respondent was equally displeased with the fact that students were encouraged to write their own tests but it was not possible because they were not provided with sample output for programs.

The feedback that concentrated on debugging features was generally positive but blamed the lack of visible error messages. The commenters mentioned that it was possible to detect if the program was not functioning

correctly, but there often was no clue as to where to find the error. Though a couple of respondents mentioned using JavaScript console to find out what was wrong.

All assignment rounds got some positive feedback, because most errors were visually detectable. However, a few respondents expressed dissatisfaction with the system in the third assignment round because all filters were applied to the image whenever any of the sliders managing a filter was moved, albeit this feature helped one commenter to find an error in their code. One respondent thought that perhaps it was even a good thing that EDCAT did not provide detailed help to finding errors. Additionally, one respondent was bothered by not knowing how exactly the submitted code was handled by the system and what was the hoped outcome; the respondent suggested that the system could indicate if the submitted solution was similar or close to the model solution.

Otherwise free-form feedback related to the usefulness of EDCAT was mostly positive encouragements. The respondents mentioned for example that it was motivating to play with one's code because of the system, and that the best thing about the system was that one could really see one's own work as a "*modified media content*" or "*do something concrete*". One of the respondents thought that EDCAT provided a handy way of concealing GUI code and permitted the student to concentrate on a well-defined problem.

Ease of Use

The ease of use of the system was similarly estimated with four Likert items, whose responses are compiled to Figure 7.2. The respondents evaluated the ease of use of the system positively through each statement. Roughly four fifths of the respondents answered either *agree* or *completely agree* in respect of statements that argued that the system was easy to learn to use, and in its entirety easy to use. Quite a similar response got the statement that evaluated if students would have needed more support from course personnel; 70 percent of respondents answered either *disagree* or *strongly disagree*. The only statement that received a milder support was the one claiming that

interaction with the system was clear and understandable; only two thirds of respondents chose *agree* or *completely agree* and slightly over a fifth of the respondents chose *disagree* or *completely disagree.*

In the free-form comment section students left feedback mostly unrelated to EDCAT. Five out of seventeen students who left a reply to this section concentrated on the fact that students were required to make a separate submission in order to turn in their solution for grading and that web-UI was intended merely for experimenting with the functionality of their code. One respondent also mentioned that the system did not provide sufficient information about the queue situation; the position in the queue and the time

Ease of use

Figure 7.2: The results of the ease of use category in the student questionnaire. The Likert items of this category are listed below.

S2.1 It was easy for me to learn to use the system

S2.2 Interaction with the system was clear and understandable

S2.3 I would have needed more support from course personnel to use the system

S2.4 I think that the system was in its entirety easy to use

estimate were not updated after the submission. A few students hoped that they had heard about the JavaScript console sooner and one of them noted that JavaScript code was not exactly the same as the Scala code, which made things more complicated.

Otherwise the respondents were generally satisfied with the ease of use of EDCAT, but there were also improvement suggestions and some displease. One respondent said that the system was simple to use, but that "*the test design for individual rounds was varying and often left to hope for.*" Again the compilation time was mentioned by one respondent. Similarly another respondent gave a concrete suggestion to make the filters in the image manipulation round possible to turn on and off, which could help to determine which filter does not perform accurately and possibly decrease the number of unnecessary submissions.

User Experience

The user experience of the system was evaluated again through four Likert items, whose responses are compiled to Figure 7.3. Unlike the previous question categories, respondents were clearly unsatisfied with the user experience of the system. Seventy-four percent of the respondents thought they did not receive feedback from the system sufficiently quickly answering either *disagree* or *completely disagree* to the first statement in the category. The third statement, which evaluated if the students received sufficient information about runtime exceptions, received almost equally poor score of sixty-eight percent answering either *disagree* or *completely disagree.*

In the free-form comment section for user experience, most respondents — as many as twenty-two out of twenty-eight — concentrated on EDCAT's performance time. Of those, one respondent thought that the system did not even need to perform faster in this kind of use, while others mostly thought that the system performed too slowly. In addition to speed issues, the respondents again mentioned that during runtime, exceptions were not visible other than in JavaScript console. Some respondents had misinterpreted that the system was lagging when in fact their code had caused a runtime exception

they did not notice. These issues worsened during the third assignment round when compilation times were especially poor and there were many submissions. One commenter said that "*the queue situation was at times unreasonable*" and some reported waiting twenty or thirty minutes at times and the system even crashing under stress.

Some minor issues included that A+ did not update the submission page; it had seemed that the submission had not yet reached the server when it was already compiling and similarly the page required refreshing to get the compiled result after submission. Additionally, in the first and second assignment rounds and at the start of the third assignment round, refreshing
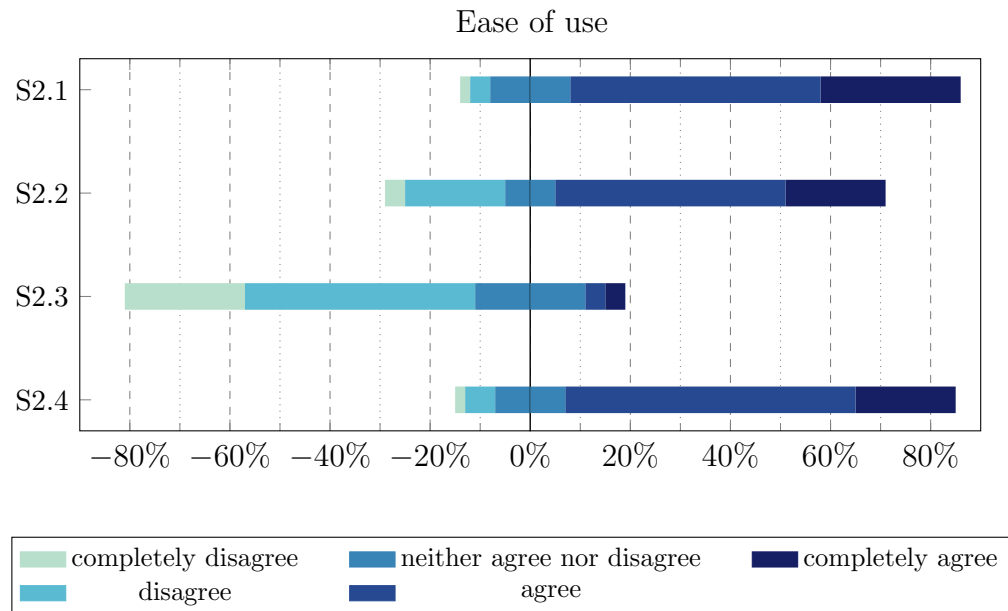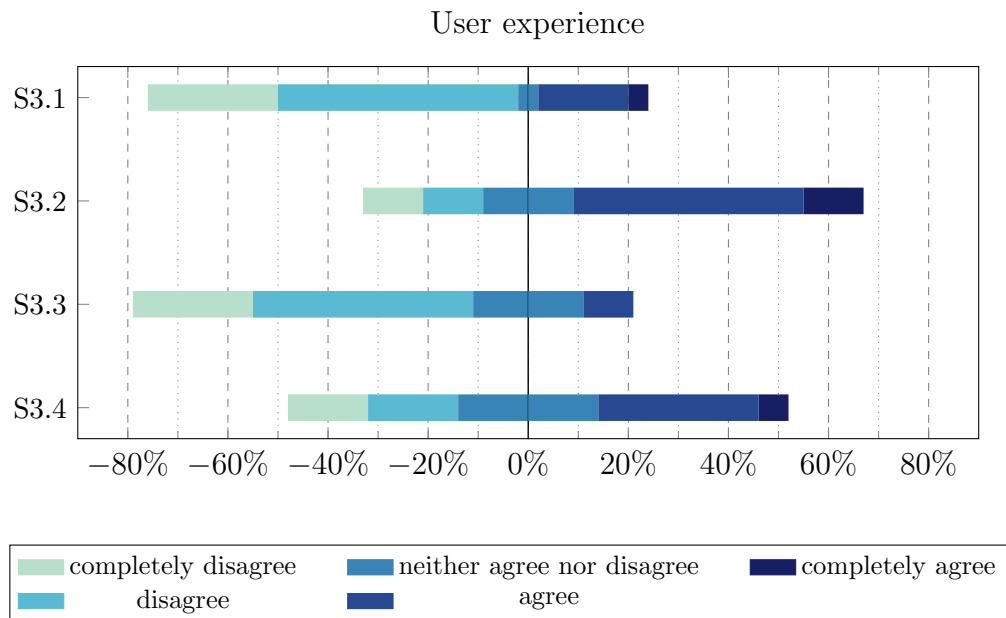

Figure 7.3: The results of the user experience category in the student questionnaire. The Likert items of this category are listed below.

S3.1 I received feedback from the system sufficiently fast

S3.2 I trusted that I could try out my code with the user interface returned by the system

S3.3 I received sufficient information about runtime exceptions

S3.4 I liked using the system

the submission page caused a new submission, which caused dissatisfaction with one respondent before the issue got fixed. One respondent mentioned also that UI did not always load when it was ready in the second and third assignment rounds. This issue was related to the image not being loaded before the canvas and the rest of the UI was drawn. One respondent also hoped that the GUI for sound manipulation had provided a simple undo functionality for filters and better functionality to make sure that the solution was correct.

Problems, Improvement Ideas and Useful Features

At the end of the student questionnaire there were three open-ended questions regarding any specific problems with using EDCAT, improvement suggestions and the most useful features. To the question "*If you had any explicit problems while using the system, what were they related to*" only seventeen respondents provided a reply. In addition to already mentioned issues related to runtime exceptions and performance time there was really no new usable information. One respondent claimed that the files sometimes got corrupted between the compilation server and A+. However, there is no evidence of this and errors that might have seemed related to corrupt files were caused by a student uploading the same Scala file twice or uploading the compiled class file. Another respondent said that "*pre-runtime exceptions (like NotImplementedError) thrown by Scala sometimes go to the web console*" instead of the pre-formatted box with compilation errors. However, NotImplementedError is thrown only after the method that has not been implemented is called after successful compilation at runtime.

The second open-ended question; "*What in your opinion should be improved in the system*" got similarly responses related mostly to runtime exceptions and performance time. However, the respondents left somewhat useful suggestions as well. One respondent suggested that there could be something closer to a debug UI with smaller data amounts, for example RGB values for an area of nine times nine pixels from which one could validate that an image filters works as expected. This suggestion is somewhat conflicting

with the goal that students should write their own tests and validate their program functionality themselves; the described suggestion is very close to what a grader application would do except giving points. Another student suggested that students should not be allowed to have more than one submission in the compilation queue at any given time, which would prevent unnecessary submissions and force students to examine the code they submit more closely.

Some respondents suggested replacing the system with local GUIs, which in their opinion would make "*testing faster and more helpful*". There were suggestions to add a console that is visible at all times and displays line prints and exceptions, or a commenting feature which could be useful when there are many submissions with varying level of functionality and one needs to find a specific one, or that when submitting the assignment with a pair also the other person could see it. Some respondents were displeased with separate submissions for grading and web-UI, and that the feedback forms for the assignment round were not better integrated in the system and they were unable to see if they had already filled the form or not. A few respondents wanted more openness in respect of how the system functioned, which would definitely have eliminated some confusion with the third assignment round.

The final question of the student questionnaire asked "*What functions and features were the most useful*". There were those who thought that the whole system was useless and those who thought that it was useful in general and had potential. The respondents mentioned testing in general, and a few especially with the sound manipulation assignment as well as the visual feedback they got while using web-UI and the interactive functionality. A few respondents thought that the most useful feature was that one could see their code in action as a part of some real program and that changing the code affected how the program functioned. Hiding the UI implementation and having multiple versions of one's code were similarly seen as useful features. One respondent apparently used EDCAT as a primitive version control and repository system while switching working stations.

7.2 Assignment Feedback Forms

7.2.1 Data Collection

With every turned in assignment students were required to fill in a short feedback form that mainly concentrated on evaluating if the assignment and its instructions were successful and to track how much time students used on each assignment. Additionally, students were asked to report which parts they finished and if they implemented any bonus functionality that should be taken into account while grading. At the end of the form students could leave free-form feedback regarding to the assignment and feedback or suggestions regarding to the submission procedure.

Giving feedback was mandatory for each assignment and the students who failed to leave feedback were given one point penalty. The field for submission procedure feedback was not mandatory and most students left it blank or filled with unrelated comments. However, possibly some students who did not participate in the student questionnaire were reached through the feedback form.

7.2.2 Results

The feedback forms were intended to catch issues related to EDCAT that could be addressed already during the course. For example, compilation time problems were brought up early on and worked on while the assignments were ongoing. Some problems with the A+ system, like the resubmissions on refresh, some guide texts and error formatting, were also corrected based on the student feedback already during the course.

