



NTNU – Trondheim
Norwegian University of
Science and Technology

Integrating a Web-Based Editor in the Cloud with TDT4100s Course Wiki

Lasse Brudeskar Vikås

Master of Science in Computer Science

Submission date: June 2015

Supervisor: Hallvard Trætteberg, IDI

Norwegian University of Science and Technology
Department of Computer and Information Science

Summary

Learning how to develop software is by itself not an easy task; Even more so if there is a steep learning curve at the initial stages just to configure the development environment. For the students at NTNU in the course TDT4100, the exercises they need to work on requires many steps to set up the development environment before they can begin working on the code.

In this research, a previously created web editor was used and developed further to fit the new requirements. The web editor itself, due to the requirements, had to support many features that would have existed in the students' local development environment, such as working on multiple files, syntax highlighting, error/warning checking, content assists, running code and performing tests.

As well as having these features, the web editor had to be available for the students for each exercise they have to work on. This meant that the web editor must support being initiated from the location of any exercise, with the content that was required for that exercise. The instructor of the course must be able to decide where the web editor should be, and what content is required for the exercise.

The result from this research is two artifacts; Confluedit and Eclipsky. Confluedit has the responsibility of initiating the web editor by giving it the exercise description, while Eclipsky handles the projects and serves the web editor.

To verify the quality of Eclipsky, performance measuring and usability testing was performed. The performance of the implementation was within the requirements, with the highest average delay just above half a second from starting a new exercise. When measuring the basic use of the web editor, operations like updating the resources, content assists and error/warning checks had a delay below *25ms*. Although there were some usability issues that needs to be looked into, the overall quality was satisfactory.

Sammendrag

Å lære hvordan man utvikler programvare i seg selv ikke en enkel operasjon; Spesielt ikke hvis det er en bratt læringskurve i starten bare for å konfigurere utviklingsmiljøet. For studenter på NTNU i emnet TDT4100 krever oppgavene som de må jobbe med å sette opp ett lokalt utviklingsmiljø, noe som tar mange steg før de kan begynne å kode.

For denne forskningen ble det brukt en allerede eksisterende web editor som ble utviklet videre for å dekke de nye kravene. Web editoren i seg selv måtte, på grunn av kravene, støtte mange funksjoner som studentene ville hatt i sitt lokale utviklingsmiljø. Disse funksjonene er blant annet å kunne jobbe med mange filer, syntaksmerking, feil/advarsel varsling, innholdshjelp, kjøring av kode og utføre tester.

I tillegg til disse funksjone, måtte web editoren være tilgjengelig for studentene for alle oppgavene de må jobbe med. Dette betyr at web editoren måtte støtte å bli opprettet fra lokasjonen til enhver oppgave med det innholdet som kreves for den gitte oppgave. Instruktøren i emnet må kunne bestemme hvilken lokasjon web editoren skal kunne opprettes i, og hvilket innhold som kreves for øvingene.

Resultatet fra denne forskningen er to artifakter; Confluedit og Eclipsky. Confluedit har ansvaret for å åpne opp web editoren ved å gi den en beskrivelse av oppgaven, mens Eclipsky tar seg av prosjektene og dele ut web editoren.

For å verifisere kvaliteten av Eclipsky ble det gjort forskjellige evalueringer. Dette besto av å måle ytelse og utføre en brukertest. Ytelsen på systemet var innenfor kravene med en høyest gjennomsnittlig verdi på like over ett halvt sekund på å opprette ett nytt prosjekt. Under måling av selve bruken av editoren, funksjoner som å oppdatere ressursene, innholdshjelp og feil/advarsel varsling hadde en gjennomsnittlig ventetid på under $25ms$. Til tross for at det ble oppdaget noen problemer med brukervennligheten som må jobbes videre med var den totale kvaliteten til systemet tilfredsstillende.

Preface

This project was defined by Hallvard Trætteberg at the Department of Computer and Information Science at the Norwegian University of Science and Technology. I would like to thank Hallvard Trætteberg for the support throughout the project.