Like in the student questionnaire, compilation time and the possibility for local execution were the most often raised topics. Similarly, students complained lacking the sufficient information about runtime errors and the operational principles of the system and wished for a console to see error messages and prints. There were of course those who had no problem with the system, and were happy to see their code in action.

The feedback for the first assignment round mostly showed how lost students were at the beginning of the course; the separate trial and grading submissions and even the feedback form were confusing for some. There were also some suggestions that were mentioned in the student questionnaire. One student suggested that the page featuring the web-UI should have a submission feature, so that the assignment could be submitted for grading immediately after one was convinced that it functioned correctly.

The feedback for the second assignment mentioned that the web-UI did not work with Safari web browser. This became apparent only after the assignment was published and the students were notified only at the beginning of the deadline week. Additionally, some students had problems with the implementation of their own filters as they did not know the used sample rate, the students of course did not think of to ask the course staff for this information.

A few students mentioned having problems with the web-UI in their own environments; for some the sounds slowed down and for some increased in speed. The students circumvent the problem by switching to computers provided in the computer lab. Some students mentioned that the play progress indicator animation at times was not in sync with the actual playback of a sound. The lag could similarly be caused by a slow processor that is unable to calculate the indicator's position fast enough even though the thread repeating the sound is uninterrupted.

For the second assignment, students had as well came up with UI-related suggestions. One student mentioned that perhaps it is undesired that the UI allows more than one sound to be repeated simultaneously. A few students hoped for a reset functionality so that the applied filter could be reversed and the sound returned to original with one button click. Now it required selecting another sound file before selecting the desired sound file again.

The feedback for the third assignment was unsurprisingly centred on speed related issues. Similarly unsurprisingly, the students had not considered trying to develop their testing skills as a good strategy after experiencing discontent with compilation times in the three previous assignments. Based on the feedback, many students solely relied on EDCAT for testing. However,

the students should not be blamed too roughly as the system was indeed unforgivably slow. In addition to speed issues, students raised the same issue as in the student questionnaire. It was much harder to pinpoint errors because all filters were applied to the image, even if it was fun to apply more than one filter to the image after having implemented all of them successfully.

7.3   Student Interviews

7.3.1   Data Collection

Out of fifty respondents for the student questionnaire, nineteen students volunteered for additional interview through contact information form. All volunteers were contacted by email and asked to select a time for an interview from given options. Finally, only eight students confirmed a suitable time and six interviews with one student and one interview with two students were held.

The interviews were unstructured but followed the same basic outline. Before starting the actual interview, the interviewees were told that the purpose of the interview was to confirm the results already gathered through the student questionnaire. They were notified that the interviews would be used as a part of this thesis and what topics would be discussed during the interview. Then, all interviewees had a chance to ask any questions and were asked for a permission to record the interview, which all of them gladly gave. All interviews were held in Finnish and all excerpts have been translated retaining the meaning.

The primary discussion was centred around four themes; programming experience, problem solving in general and related to EDCAT, the purpose of EDCAT, and the improvement of EDCAT. First the interviewees were asked to describe their programming experience and if it has been useful during the course. They were asked to describe what kind of problems they had encountered during the course and how they had gone on solving them. Then the focus was shifted toward EDCAT and interviewees were asked to define the purpose of the system. They were asked to reflect if the system had

helped them and if they used some other methods of testing their program. Next, interviewees were asked to describe if they had any problems with the use of the system or encountered exceptions and how they worked around those kinds of a situation. Finally, the interviewees were then asked for improvement ideas and some ideas for yet unimplemented functionality were offered to interviewees for evaluation.

7.3.2   Results

Out of eight interviewees only two did not have relevant programming experience. Four interviewees had programming experience in Python, of those one had learned the language on their own while other three had completed an introductory programming course taught in Python. Of those, one interviewee had also completed an introductory course in C and a project course in Python and had some experience in Java from high school and one had completed the *Programming 1* course and attempted to complete the *Programming Studio 1* course the previous year completing roughly the first half of the assignments. The last two interviewees had the most relevant programming experience. One of the interviewees had previously completed an introductory programming course in Java and the *Programming 1* course and the other interviewee had 10 years of programming experience overall and has been programming for work for the last three years including in Java.

All interviewees with relevant programming experience thought that their previous experience was useful and helped to understand the problems in the assignments. The interviewees reported that even though Python experience was useful in the beginning, this course together with *Programming 1* quite early on exceeded their programming knowledge especially in regard to object-oriented programming. The interviewee who had previously attempted to complete the course thought that this time round it was easier to estimate how long it would take to complete the assignments and work on them without teaching assistants' help.

When asked what kind of problems the interviewees had had and what they felt was difficult, most interviewees mentioned getting started on the

assignment and understanding the big picture based on the assignment in-
structions. A few interviewees mentioned that it is difficult to find help when
starting on a new kind of problem while one interviewee thought that it is
difficult to understand the code that is provided as a part of the assignment.

> '– – if there is some ready-made code given, then of course when
> you are not used to reading others' [code], then especially in the
> beginning it took more time [to read and understand the code] –
> – [in the very first assignment] I was like "what does this mean?
> what is this?". . . even though you didn't actually need to use [the
> code], it was in itself very confusing that there were some other
> [code]. . . "do I need to do something with this?".'

To solve the problems, most interviewees had turned for help to teaching
assistants and friends although it was not necessarily their first means of
finding help. One interviewee, for instance, thought that it is not so useful
to straight away ask for help and that one learns more while trying to
solve a problem on their own. Reading the relevant documentations and the
assignment instructions as well as solutions for previous assignments alongside
the currently problematic code were also popular manners. One interviewee
mentioned using the rubbed duck technique, while another mentioned that a
clear error message is often useful and sufficient. However, the interviewee
regretted having poor debugging skills and wished that they would have had
more practise in the area. Finally, one interviewee mentioned using a pen and
paper for sketching the problem and searching the web with a search engine.

Most interviewees thought that EDCAT's purpose was to ensure that
the final solution for the assignment was more or less working correctly and
realised that using only EDCAT was not enough to extensively test their
program.

> 'the purpose was to simplify testing the code, in a way it is the
> most important thing. . . to hide the details of the UI code. . . you
> only submit the functionality in a few files and hopefully it works'

They thought that EDCAT had been useful. Some of them wrote their own unit tests or tested their solution with REPL. The interviewees thought that EDCAT was especially useful with the second and third assignment rounds. In the second assignment, one could compare their own solutions with the reference and in the third assignment it was useful to see the actual image and judge the result visually. Additionally, one interviewee mentioned that the system helped developing suggested bonus functionality.

The interviewee who had tried to complete the course the previous year had had difficulties with setting up references to sound files for the local GUI program last time and thus thought EDCAT helped with that issue. However, the interviewee thought that local GUI allowed more flexibility and did not tie the development to an internet connection.

A few interviewees had a different idea about the purpose of EDCAT. One thought that the system simulated the situation when one does not have access to all the code for a project and only knows the interface one needs to meet. Another interviewee trusted the system to advise if the solution was correct. The interviewee relied on the system as a sole method of testing and thought that it was not useful. The system did not reveal where the problems and errors lay and forced them to rely on the help of teaching assistants.

The problems related to the use of EDCAT or encountered exceptions were minimal. The interviewees mentioned the speed of the system and the difficulty of finding the errors as the biggest problems encountered while using the system. One interviewee confessed to accidentally taking down the server, luckily well before deadline. However, there is no evidence of this ever happening. The most noticeable error messages were the result from compilation errors but one interviewee mentioned also seeing some error messages in the GUI. Some error messages were shown in the second assignment round as presented in Chapter 6.

When asked what things should be changed or improved if EDCAT is used in the future, the interviewees again brought up speed and debug functionality. They wished that error messages would be visible and not hidden in JavaScript console. This console could be useful also for elementary print line checks. Similarly as in the questionnaire, some interviewees were confused about

points not showing and two submission areas; they hoped the submission could be streamlined and points shown.

Some interviewees thought that the answer could lie in better directions on functioning and the use of the system. The operational principle of the system is quite unique; like nothing students have encountered before and different from the work methods used in *Programming 1* that most students are taking simultaneously with *Programming Studio 1*. Additionally, as the system is not intended to test the students' solution fully, there should be more support for writing good test programs.

One interviewee said that necessarily nothing should be changed and another agreed noting that the system is as such already useful for the beginner. However, the interviewee continued, a more experienced programmer would probably want a local implementation to have a better possibility to study the functioning of their solution more closely. The interviewee, wished that there would have been a possibility to observe how the values in sound and image data changed after applying the filters. Another interviewee suggested that in the GUI for the third assignment all image filters should not be applied every time and could be turned on and off. Finally, an interviewee had a suggestion that one should be able to better compare and see the changes between one's own submissions, possibly similarly as between commits on *github*.

For functionality that was not implemented for this version of EDCAT, we had four ideas to test. The first idea was to add functionality for commenting on one's own submissions. This would allow private discussion about problems with teaching assistants and adding personal notes for different solutions to keep track of changes. The interviewees thought that it should not be a separate system but better integrated with Piazza that is already in use. Some interviewees thought that commenting could at some level be useful, especially if well implemented and it would be easy to clearly reference specific rows. However, a few interviewees noted that even though if the system better supported interaction with teaching assistants, the face-to-face help that one gets is still irreplaceable.

The second suggestion was to add the possibility to view the instructor's solution without showing the code while the assignment is still due. The

interviewees were not especially exited about this possibility. About half interviewees thought that there would be no harm and the other half did not see any use in it. As one interviewee noted, usually one knows what the desired outcome is and it is often more useful to think for oneself than to blindly compare with the correct solution.

Another idea was to encourage interaction between students by allowing to similarly view other students' solutions without seeing the code before the final deadline. This idea got similarly little support. Although one interviewee thought that if combined with possibility for commenting it could help students who have similar problems. Another interviewee thought that this functionality could be useful only when bonus functionality is considered and other students' solutions are not interesting. The general opinion seemed to be that possibly the code of more advanced students could be interesting to see after deadline has passed.

Finally, we tested an idea of having a possibility to download one's full solution for use in for example a portfolio. The interviewees thought that it would not be very useful although nor harmful either. The amount of code produced by students is quite minimal and the problems very trivial for the downloaded project to showcase anything interesting for possible employers.

7.4   Summary

In conclusion, all three sources of student feedback provided a uniform perspective into students' views. EDCAT was generally seen useful as it helped to showcase the functionality of one's code as well as hide the GUI code that was irrelevant to the solution of the assignment. However, EDCAT failed to support debugging because runtime error messages were not clearly visible. Additionally, the performance in regard to the speed and efficiency of EDCAT was weak.

The students had very few problems with the use of EDCAT and they were mostly related to the general use of the system and not the assignment GUIs themselves. We had decided to have separate submissions for trial and grading submissions and award half points in the first assignment round as

well as bonus points in all rounds meant that we could not easily use the point system built in A+ system, which proved confusing and annoying to some students. Furthermore, the page did not update with the JavaScript GUI when the compilation took long and it required the student to refresh the page from time to time.

Many improvement suggestions indicated that students saw EDCAT more like a grader that would clearly tell if the solution was correct and where the problem areas are. Additionally, one common improvement suggestion, especially from more experienced students, was that the system should be more transparent and local GUIs would altogether be a better solution than EDCAT. However, there were also some suggestions that are definitely worth looking into in the future. For example, the submission procedure indeed is not optimal and should be streamlined and the suggestion to let students have only one submission in the queue for compilation would reduce the stress on the system.

The interviews suggested that perhaps problem solving techniques and testing should be better supported and formally taught. EDCAT in some ways simplified the trial runs of the code but if one could not find the console the system was merely confusing and did not live up to the service provided with a local GUI. Additionally, commenting and possibly sharing features should be explored in the future.

Chapter 8

Teaching Assistant Questionnaire

8.1 Data Collection

The questionnaire for teaching assistants was conducted at the very end of the Programming Studio 1: Media Programming course. All thirteen of course's teaching assistants, who were involved in helping students with the programming assignments and grading the assignments, provided their response to the questionnaire. You can find the teaching assistant questionnaire in Appendix B.

First, the teaching assistants were asked if they had previously worked in a course where their tasks included grading students' programming assignments. Upon a positive reply, they were asked to evaluate if EDCAT had helped them and how, or if perhaps it had caused more work compared with the courses where it is not used.

The rest of the questionnaire was organised around two themes; *assisting students with assignments* and *grading students' assignments*. Both were presented as a checkbox grid where for each presented statement the respondents were asked to tick the boxes for those assignment rounds for which the statement holds true. Both statement groups were followed by an open-ended question for commenting on any of the statements or the previous theme in

general. Finally, at the end of the questionnaire there was one open-ended question for any comments regarding EDCAT.

8.2 Results

Out of thirteen respondents, only three had previous experience with grading programming assignments. Coincidentally, at least some of their experience comes from the previous iterations of *Programming Studio 1* course, where the first three assignments have been very similar in the previous years. One of the respondents found that EDCAT helped form a general idea about the level of functionality of the student's solution. The remaining two, thought that EDCAT caused them extra work. One of them mentioned that there is at least one more step to try student's solution, as the code first needs to be uploaded to the server; the other felt that when the student's solution was greatly defective, online GUI made it more difficult to find the bugs compared with a locally run GUI.

8.2.1 Assisting Students with Assignments

The first statement category, *assisting students with assignments*, consisted of five statements; the first four were set to map out how teaching assistants used EDCAT while helping students to solve their assignments and the final statement asked to estimate if online testing GUI was useful while helping students. All statements, translated into English, and the summary of responses can be seen in Figure 8.1.

Ten respondents reported both utilising EDCAT while tutoring students at computer lab sessions as well as running student's program code together with the student in all the first three assignment rounds. One respondent agreed with the first two statements in regard to only the second and third assignment rounds, which leaves two respondents who both agreed with one of the statements on only one round; one respondent reported utilising EDCAT while tutoring students at computer lab sessions in the first assignment round, and other reported running student's program code together with the student

Assisting students with assignments
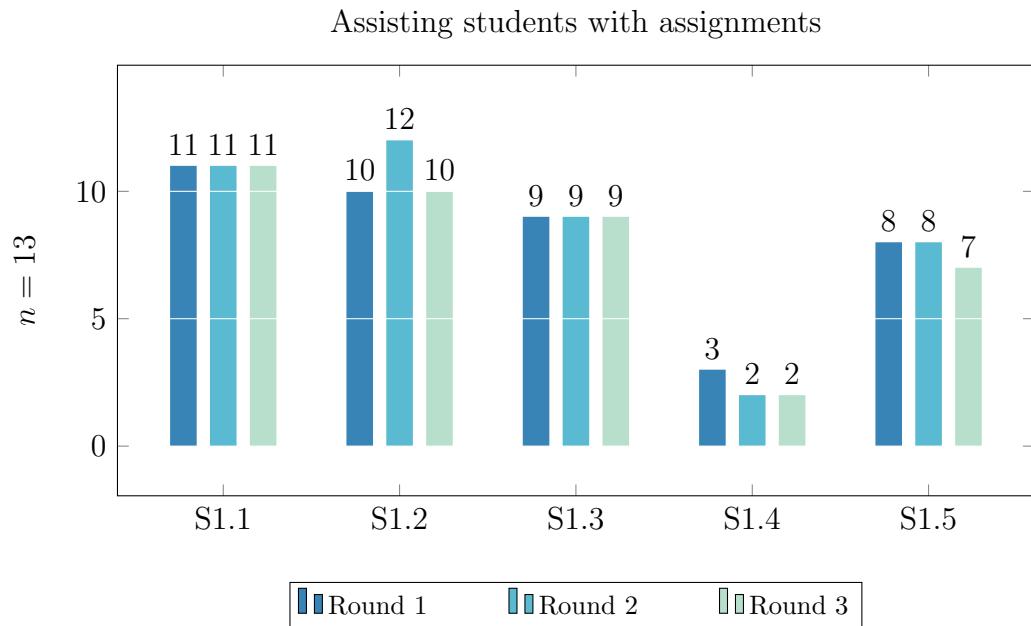
Figure 8.1: The amounts of positive answers to the below statements from the teaching assistant questionnaire.

S1.1 I utilised online testing GUI while tutoring students at computer lab sessions.

S1.2 We ran student's program code together with the student.

S1.3 I ran student's program code alone.

S1.4 I utilised online testing GUI while tutoring students via IRC or Piazza.

S1.5 I think that online testing GUI was useful while assisting students to solve programming assignments.

in the second assignment round. For the third statement, where respondents were asked to state in which assignment rounds they ran students' program code alone, nine respondents provided an affirmative response for all the first three assignments rounds.

The first three statements in the first statement group were especially intended to estimate if EDCAT is a useful tool while helping students to solve their assignments at lab sessions. However, the theme of the questions was stated clearly only in the open-ended question at the end of the statement

group which left room for the misinterpretation of especially the second and the third statements. The responses provided for the first three statements in this group are somewhat contradictory considering the intended meaning of the statements and do not necessarily provide a truthful picture about the extent of use of EDCAT at programming lab sessions.

Only three respondents reported using EDCAT while remotely tutoring students via IRC or Piazza; of those, two used the system in all three assignment rounds, and one only for the first assignment round. Statistics in Piazza show that only two of the previously mentioned affirmative respondents contributed to students' questions in Piazza. Both provided their help in the first assignment round with issues that did not require accessing students' submissions. Additionally, Piazza's statistics revealed two active teaching assistants, who did not report utilising EDCAT while tutoring remotely. It is thus possible to draw a conclusion that EDCAT was mostly utilised while helping students through IRC, which is a fair assumption as Piazza has a feature for uploading files and including code snippets is much easier in comparison with IRC, which is strictly text based. Finally, both respondents who used EDCAT in all three assignment rounds to remotely assist students, recognised EDCATs utility in remote tutoring.

Finally, concluding the *assisting students with assignments* statement category, seven respondents think that EDCAT was useful for all three assignment rounds while assisting students with their assignments, while four respondents did not agree at all with the final statement. This leaves two respondents of whom one found that EDCAT was useful in the first and the other respondent in the second assignment rounds.

The responses to the open-ended question at the end of the statement category reveal a generally positive attitude towards EDCAT. However, the system fell short of expected in regard to error messages and performance. The defects in students' program code were not always easy to find using only EDCAT as error messages were printed out in the JavaScript console and were not equivalent to those in Scala. Similarly, issues regarding compilation time and the consequent waiting time constituted a situation where debugging was forced to local environment and specifically REPL. As previously mentioned,

EDCAT's utility in remote tutoring was recognised by most assistants who helped students through IRC and Piazza. Equally, a few respondents estimated that the system could have proved useful if they had helped students remotely.

8.2.2   Grading Students' Assignments

The second statement category, *grading students' assignments*, consisted of six statements. The first five statements were set to assess if teaching assistants, while grading, ran students' code locally with GUIs or test classes, or online with EDCAT. As in the first statement category, the final statement asked to estimate if online testing GUI was useful while grading students' program code. All statements, translated into English, and the summary of responses can be seen in Figure 8.2.

Nearly all of the respondents reported downloading students' program code for grading purposes. In the first round, one of the assistants did not participate in grading due to schedule restrictions, and thus did not need to download students' code. However, one of the respondents successfully managed to grade students' submission without downloading solutions in the first and second assignment rounds. The first two assignment rounds were quite simple, consequently most of the mistakes were revealed just by reading the code. Additionally, the GUI in the first two assignment rounds supported error detection quite well.

For grading purposes, the teaching assistants were provided with example solutions including the original GUI Scala code written with Scala.js. Only two respondents used local GUIs for grading; one in all three assignment rounds, and one in only the first assignment round. Some assistants did not manage to correctly configure *sbt* for their environment while others did not bother at all. Nearly all respondents utilised local test classes while grading; most in all of the first three assignment rounds, while only one in none of the assignment rounds.

Out of thirteen respondents only two reported that they did not at any point submit students' solutions through A+. Four respondents systematically

Grading students' assignments
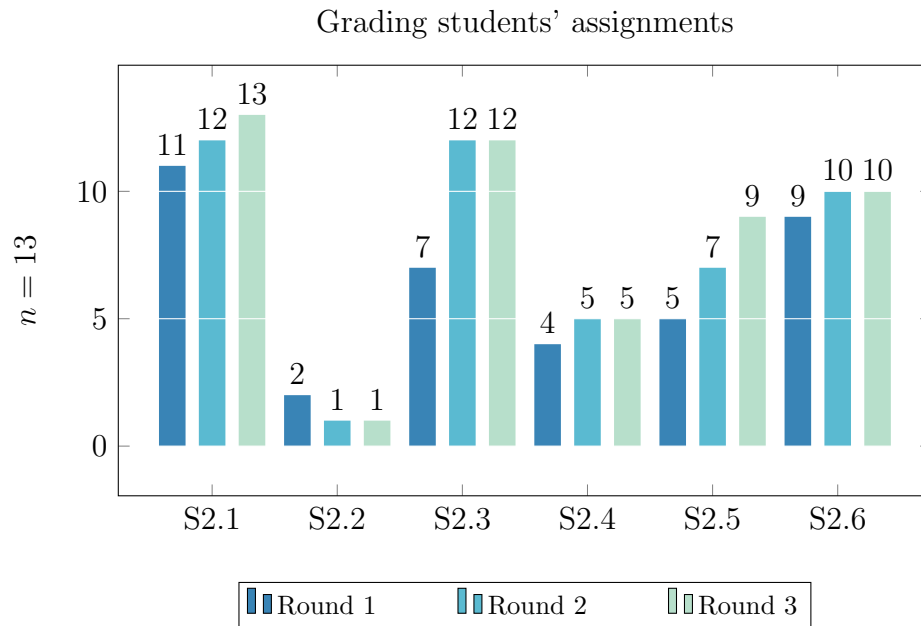
Figure 8.2: The amounts of positive answers to the below statements from the teaching assistant questionnaire.

S2.1 I downloaded students' program code to my own computer for grading.

S2.2 I ran students' program code locally on my own computer with a GUI.

S2.3 I ran students' program code locally against test classes.

S2.4 I submitted through A+ student's unaltered programming code to online testing GUI and tried the code there.

S2.5 I submitted student's altered (for example, I fixed bugs) program code through A+ to online testing GUI and tried the code there.

S2.6 I think that online testing GUI was useful while grading students' program code.

submitted students' unaltered solutions through A+, of whom two submitted also students' modified solutions in all three assignment rounds. Otherwise, there was no easily detectable pattern in how the respondents used A+ for grading students' assignments. It is clear that there were more submissions of students' altered solutions and the number grew as the course progressed and assignments became more complex.

The view of all submission in A+ is searchable, and there is thus no necessity to upload students' code before trying it with online GUI if final solution was identical to one of the trial submissions. However, if one cannot easily see that one of the trial submissions is identical to the final one, if for example the student added comments to the code only after it appeared to function correctly, it is just easiest to submit the student's solution again. The increase in altered submissions is most likely related to the increase in the complexity of the assignments, and consequently increase in erroneous submissions. In order to identify the errors accurately, the assistants needed to correct the solutions and submit them through A+ to online testing GUI more often.

Finally, as a final statement in the *grading students' assignments* statement category, nine respondents think that EDCAT was useful for all three assignment rounds while grading students' assignments, while three respondents did not agree at all with the final statement. This leaves one respondent who found that EDCAT was useful in the second and third assignment rounds.

The general opinion, drawn from the responses to the open-ended question at the end of the *grading students' assignments* statement category, is that online testing GUI was quite useful when student's solution was more or less correct but did not provide enough support for analysing and correcting errors. One respondent found the online testing GUI especially useful in the second assignment round where it was possible to compare the waveform of a sound filtered with student's code against a filtered sound provided with the online testing GUI. Additionally, one respondent noted that online testing GUI provided a good platform for testing bonus functionality, which could not have been covered in tests written prior to grading.

However, if there were many or severe faults in student's solution, which could even crash the online testing GUI, correcting the errors and ensuring that the alterations would fix the problem required a new submission. This took of course more time and effort than if the code would have been executed locally; the teaching assistants were not able to use the debugger and the compilation was quite slow on the server. Consequently, some respondents thought that there was no significant benefit to having the testing GUI online

compared with situation where the testing GUI would have been distributed as a locally executable project. Of course, the locally executable project containing the GUI was distributed to the teaching assistants in each of the assignment rounds, but as previously mentioned some of the assistants did not even bother to try and configure *sbt* in order to execute the project.

8.2.3 Overall Feedback

The respondents submitted encouraging feedback through the comment section at the very end of the questionnaire. Although they found also features that should be addressed, should the system be taken into use later on. Perhaps the biggest shortcoming with EDCAT was the long compilation time and the consequent queues. The students worked on and submitted the assignments mostly at the same time during computer laboratory sessions near the submission deadline, which of course aggravated the situation.

There were also some specific issues in the implementation of the second and the third assignment rounds' GUIs. In the second assignment round, one of the model waveforms was not exactly the same as a waveform produced by a correctly implemented filter, which caused some confusion. Similarly, in the third assignment round, the design choice to have all image filters to be applied to the picture with each filter modification caused the GUI to crash if any of the filters had an error. This made error detection difficult, and at times it seemed that the GUI just was not responding. Additionally one assistant pointed out that the sample image was square, which made one aspect of multidimensional arrays unclear.

Furthermore, the respondents found shortcomings not related to any specific assignment round. As previously mentioned, the compilation to JavaScript resulted in error messages that were not equivalent to those in Scala and the reference to where in code the error had occurred was often lost. One respondent thought that the system should have supported better printing debug messages, a method of debugging that is often preferred by novice programmers. On the other hand, another respondent saw that students relied on too much on EDCAT. Online testing often was their only means of

ensuring the functionality of their program code, while many of the mistakes would have been detectable by a few tests through REPL and the related faulty submissions, which added to waiting times, avoidable. Additionally, one respondent thought that students could have benefited from viewing different kind of GUI code implemented from the beginning of the course as they need to build a GUI in their final assignment round.

Finally, one respondent reported that many of the students thought it was neat that one's work was visible in action in-browser. A few respondents recognised that it was useful that the GUI could be updated after it had been made available without the students needing to download new source code. Similarly, EDCAT had a clear advantage compared with a traditional instructor prepared GUI in remote tutoring. However, the respondent noted that perhaps the system should had only been used for remote tutoring and students be provided with a test suite for debugging. All in all, one respondent noted that from purely the assistant's point of view the system was nice.

8.3   Summary

Ultimately, EDCAT was somewhat useful for teaching assistants, for both when helping students to solve assignments and when grading their submitted work. In computer lab sessions EDCAT did not have any advantage over having the GUI provided locally; The system underperformed and did not show error messages clearly. However, those teaching assistants that helped students via IRC or Piazza really benefited from having a quick access to students' submissions.

When grading practices are considered, EDCAT had a clear advantage over the locally run GUIs; When student's submission functioned correctly or relatively correctly, there was no need to download the code and run it locally. However, EDCAT did not entirely eliminate the need for teaching assistants to download the code. Nearly all teaching assistants ran local tests in order to better evaluate students' code.

During the grading process, the teaching assistants were subjected to the same shortcomings of the system as were the students. If there was a

need to resubmit student's code, the compilation time was naturally longer than it would have been with a local GUI. In each round assistants were provided with code to locally execute students' code, however very few of them bothered setting up the tools to use the local distribution.

Chapter 9

Usage Data Analysis

9.1 Data Processing

As a material for this section, Celery task log, the inspection of A+ submissions and grading information were used. The Celery log file, which contained over one hundred thousand rows of data related to submissions during the course, was first divided into four files that each included submissions to one assignment. The logs were stripped of unnecessary information leaving only lines with submission ids, compilation times and error information. The stripped data files contained only one row of data per submission and were further processed with Excel, which allowed the calculation of average compilation times.

The log entries were matched with submissions from the A+ system, which added incomplete submissions that were not shown in server data as well as matched submissions to a specific person. All teaching assistants' and instructor's submissions could at this point be removed from the data to demonstrate only the students' use patterns. Additionally, matching submissions with the responsible students made it possible to analyse the number of students' submissions and the dates that students started working on the exercises. The students who did not have any submissions to the second and third assignments, which were a requisite to pass the course, were at this point removed from the data.

The grading information, which logged pair submitters, was used to recognise the pairs and the points and grades were added to the data as well. When submissions to an individual assignment are considered, pair submitters are reduced to one entity with shared submission number, starting time and grade. If the points or grade between members of the pair differ, the average is used. Additionally, there were four pairs that completed all four assignments together, those pairs are treated as entities also when the combined data from all four assignments is handled.

9.2 Results

Submission statistics that are discussed in the following section show the submissions of altogether one hundred and sixty students. First, the compilation times and the number of submissions for each assignment are presented in Section 9.2.1. Then we show the distribution of all submissions on a timeline and highlight students' first submission to each assignment in Sections 9.2.3 and 9.2.4 respectively. Finally, the number of submissions in relation to reached points are presented in Section 9.2.2 for each assignment round and the whole course.

9.2.1 Compilation Times and the Number of Submissions

The number of submitters and submissions to each assignment round have been compiled to Table 9.1. Submissions have been categorised into successful, failed or incomplete submissions. Successful submissions are those that were successfully compiled but could have functioned incorrectly and even produced a runtime exception. Failed submissions returned compilation error and incomplete submissions did not contain all necessary files for compilation.

As the course progressed and the assignments grew more complex, the students used more trial submissions. It is possible that also the increasing options for bonus features and growing programming experience affected the number of submissions. For example the assignments in the first round

		A 1.1	A 1.2	A 2	A 3
	Successful	445	920	2566	2767
Number of	Failed	71	56	130	73(6)
submissions	Incomplete	9	10	14	12
	All	525	986	2710	2858
Number of submitters		151(4)	150(5)	147(13)	137(13)

Table 9.1: The number of successful, failed and incomplete submissions to each assignment. The number inside parenthesis for the third assignment's failed submissions is the amount of timed out submissions. The number of submitters is shown for each assignment, the number of pair submitters is shown inside parenthesis.

Compilation times (s)		A 1.1	A 1.2	A 2	A 3
	average	60,55	57,57	32,90	54,01
Successful	minimum	26,95	28,42	26,86	44,28
	maximum	96,55	97,18	88,15	277,15
	average	26,51	34,31	17,96	11,01
Failed	minimum	19,13	18,31	14,20	2,22
	maximum	30,84	46,49	27,53	114,80
All	average	55,87	56,23	32,18	52,90

Table 9.2: The average, minimum and maximum compilation times divided for successful and failed submissions to each assignment.

did not differ that much in difficulty, but the assignment 1.2 had much more complex and indefinite bonus exercise of implementing an elementary artificial intelligence for the automated opponent. It is also notable that more students chose to complete the latter assignments with a partner. The ratio of successful to failed submissions, on the other hand, does not reveal any clear trend.

The analysis of compilation times for each assignment is presented in Table 9.2. Again, the successful and failed submissions have been treated separately. The incomplete submissions do not of course have a compilation time, as they did not reach the server and resulted in an error message shown right away to the student. In addition to the average compilation time, the minimum and maximum successful and failed compilation times have been found.

The compilation times in the first assignment round were far from optimal considering the relative simplicity of the resulting code. However, as there were not that many submissions and long queues did not form, the overall waiting time was reasonable for an average student. The compilation script was tweaked for the second assignment round and most of the background code was compiled to a jar-package to shorten the compilation time further. This had a clear positive effect on the compilation times, but the failed compilation still took too much time.

For the third assignment round another script was added that attempted to compile the submitted files with *fsc* before they would be compiled to JavaScript, which brought the compilation time for failing submissions to a desired level. However, the attempts to shorten the compilation time for successful submissions failed. Nearly all of the compiled code was produced by students and packaging the background code did not provide any help like in the second assignment round. This caused major problems towards the end of the assignment round as there were many submissions and the queue grew as many as to 37 submissions resulting in over than half an hour waiting times at the worst.

9.2.2 Number of Submissions per Student

The number of submissions were compared with the points received from the assignment round. The results are pictured in Figure 9.1. For the assignment 1.1 most students used four or less trial submissions with good results. But as already shown in Table 9.1 there were more than twice the amount of submissions to the assignment 1.2 and most students managed with small number of submissions of eight or less. However, there were clearly more students who iterated their solution most likely working on the bonus exercise.

There were more submissions per student for the second and third assignments than in the first assignment round, but still most students managed with ten or less submissions. Figure 9.1 reveals some students who had a significant number of submissions. Probably some of them had difficulties while others worked on the bonus filters. Additionally, there is somewhat

more dispersion between submission numbers and more students received slightly lower points for the third than for the second assignment round.

The comparison between the number of overall submissions and the final grade shows that students who reached low passing grades had on average less submissions than those who reached higher grades. Students with grades one and two used on average around thirty submissions while students reaching higher grades of three and four used on average around fifty submissions. For the highest grade of five students used on average as many as sixty
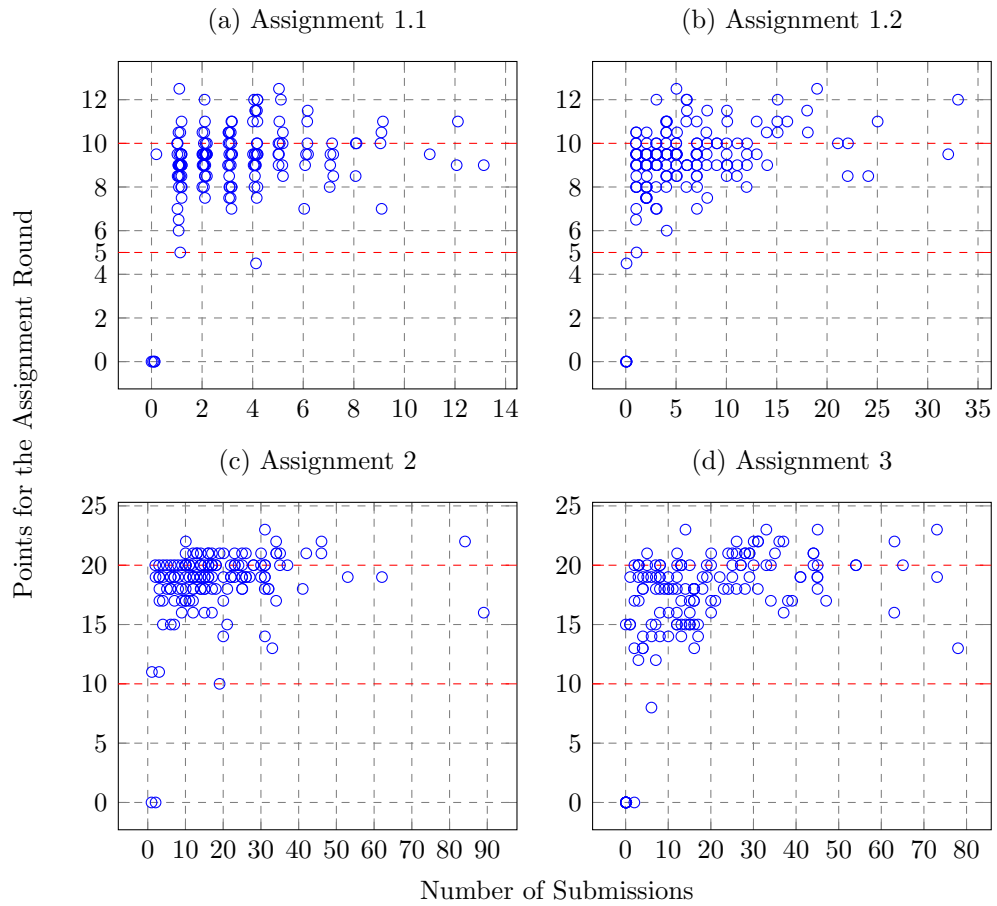

Figure 9.1: The number of individual students' or student pairs' submissions compared with the points reached during the assignment round. Some jitter has been added to improve readability of the figures. The two horizontal lines indicate the minimum points to pass the assignment and the maximum points awarded without bonus functionality.
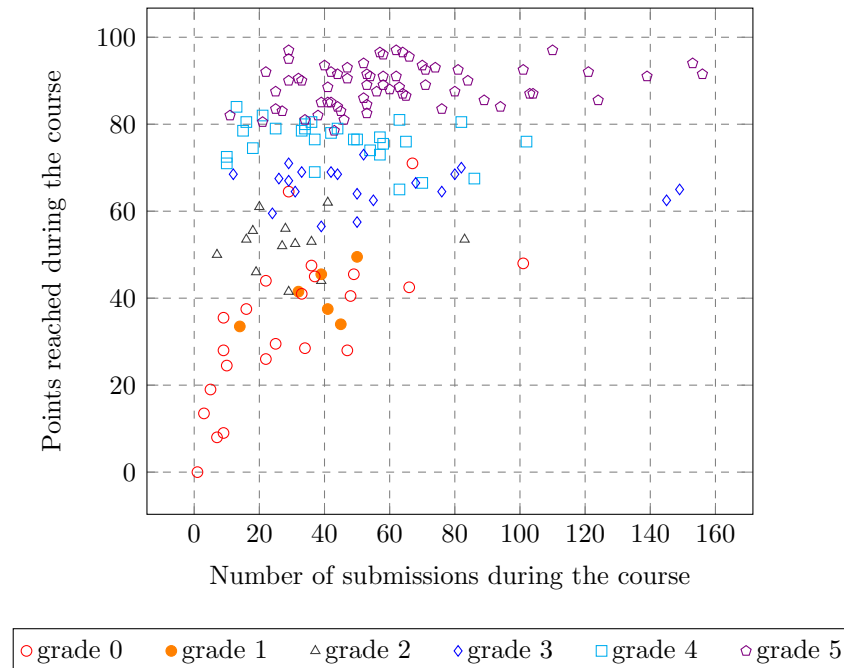
Figure 9.2: The number of individual students' or student pairs' submissions to the whole course compared with the points reached during the course.

submissions. Additionally, there were some outliers who had significantly more submissions than other students who reached the same grade.

9.2.3   Distribution of Submissions

The analysis of the submissions, which are pictured in Figures 9.3 and 9.4, unsurprisingly revealed that students tended to work on the assignments during the daytime and near the deadline. There is a clear increase in submission frequency for assignments 1.1 and 1.2 after Wednesday 12 p.m before the submission deadline on Friday, which is most likely explained by the fact that the course *Programming 1* had its deadlines on Wednesdays at noon. However, for the second and third assignment rounds the distinction is not apparent. Possibly the increasing difficulty of the assignments encouraged some students to try and start earlier, like the course staff suggested throughout the course, and overall increased the number of submissions, so students stayed working on the assignment longer also well before the deadline.

(a) Assignment 1.1
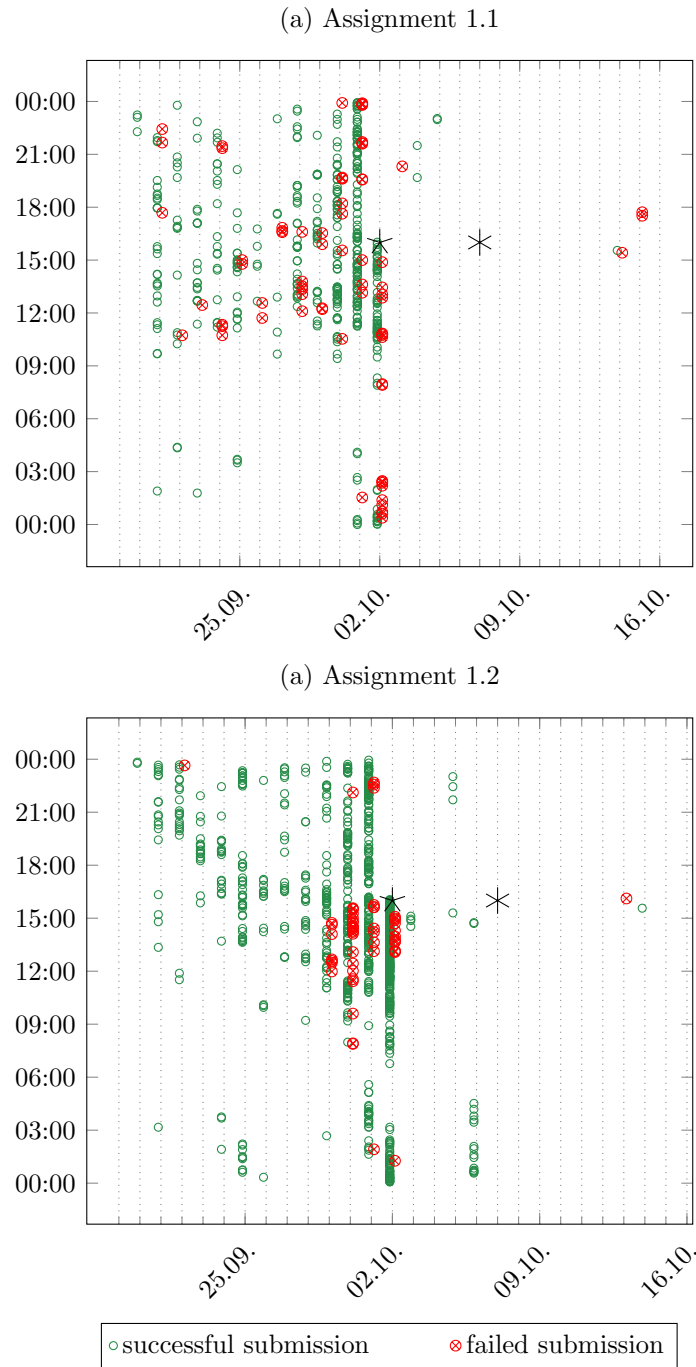
(a) Assignment 1.2

Figure 9.3: All students' submissions to the assignments 1.1 and 1.2. The star marks the deadline, which was on the 2nd of October at 4 p.m. and the asterisk marks the late submission deadline, which was on the 7th of October at 4 p.m. The submissions after the hard deadline came from students who did not reach the minimum points to pass the exercise and were obliged to work further on their assignment. There were 516 submissions to the assignment 1.1. and 976 submissions to the assignment 1.2.

(a) Assignment 2

(b) Assignment 3

○ successful submission   ⊗ failed submission   △ timed out submissions
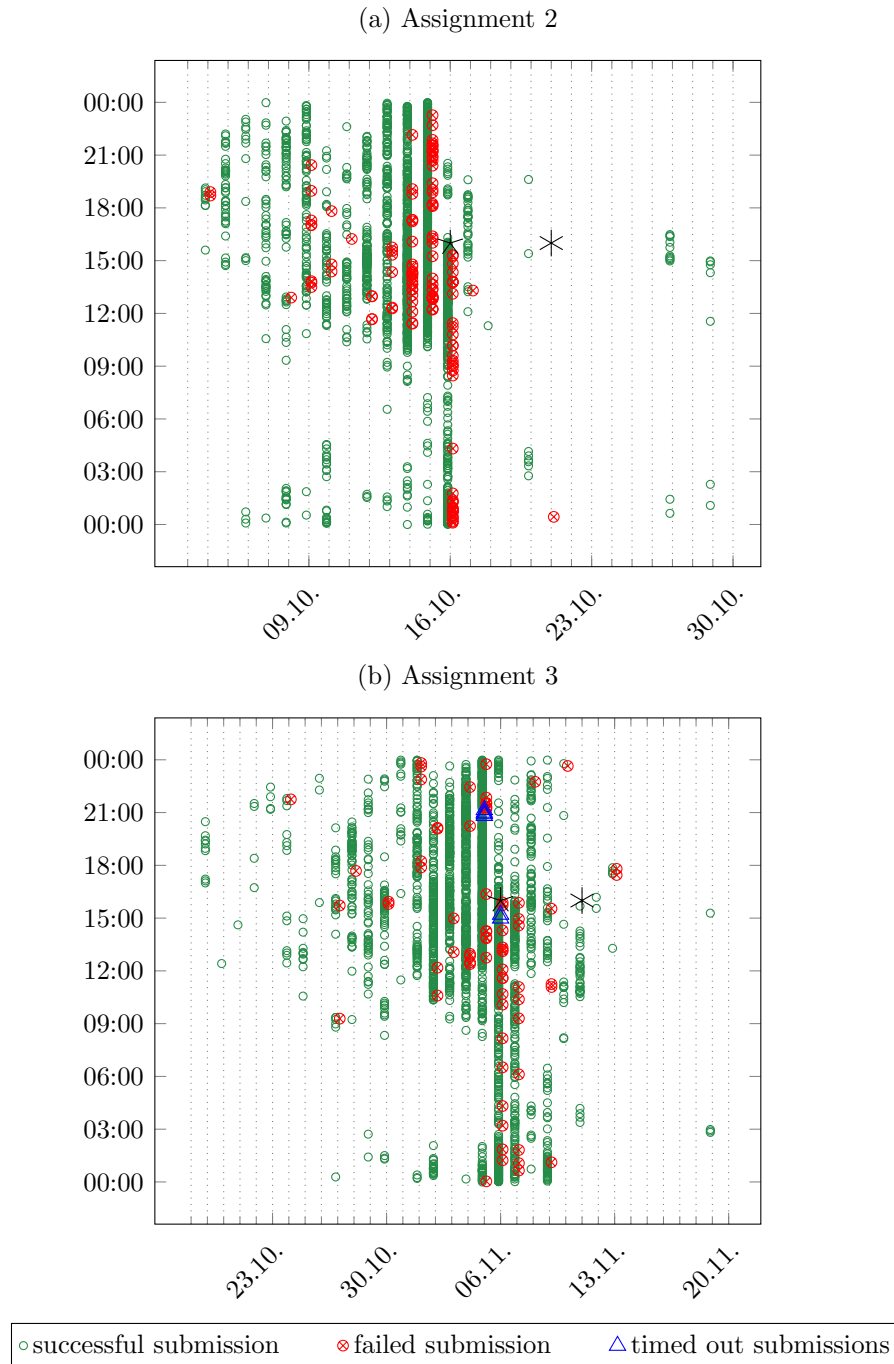
Figure 9.4: All students' submission for the second and third assignment rounds. The star marks the deadline, which was on the 16th of October at 4 p.m. for the second and on the 6th of November at 4 p.m. for the third assignment. The asterisk marks the late submission deadline, which was on the 21st of October at 4 p.m. for the second and on the 11th of November at 4 p.m. for the third assignment. The submissions after the hard deadline came from students who did not reach the minimum points to pass the exercise and were obliged to work further on their assignment. There were 2696 submissions to the second and 2846 submissions to the third assignment round.

Interestingly, the distribution of failed submissions to assignment 1.2 is also centred around last three days before the deadline whereas for assignment 1.1 the failed submissions were distributed quite evenly. A quick analysis of the failed submissions to assignment 1.2 revealed that twenty-seven students had failed submissions; most of them had one or two and at most four failed submissions while one student was responsible for twenty-three failed submissions, which is just a bit over a third of all failed submissions. A quick look through some of that student's submissions revealed that the student had on more than one occasion submitted the failing file multiple times in a row without any alterations.

The analysis of submissions to the second assignment round in Figure 9.4 (a) shows that most of the failed submissions occurred during the last three days prior to the deadline. This time analysis of failed submissions did not reveal any clear culprit behind failed submissions, although there were two students who were responsible for roughly eight and seventeen percent of failed submissions each. However, as there were a hundred and thirty failed submissions to the second assignment round from fifty-nine students in all, failed submissions were quite evenly distributed among the students.

There were significantly less failed submissions to the third assignment than to the second assignment. The failed submissions are quite evenly distributed through out the time that the assignment was open and between students who had failed submissions. Naturally, most of the failed submissions are near the submission deadline. Additionally, the third assignment round had six submissions that eventually timed out due to long compilation times and a long submission queue.

9.2.4   First Submission

Figure 9.5 reveals that significant number of students started working on the assignment only after the deadline for *Programming 1* had passed. There is also a slight indication that those who began working on their assignment closer to the deadline received more often less points than those who started on the assignment right when it opened. Unsurprisingly, students had first
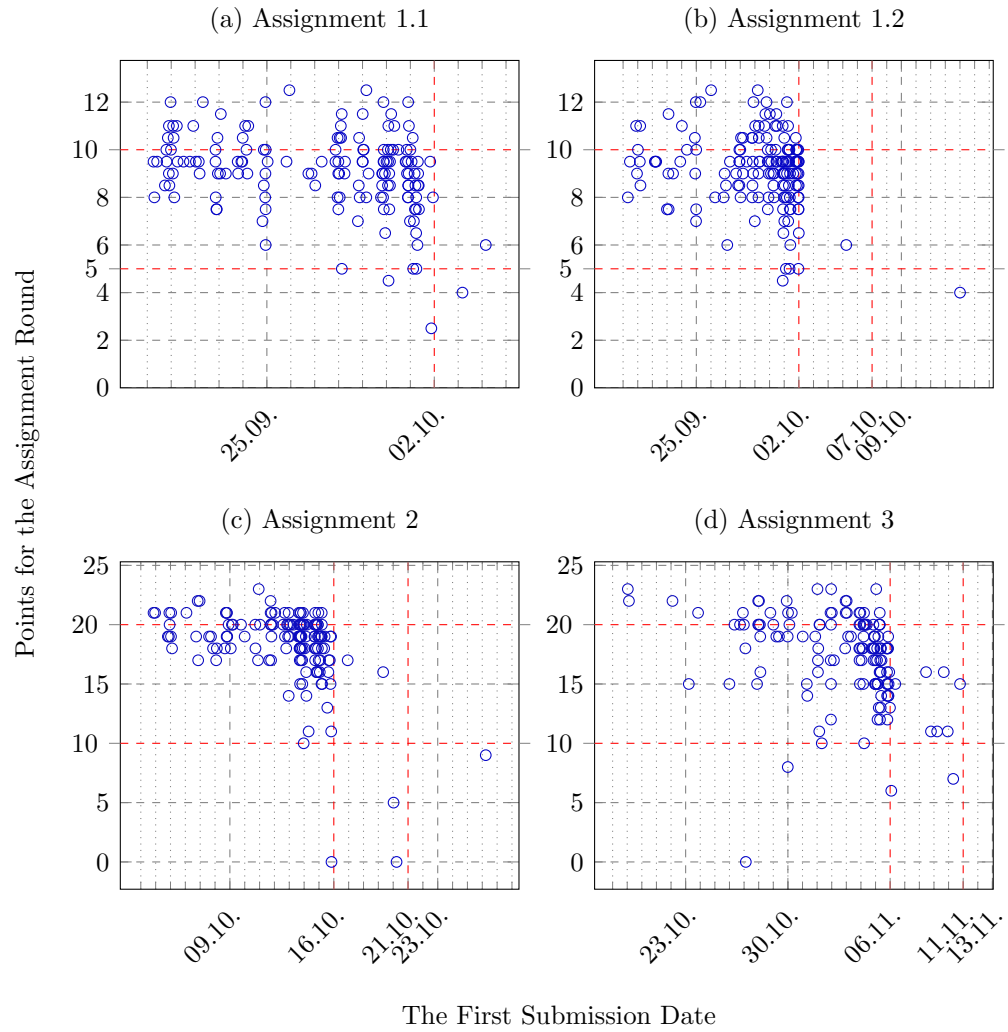
Figure 9.5: Students' first submissions compared with the points achieved for the assignment round. The late penalty is applied where necessary. The two horizontal lines indicate the minimum points to pass the assignment and the maximum points awarded without bonus functionality. The vertical lines indicate the deadline and the late submission deadline.

submission to the assignment 1.2 slightly later than for the assignment 1.1, which is most likely explained by the students first completing the assignment 1.1 before moving onto assignment 1.2. Despite the encouragement from the course staff, not many students started working on the second or third assignments earlier than after the deadline in *Programming 1* course.

9.3   Summary

We had no hypothesis for the number of trial submissions students would make each round. It was expected that more difficult assignments would receive more submissions, which they did. However, five submissions per students to the first round and twenty submissions per students to the second and the third rounds on average seems quite little. We suspect that there would have been more trial runs with a local GUI simply because the cost of an execution would have been lower. The number of failed submissions — code that did not compile — was on the other hand surprisingly high. The dependencies between the logic code and the GUI code were non-existent and nearly all compilation errors should have been avoidable.

The comparison of the number of submissions per student and the reached points revealed that generally those students who used more than the average number of submissions reached higher points. However, there were outliers; some students had a great number of submissions but reached only average points. The behaviour is more distinct in inspection of the number of submissions per students compared with the points for the whole course. However, just by looking at the data it is impossible to name a reason why some students used so much more submissions than others. Some further inspection of first submission dates and the number of submission is synthesised in Appendix C, but it revealed no clear pattern for submission numbers.

The analysed compilation times revealed clearly what had already been observed during the course. The compilation takes too long, and there are problems when the submissions of several students coincide. The system with its current speed is not a viable replacement for local GUIs.

Analysis of submission dates revealed that many students started working on the assignments only three days before deadline. Although this kind of behaviour was expected, it was surprising that so few students altered their schedule despite all encouragements to start earlier and the fact that the compilation took long and thus slowed down the development. The analysis revealed that the students who started working on the assignment closer to the deadline received on average slightly less points.

Part IV

Discussion and Conclusions

Chapter 10

Discussion

This chapter reviews the results as a whole and further analyses the findings compared with the literature. The theoretical implications are discussed in Section 10.1, while practical implications are covered by Section 10.2 with the added perspective from the instructor covered by Section 10.3. Section 10.4 discusses the limitations of the evaluation process.

10.1   Reflection on Research

In the autumn of 2015, the retention rate in our course was relatively good, with 137 students out of 170 ($\approx 81\%$) completing the course. We cannot attribute the retention rate to the successful design of the exercises although the assignments discussed in this thesis have received good feedback during the course. When students were asked to evaluate the interestingness of the assignments on a scale from one (poor) to six (awesome), the rock-paper-scissors assignment got 4,12 (n=151); the tic-tac-toe assignment 4,48 (n=148); the sound filtering assignment 4,47 (n=148); and the image filtering assignment 4,21 (n=145).

The course design and the effort we put in attending to our students, so that they begin working on the assignments on time and turn them in before the deadline, also has significant effect on the retention rate. In order to pass the course, students are expected to (1) successfully complete all assignments,

some of which could be replaced by an exam this year, (2) attend the problem based learning sessions for at least the second and the third assignment round, and (3) return two essays; one at the beginning of the course and one at the end of the course. Most of the students who did not complete the course successfully, either did not start the course at all or dropped out during or after the second assignment round. Some students who failed the course could not meet the requirement of attending the mandatory PBL sessions. When a student is committed to taking the course, she usually finishes it successfully.

Our choice for the use of games and media part of the learning content we want to teach our students. The course is not a prerequisite for any other course, but takes a side-track familiarising students with media computation. Other important learning outcomes of the course include project management, social, and presentation skills. The course does not have a final exam, and we did not measure how well our students actually master these skills. However, the assignments are designed so that students should meet enough of the learning outcomes to pass them successfully.

In the final feedback for the course that, the students are asked to evaluate if their interest toward computer science has increased on a scale from one (completely disagree) to four (completely agree). One hundred and three students gave final feedback at the end of the course in 2015, they gave the average evaluation of 2,82; which was slightly lower compared with the previous year (3,07; n=88). While the decline in the evaluation this year is disheartening, we do not actually know what effect the course had on students and should not be too discouraged about the result as the literature suggests that curriculum will most likely have no effect on students' choice for minor [13]. Furthermore, the students evaluated that the things they learned during the course will probably be useful for them in the future with grade 3,98 on a scale from one to five and that the course was useful with grade 3,12 on a scale from one to four.

To conclude, this thesis did not contribute to the research on student motivation, because the focus was on the technical implementation and trial of the system. From the perspective of the research on programming environments and web-based programming environments in particular, the

thesis presented a unique implementation of a modern system that provides a managed execution environment for students' exercises. The produced GUIs were modern as well as supported interaction and media manipulation. The students enjoyed using the system, and were impressed that their code was doing "something real". Furthermore, the system that is tailored for Scala programming course is a novel one, because Scala is yet not a very common choice for CS1 courses.

10.2   Implications of the Results

Clearly, EDCAT's biggest disadvantages compared with a local trial GUI were with performance times and runtime error messages. Especially the speed constitutes a problem of such scale that it should be solved before the system is considered for further use. In the present trial configuration of EDCAT, the system was run on a virtual server with shared CPU and disk resources. This meant that JVM, as well as *sbt* needed to be started for the compilation of each submission and disk input and output operations were not at their optimal speed.

Regardless of server configuration improvements, the software should be modified so that it supports updates for a time estimate and position in the queue while the code is compiled. This way the page could be automatically refreshed once the compilation is completed. Additionally, limiting the number of submissions per student on the server to only one submission would help to contain the stress on the server.

The problem with runtime error messages is twofold; Viewing the error messages requires opening the browser's JavaScript console and the generated error messages do not correspond to right lines in Scala code. Opening the console is not a big inconvenience but impossible a task if one is not aware of its necessity. The easiest way to overcome this issue is to instruct students on how to access the JavaScript console. A better solution would be to have a console visible at all times alongside the trial GUI.

Because only the generated JavaScript was passed to A+ and the source map was discarded, the error messages did not correctly convey the location

of the errors. The source maps could have been included although discarding them contained the size of response files; The response files were on average one megabyte in size and the source maps equally large. If the JavaScript file would have been fully optimised and minified, the equivalent file sizes would have been in the range of two hundred kilobytes for the JavaScript file and six hundred kilobytes for the corresponding source map file. However, the limitations of the compilation server did not allow us to use the full optimisation procedure.

Both the student questionnaire as well as the interviews revealed that there were misconceptions regarding the purpose and use of EDCAT. There were students who expected the system to explicitly indicate if their solution was correct or wrong. This may very well be related to the fact that the *Programming 1* course — which nearly all students taking the *Programming Studio 1* course are completing concurrently — uses a grading system that clearly shows which tests the student's solution did not pass. All students did not think of their solution as a part of a complete program but more so that EDCAT was a system their code was passed through to test it.

Students' misconceptions could be easily avoided with comprehensive introduction and instructions to the purpose and use of the system. It could be beneficial to similarly explain broadly how EDCAT works. The general introduction accompanied by a demonstration could be given in the first lecture of the course and there could be an introductory exercise that showcases the operation of the system and does not affect the course grade. Additionally, each trial GUI could have their own instructions and explanation of the operating principle. This way similar obscurities as in the third assignment round, where students did not expect all image filters to be applied to the image, would be avoided.

Better instructions could have been useful also in regard to the submission procedure and grading practices. As some students mentioned, they were annoyed by the need to separately submit code for grading after submissions for trial runs and the fact that A+ did not show their points. Sharing information would be a great first aid to the annoyance, but those issues could also be overcome altogether. The page with the trial GUI could have

an option to submit the code for grading so that the student would not need to upload the code twice. Additionally, A+ could be modified to support half and bonus points, or the grading should be adjusted so that there would not be half points and the bonus points would be taken into consideration while setting the grade limits.

Furthermore, the system as is, did not measure up to local GUIs in terms of debugging features. As one of the teaching assistants had noted, some students used EDCAT as a sole measure for running their program code. They definitely encountered difficulties when their code contained errors as they did not have proper tools or skill to solve them efficiently. Additionally, the assignment feedback forms revealed that many students had difficulties with getting started with the assignment and many struggled also with errors. The students should be taught a systematic way to debug their code and those skills should be formally practised throughout the course.

Additionally, browser dependencies caused some minor issues in the second assignment round. Those could have been noticed and avoided by testing the system with different browsers. All in all, the students enjoyed the fact that their code was used with real media content and produced some real program. The web environment and interactivity should definitely be taken a greater advantage of in the future. Allowing the upload of students' own image and sound files for manipulation would highlight more strongly that the program code students write actually performs some real functions.

From the point of view of teaching assistants, EDCAT's potential in remote tutoring should have been supported and emphasised more. The students should be encouraged to utilise the provided forum platform and guided to share links to their submission so assistants could easily see to what students' problems are related. Similarly, the teaching assistants should have been formally assigned to have also remote tutoring duties.

Compared with the local GUIs, using EDCAT was beneficial for teaching assistants' grading tasks especially when the student's solution was correct or nearly correct. With solutions far from correct, the locally run GUI with IDE's debugger would have been more helpful. Furthermore, many teaching assistants ran students' code against a test suite which required to

first download students' code. Testing functionality could be integrated as a feature for teaching personnel; The assistants could together define tests that could be ran similarly in browser. All the assistants would have the same tests at hand and the grading procedure could be more uniform.

The responses to the teaching assistant questionnaire revealed that teaching assistants were unwilling to put effort towards using the local GUI even when it was provided. This encourages to further develop online features in order to facilitate teaching assistants' work and allow them to concentrate more on the relevant tasks of helping the students and less on the technical issues that they need to overcome to do their job.

The technical usage data in addition to justifying the blame for long compilation times revealed that students started working on the assignments close to the deadline and did not use that many submissions. There is no reference value to estimate if the number of submissions was uncharacteristically low or data to show if starting late in some cases meant that students did not have enough time to reach the correct solution. However, as the server was slow, the cost of submitting code for a trial run could be high in students' eyes.

It is possible that some students wrote the program code in its entirety before running it with the trial GUI for the first time, which would make debugging more difficult. Bringing the compilation times down and encouraging students to submit code after each functionality would probably benefit most students. The late starting date as such will not be a problem, if the student still has time to finish the assignment before the deadline. However, some students inevitably do run out of time and should be provided with better support to plan the time needed to finish the assignment. As with the debugging skills already discussed earlier, students' study skills are lacking at the beginning of their academic career. However, even if providing support for planning study time in the introductory programming course is useful, it is not the course's role. Nevertheless, there are always stubborn students who either ignore the course staff's kind suggestions or are more interested in other courses or activities.

Perhaps the most surprising revelation was that students submitted code that did not compile on its own. The students are probably not familiar

enough with their development environments as the compilation errors are highlighted by the IDE with red marks. Some compilation errors were caused by inconsistencies with the GUI code, which are fully acceptable. Examining the compilation errors further and during the course would give a chance to address the most common and grave issues could be discussed during the lectures. The errors could also shed some light on how the problem solving process progresses from an erroneous submission to a fully functioning one.

10.3   Instructor's Perspective

Apart from the already presented practical implications drawn from students' and teaching assistants' feedback, EDCAT affected the work of the course's instructor. The distribution of GUI code to students was simple and easy and did not require any more work compared with the locally ran GUI distribution. Furthermore, the distributed solution proved quite useful as in the second assignment round the assignment and the related GUI code was updated after the assignment had been open for one week.

To monitor that the GUI code does not have any errors, the system was in the beginning configured so that compilation errors were reported by email to the instructor. At first it was nice to follow that students were working on the assignment and it was possible to intervene when the same error occurred multiple times in a short period of times. Usually, this meant that the same student submitted identically erroneous program code many times in a row and could be given instant feedback through A+ by the instructor. After it became apparent that most compilation errors were related solely with sloppiness in the students' work the compilation script was modified and emails would be sent only if the compilation error resulted when student's code was compiled together with the GUI code. Moving the trial execution online meant also that the instructor could follow when students started working on the assignments and encourage those who were putting off the work.

As the instructor is not involved in the computer lab sessions but helps students only through IRC and Piazza, EDCAT greatly facilitated remotely

helping students with their problems. It was fast to try out student's code to duplicate the error and easy to compare with the model solution to confirm what caused the error. All in all, EDCAT proved to be useful although it did not support the development of students' own functionality as well as local GUIs. EDCAT was at its best in the first assignment round where the resulting programs were small and simple but its benefits in showcasing that students can produce real programs that use real media files should not be disregarded.

10.4   Limitations and Validity

This study was limited to evaluating if EDCAT is useful and can provide equal support to local GUIs in an introductory programming course that has a special focus on media programming. As already mentioned, the system is beneficial with assignments that result into interactive and visually rich programs. All assignments are not well suited to work with an online system; For example it might be necessary to limit the networking functions of the environment and file writing or creation is often wise to prevent.

Only roughly a third of the students in the course responded to the student questionnaire, which is not such good a sample. Similarly, the interviews did not capture a representative group of students. However, both the responses to the questionnaire and the interviews painted quite concise a picture of the main problems related to EDCAT. Some further information about the programming experience of the respondents and the interviewees would possibly have revealed more clearly the different needs novice programmers might have in comparison with the more experienced programmers. However, there was already some indication that the novices and the experienced have a different outlook on EDCAT.

Furthermore, the students had clear misconceptions about the purpose of EDCAT and what part of the system was actually evaluated. The students did not understand that A+ was a separate system that showed them instructions and provided submission features and what was meant with EDCAT was responsible for compilation and returned the trial GUIs. Thus, some responses

were disregarded and some responses may have been misinterpreted. However, the results were useful in evaluating and assessing the full submission process.

In addition to the used evaluation methods, observation could have been used to better understand how students used EDCAT. Observations could have revealed usability problems and additional ways to misuse the system, similar to that of using the system as a rudimentary version control system.

Chapter 11

Conclusions

This thesis set out to implement and trial a system for serving students a means to execute their program code in a browser with an interactive and modern GUI that use media content. EDCAT offered the sufficient functionality to validate the idea and prove its worth in the programming education context. The system was easy to use for students, the teaching assistant as well as the instructor and it did not require extra effort compared with the use of traditional and a fully local environment. The distributed design of the system proved useful and the chosen technologies supported the needs of the course.

EDCAT allowed to hide potentially difficult code from the students' eyes and produce modern and attractive GUIs. The students needed to download very little code and the updates to the exercise materials reached every student as soon as published. However, the debugging functionalities were not comparable to the locally executed GUIs as the error messages were not properly shown, not to mention the lack of a debugger available in an IDE.

The results indicate that the technical setup was defective in the first trial run of the system and proper server resources should be allocated for the system to handle the huge number of compilations within an appropriate time limit. Furthermore, technical details, in terms of browser dependencies and other limitations, must be thoroughly investigated in order to avoid the problems arisen in the second assignment round.

The feedback given both by students and the teaching body has already proven useful and must be utilised when the system should be further developed. The support for comprehensive error output should be added by the use of a simple console. This requires modifying the GUI code as well as taking use of the source maps produced by Scala.js compilation. Additionally, the feedback revealed that a more straightforward submission procedure should be implemented in the future to prevent potential confusions caused by two step submission used in the trial. Furthermore, guidance on using the environment should be given so that students could utilise its full potential.

There is still much work in developing the functionality to aid the work of teaching assistants who could grade the submissions and tutor students with the better support from the system. Implementing test suites would require little effort and equip the teaching assistant with a uniform assessment basis. The feedback functionality should be modified towards a bidirectional implementation and used for providing help also during the completion of the assignments. Additionally, functionality to gather data on student's process should be developed to allow following students' problem solving process.

Based on the results we achieved with the trial run of EDCAT, we can recommend the use of a similar system for GUI implementation and distribution. EDCAT has potential showcasing program code written by students used in an appealing GUI built with modern technology. The system is easy for the teacher to manage and composing good GUIs with the use of Scala.js takes significantly less effort than using Swing.

11.1 Future Work

In this first version only the bare minimum functionality for supporting the teaching assistants' work was implemented. As already discussed, developing tools for remote tutoring and especially aiding the grading process are areas worth looking into. The remote tutoring is somewhat supported but requires an additional environment for discussion between course staff and students. There is no sense in fully integrating a discussion platform with EDCAT but adding links to starting a new question regarding the submission could lower

the barrier to using the remote help. Better support for online tutoring is crucial if EDCAT should ever be used in a course with a big student body, similar to *Programming 1* course.

Grading students' work fairly and efficiently required downloading students' code in order to run it locally against tests. The testing system could be integrated into EDCAT so that assistants could devise a test suite together online and use it with every submission. This would also allow the partial — or if desired the full — automated grading of students' submissions and teaching assistants could concentrate on grading style and clarity, and writing feedback. Some parts of the same system could also be open for students, so that they could easily define their own tests.

An important skill for any programmer is debugging — a skill that novices need and are lacking in the most. Supporting the development of debugging skills could be supported by the trial GUIs. Should an error occur, students would be given not only the error message but also some aid as to what might cause the error on the particular line and how one should proceed in order to correct it.

Forcing the trial runs online allows for an unprecedented vantage point for monitoring students' efforts solving assignments they are given. The unfinished submissions might provide some interesting perspective into how students go about solving the assignments and the problems that might occur during the process. EDCAT might provide the means to study students' problem solving unobtrusively.

From the theoretical point of view, it would be interesting to see how well our course materials succeed in teaching the students about media programming. The exercises are in need of formal evaluation and possible redesign. Similarly, it would definitely be interesting to conduct a longitudinal study with our students, to evaluate how they choose their minor and what affects their decision making. From the technical point of view, the potential of using Scala.js in programming education should be further explored in the future. Additionally, the exercise design should be further developed to allow students to extend their solution with more freedom.

Bibliography

[1] ASTRACHAN, O., AND REED, D. AAA and CS 1: The Applied Apprenticeship Approach to CS 1. *SIGCSE Bulletin 27*, 1 (Mar. 1995), 1–5.

[2] BAYLISS, J. D., AND STROUT, S. Games as a "Flavor" of CS1. *SIGCSE Bulletin 38*, 1 (Mar. 2006), 500–504.

[3] BECKER, K. Teaching with Games: The Minesweeper and Asteroids Experience. *J. Comput. Sci. Coll. 17*, 2 (Dec. 2001), 23–33.

[4] BEN-ARI, M. Constructivism in Computer Science Education. *SIGCSE Bulletin 30*, 1 (1998), 257–261.

[5] BEN-ARI, M. Constructivism in Computer Science Education. *Journal of Computers in Mathematics and Science Teaching 20*, 1 (2001), 45–73.

[6] BRUCE, K. B., DANYLUK, A., AND MURTAGH, T. A Library to Support a Graphics-Based Object-First Approach to CS 1. *SIGCSE Bulletin 33*, 1 (Feb. 2001), 6–10.

[7] BUCK, D., AND STUCKI, D. J. Design Early Considered Harmful: Graduated Exposure to Complexity and Structure Based on Levels of Cognitive Development. *SIGCSE Bulletin 32*, 1 (Mar. 2000), 75–79.

[8] DU BOULAY, B. Some Difficulties of Learning to Program. *Journal of Educational Computing Research 2*, 1 (1986), 57–73.

[9] Edwards, S. H., Tilden, D. S., and Allevato, A. Pythy: Improving the Introductory Python Programming Experience. In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education* (New York, NY, USA, 2014), SIGCSE '14, ACM, pp. 641–646.

[10] Grissom, S. A Pedagogical Framework for Introducing Java I/O in CS1. *SIGCSE Bulletin 32*, 4 (Dec. 2000), 57–59.

[11] Guo, P. J. Online Python Tutor: Embeddable Web-Based Program Visualization for CS Education. In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education* (New York, NY, USA, 2013), SIGCSE '13, ACM, pp. 579–584.

[12] Guzdial, M. A Media Computation Course for Non-Majors. *SIGCSE Bulletin 35*, 3 (June 2003), 104–108.

[13] Guzdial, M. Exploring Hypotheses about Media Computation. In *Proceedings of the Ninth Annual International ACM Conference on International Computing Education Research* (New York, NY, USA, 2013), ICER '13, ACM, pp. 19–26.

[14] Guzdial, M., and Soloway, E. Teaching the Nintendo Generation to Program. *Communication of the ACM 45*, 4 (Apr. 2002), 17–21.

[15] Hamid, N. A. Automated Web-Based User Interfaces for Novice Programmers. In *Proceedings of the 50th Annual Southeast Regional Conference* (New York, NY, USA, 2012), ACM-SE '12, ACM, pp. 42–47.

[16] Hansen, S. The Game of Set®: An Ideal Example for Introducing Polymorphism and Design Patterns. *SIGCSE Bulletin 36*, 1 (Mar. 2004), 110–114.

[17] Hiisilä, A. Kurssinhallintajärjestelmä ohjelmoinnin perusopetuksen avuksi (Course Management System for basic courses in programming). Master's thesis, Helsinki University of Technology, Espoo, Finland, 2005.

[18] HITZ, M., AND KÖGELER, S. Teaching C++ on the WWW. *SIGCSE Bulletin 29*, 3 (June 1997), 11–13.

[19] JAZAYERI, M. Some Trends in Web Application Development. In *Future of Software Engineering, 2007. FOSE'07* (2007), IEEE, pp. 199–213.

[20] JIMÉNEZ-PERIS, R., KHURI, S., AND PATIÑO MARTÍNEZ, M. Adding Breadth to CS1 and CS2 Courses Through Visual and Interactive Programming Projects. *SIGCSE Bulletin 31*, 1 (Mar. 1999), 252–256.

[21] LAMBERT, K., AND OSBORNE, M. Easy, Realistic GUIs with Java in CS1. *Journal of Computing Sciences in Colleges 16*, 2 (Oct. 2000), 209–215.

[22] LEWIS, M. C., AND MASSINGILL, B. Graphical Game Development in CS2: A Flexible Infrastructure for a Semester Long Project. *SIGCSE Bulletin 38*, 1 (Mar. 2006), 505–509.

[23] NG, S. C., CHOY, S. O., KWAN, R., AND CHAN, S. F. *A Web-Based Environment to Improve Teaching and Learning of Computer Programming in Distance Education.* Springer Berlin Heidelberg, Berlin, Heidelberg, 2005, pp. 279–290.

[24] PERRY, R. Multimedia Web-based Programming Development Environment. In *2007 37th Annual Frontiers In Education Conference-Global Engineering: Knowledge Without Borders, Opportunities Without Passports* (2007), IEEE, pp. F2H–13.

[25] REGES, S. Conservatively Radical Java in CS1. *SIGCSE Bulletin 32*, 1 (Mar. 2000), 85–89.

[26] ROBERTS, E., AND PICARD, A. Designing a Java Graphics Library for CS1. *SIGCSE Bulletin 30*, 3 (Aug. 1998), 213–218.

[27] ROBERTS, E. S. A C-Based Graphics Library for CS1. *SIGCSE Bulletin 27*, 1 (Mar. 1995), 163–167.

[28] SHULTZ, G. Integrating 3D Graphics into Early CS Courses. *Journal of Computing Sciences in Colleges 21*, 3 (Feb. 2006), 169–178.

[29] SIMON, B., KINNUNEN, P., PORTER, L., AND ZAZKIS, D. Experience Report: CS1 for Majors with Media Computation. In *Proceedings of the Fifteenth Annual Conference on Innovation and Technology in Computer Science Education* (New York, NY, USA, 2010), ITiCSE '10, ACM, pp. 214–218.

[30] SMITH III, J. P., DISESSA, A. A., AND ROSCHELLE, J. Misconceptions Reconceived: A Constructivist Analysis of Knowledge in Transition. *The Journal of the Learning Sciences 3*, 2 (1993), 115–163.

[31] STEPHENSON, B., AND TAUBE-SCHOCK, C. QuickDraw: Bringing Graphics into First Year. *SIGCSE Bulletin 41*, 1 (Mar. 2009), 211–215.

[32] SUNG, K., PANITZ, M., WALLACE, S., ANDERSON, R., AND NORDLINGER, J. Game-Themed Programming Assignments: The Faculty Perspective. *SIGCSE Bulletin 40*, 1 (Mar. 2008), 300–304.

[33] SUNG, K., SHIRLEY, P., AND ROSENBERG, B. R. Experiencing Aspects of Games Programming in an Introductory Computer Graphics Class. *SIGCSE Bulletin 39*, 1 (Mar. 2007), 249–253.

[34] TRUONG, N., BANCROFT, P., AND ROE, P. A Web Based Environment for Learning to Program. In *Proceedings of the 26th Australasian computer science conference-Volume 16* (2003), Australian Computer Society, Inc., pp. 255–264.

[35] VAZHENIN, D., VAZHENIN, A., AND WANG, Y.-H. WEB-based Environment for Programming and Distance Learning. In *19th International Conference on Advanced Information Networking and Applications (AINA'05) Volume 1 (AINA papers)* (2005), vol. 2, IEEE, pp. 109–112.

[36] VON GLASERSFELD, E. A Constructivist Approach to Teaching. In *Constructivism in Education*. Erlbaum, Hillsdale, 1995, pp. 3–15.

[37] YOO, D., SCHANZER, E., KRISHNAMURTHI, S., AND FISLER, K. WeScheme: The Browser is Your Programming Environment. In *Proceedings of the 16th Annual Joint Conference on Innovation and Technology in Computer Science Education* (New York, NY, USA, 2011), ITiCSE '11, ACM, pp. 163–167.

Appendix A

Student Questionnaire

The questionnaire for accessing the students' opinions regarding EDCAT is shown in Figure A.1.

Figure A.1: The student questionnaire (continued on next page).

Palaute kurssin ME-C2110 Ohjelmointistudio 1: mediaohjelmointi palautusjärjestelmästä kierroksilla 1 - 3

Kurssin kolmella ensimmäisellä kierroksella käytetty palautusjärjestelmä on rakennettu niin, että A+ vastaanottaa palautuksen, jonka se ohjaa erilliselle kurssin omalle arvostelupalvelimelle. Arvostelupalvelin huolehtii ohjelman kääntämisestä JavaScript-koodiksi ja palauttaa opiskelijalle testailuun soveltuvan käyttöliittymän.

A+ on siis erillinen järjestelmä, joka huolehtii opiskelijan palautusten säilömisestä sekä arvostelupalvelimen palauttaman käyttöliittymän näyttämisestä. Tässä kyselyssä kiinnostaa erityisesti arvostelupalvelimeen liittyvät asiat.

Auta minua kehittämään järjestelmää vastaamalla alla olevaan kyselyyn.

Kyselyn tuloksia käytetään ennen kaikkea diplomityöni tutkimusaineistona sekä myöhemmin järjestelmän jatkokehitykseen.

Kysely on anonyymi, eikä vastauksia ole mahdollista yhdistää vastaajiin.

Vastaamisen jälkeen sinut ohjataan uuteen kyselyyn, jossa voit jättää yhteystietosi, jos olet kiinnostunut osallistumaan haastatteluun liittyen järjestelmän kehittämiseen.

1. Hyödyllisyys *

	Täysin eri mieltä	Eri mieltä	Ei samaa eikä eri mieltä	Samaa mieltä	Täysin samaa mieltä
Järjestelmän käyttö auttoi minua ratkaisemaan annettuja tehtäviä	○	○	○	○	○
Järjestelmän käyttö auttoi minua löytämään ohjelmassani piileviä virheitä	○	○	○	○	○
Järjestelmän käyttö auttoi minua ymmärtämään kirjoittamani koodin toimintaa	○	○	○	○	○
Koin järjestelmän hyödylliseksi	○	○	○	○	○

2. Vapaat kommentit liittyen edelliseen kohtaan.

3. Helppokäyttöisyys *

	Täysin eri mieltä	Eri mieltä	Ei samaa eikä eri mieltä	Samaa mieltä	Täysin samaa mieltä
Minun oli helppo oppia käyttämään järjestelmää	○	○	○	○	○
Vuorovaikutus järjestelmän kanssa oli selkeää ja ymmärrettävää	○	○	○	○	○
Olisin tarvinnut enemmän tukea kurssin henkilökunnalta järjestelmän käyttöön	○	○	○	○	○
Mielestäni järjestelmää oli kokonaisuudessaan helppo käyttää	○	○	○	○	○

4. Vapaat kommentit liittyen edelliseen kohtaan.

5. Käyttökokemus *

	Täysin eri mieltä	Eri mieltä	Ei samaa eikä eri mieltä	Samaa mieltä	Täysin samaa mieltä
Sain järjestelmästä riittävän nopeasti palautteen	○	○	○	○	○
Luotin siihen, että pystyn kokeilemaan koodiani järjestelmän palauttaman käyttöliittymän avulla	○	○	○	○	○
Sain riittävästi tietoa ajon aikana tapahtuvista virheistä	○	○	○	○	○
Pidin järjestelmän käytöstä	○	○	○	○	○

6. Vapaat kommentit liittyen edelliseen kohtaan.

7. Jos sinulla oli joitakin selviä ongelmia järjestelmän käytössä, mihin ne liittyivät?

8. Mitä järjestelmässä pitäisi mielestäsi kehittää?

9. Mitkä toiminnot ja ominaisuudet olivat kaikkein hyödyllisimpiä?

Lähetä

Appendix B

Teaching Assistant Questionnaire

The questionnaire for accessing the teaching assistants' opinions regarding EDCAT is shown in Figure B.1.

Figure B.1: The teaching assistant questionnaire.

ME-C2110: Palautuskäytännöt kierroksilla 1–3

1. Nimi *

Etunimi [   ]

2. Olen aikaisemmin ollut assistenttina kurssilla, jolla opiskelijoiden ohjelmointitehtäviä arvostellaan käsin. *

☐ Kyllä
☐ Ei

3. Jos vastasit edelliseen kohtaan kyllä, koetko että online-testikäyttöliittymä auttoi sinua assistentin työssä verrattuna kursseihin, joilla järjestelmää ei ole käytössä. Jos auttoi, niin miten? Vai aiheutuiko siitä kenties enemmän työtä?

[   ]

4. Merkitse seuraavissa kohdissa, millä kierroksilla väitteet pitävät paikkaansa.

	Kierroksella 1	Kierroksella 2	Kierroksella 3
Hyödynsin online-testikäyttöliittymää kun assaroin opiskelijoita ohjelmointiharjoitustilaisuuksissa.	☐	☐	☐
Ajoimme opiskelijan ohjelmakoodia opiskelijan kanssa yhdessä.	☐	☐	☐
Ajoin opiskelijan ohjelmakoodia yksin.	☐	☐	☐
Hyödynsin online-testikäyttöliittymää kun assaroin opiskelijoita Ircin tai Piazzan välityksellä.	☐	☐	☐
Koin, että online-testikäyttöliittymästä oli hyötyä auttaessa opiskelijoita ratkaisemaan ohjelmointiharjoituksia.	☐	☐	☐

5. Voit tässä jättää kommentteja edellisen kohdan kysymyksiin liittyen tai muita ajatuksia online-testikäyttöliittymän hyödyistä ja haitoista assaroitaessa ohjelmointiharjoitustilaisuuksissa ja etänä.

[   ]

6. Merkitse seuraavissa kohdissa, millä kierroksilla väitteet pitävät paikkaansa.

	Kierroksella 1	Kierroksella 2	Kierroksella 3
Latasin opiskelijoiden ohjelmakoodin omalle koneelleni arviointia varten.	☐	☐	☐
Ajoin opiskelijoiden ohjelmakoodia lokaalisti omalla koneellani graafisen käyttöliittymän avulla.	☐	☐	☐
Ajoin opiskelijan ohjelmakoodia lokaalista omalla koneellani testiluokkia vastaan.	☐	☐	☐
Palautin opiskelijan muokkaamatonta ohjelmakoodia A+:ssa olevaan testikäyttöliittymään ja kokeilin koodia siellä.	☐	☐	☐
Palautin opiskelijan muokattua (korjasit esimerkiksi virheitä) ohjelmakoodia A+:ssa olevaan testikäyttöliittymään ja kokeilin koodia siellä.	☐	☐	☐
Koin, että online-testikäyttöliittymästä oli hyötyä opiskelijoiden ohjelmakoodin arvioinnissa.	☐	☐	☐

7. Voit tässä jättää kommentteja edellisen kohdan kysymyksiin liittyen tai muita ajatuksia online-testikäyttöliittymän hyödyistä ja haitoista opiskelijan ohjelmakoodin arvioinnissa.

[   ]

8. Vapaa sanaa kierrosten 1–3 online-testikäyttöliittymiin ja niiden käyttöön ja toimintaan liittyen.

[   ]

Appendix C

Further Analysis of Submission Numbers

Comparison of students' first submission date with the number of submissions the student or student pair used. There is no clear pattern to indicate a reason for why some students used significant amount of submissions or why some did not.
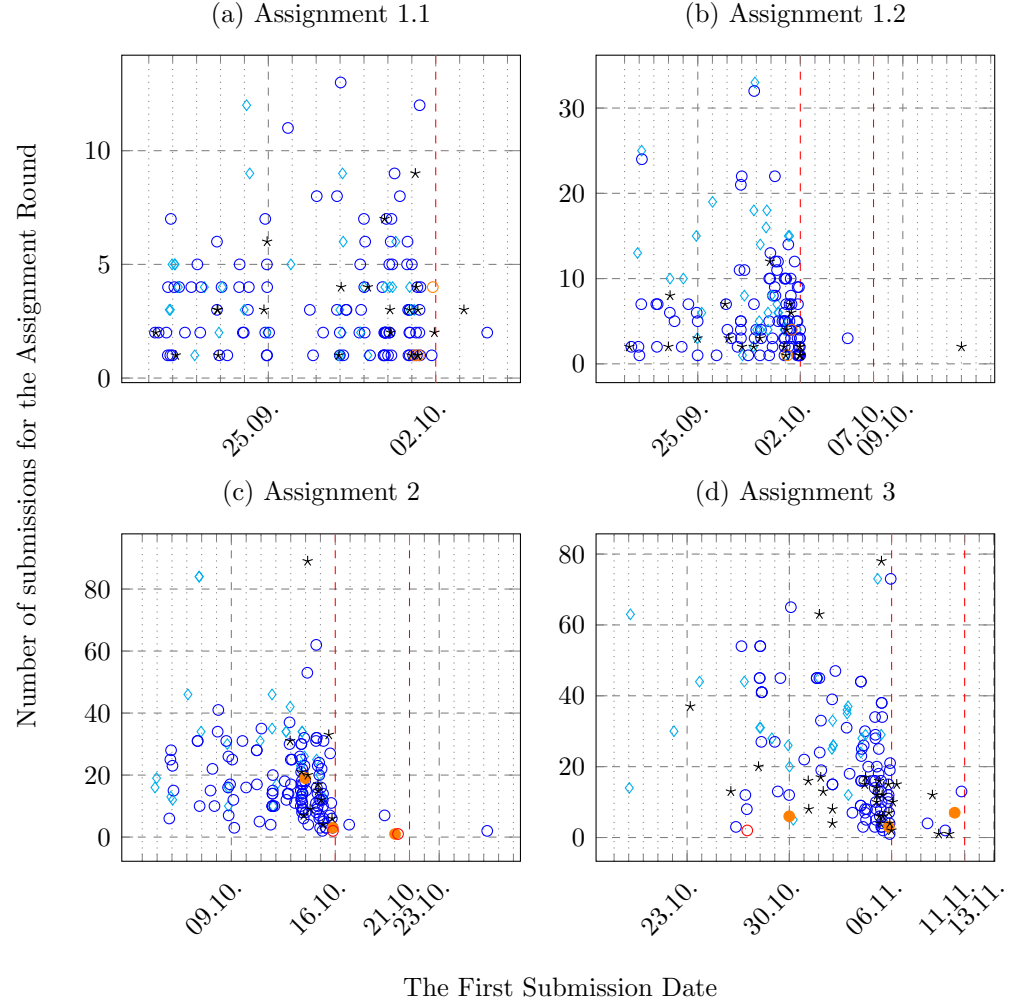
Figure C.1: Students' first submissions compared with the number of submissions for the assignment round. The vertical lines indicate the deadline and the late submission deadline. In the first assignment round the markers respond to legend's markers divided by two.