I would also like to thank all the anonymous participants that was part of the usability testing, which helped make this research possible.

Also a special thanks to Ole Kristian Nakken and Petter Troseth Almås for the help in structuring the report and giving valuable suggestions.

Contents

1	Introduction	1
1.1	Goals	2
1.1.1	Goal 1 Student Availability	2
1.1.2	Goal 2 Student Usability	2
1.1.3	Goal 3 Instructor Usability	3
1.1.4	Goal 4 Extensibility	3
1.2	Structure	3
2	Background and Theory	5
2.1	Quality Attributes	5
2.1.1	Product quality	5
2.1.2	Quality in use	6
2.2	The Course	6
2.2.1	Assignments	6
2.2.2	Exercises	7
2.2.3	Tests	7
2.2.4	Avoiding environment configuration	8
2.3	Eclipse	9
2.3.1	Eclipse Java editor features	9
2.4	Assignment System prototype	12
2.4.1	Gamification	13
2.5	Scalability	13
2.6	Confluence	14
2.6.1	Page editor	14
2.6.2	EMFS	15
3	Method	17
3.1	Development method	17
3.2	Literature	18
3.3	Data gathering	18

3.4	Usability testing	18
3.4.1	Types of testing	19
3.4.2	Performing usability test	19
3.4.3	Post-testing survey	19
4	Results	21
4.1	Overall design	21
4.2	Requirements and constraints	22
4.2.1	Editor Functionalities	22
4.2.2	Editor Performance	23
4.2.3	Implementation Constraints	23
4.3	Confluedit – Instructor’s viewpoint	24
4.3.1	Creating the macro	24
4.4	Confluedit – Students’ viewpoint	24
4.4.1	Initializing project on Confluedit	25
4.5	Eclipsky	26
4.5.1	Serving web page	27
4.5.2	Compiling and running code	30
4.5.3	Towards cloud service	30
4.5.4	OSGi-runtime	32
4.6	Properties of a project	36
4.6.1	taskId	36
4.6.2	difficulty	36
4.6.3	effort	36
4.6.4	xemfs	36
4.7	Communication components	37
4.7.1	Refresh/Update component	37
4.7.2	Close component	37
4.7.3	Completion component	37
4.7.4	Run component	37
4.7.5	Test component	38
5	Evaluation	39
5.1	Confluedit	39
5.2	Performance of Eclipsky	39
5.2.1	Test platform	40
5.2.2	Results	40
5.2.3	Starting project	41
5.2.4	Editor tasks	41
5.2.5	Run/Test performance	42
5.3	Scalability	42
5.4	Usability testing	42

5.4.1	Task performance	43
5.4.2	All task scores	43
5.4.3	Combined results	47
5.5	Issues	47
5.5.1	Running code	47
5.5.2	Content assist	48
5.5.3	Individual run time	48
5.5.4	Test result	48
5.5.5	Altering the UI	49
5.5.6	Conclusion	49
6	Discussion	51
6.1	Usability testing	51
6.1.1	Content assist	52
6.1.2	Run time	52
6.1.3	Testing	52
6.1.4	UI	52
6.2	Cloud	53
6.3	Tested exercises	53
7	Conclusion	55
7.1	Future research	55
7.1.1	Including EMFS in Confluence	55
7.1.2	Learning Analytics	56
	Appendices	61
A	Workflow	I
B	Test plan	III
B.1	Purpose	III
B.2	Research Questions	III
B.3	Participant Characteristics	IV
B.4	Method	IV
B.5	Session outline	IV
B.6	Test Environment	V
B.7	Moderator Role	V
B.8	Tasks	V
B.9	Post-test questions	VII

C	Test result	IX
C.1	Task notes	IX
C.2	Post-test question	XIII
C.2.1	Was there any problems/difficulties when using the product that prevented you from solving the tasks?	XIII
C.2.2	Did you find the product easy to work with? If so, why?	XIII
C.2.3	Was it easy to understand and use the test panel for solving the tasks?	XIV
C.2.4	What did you think about how the resources was managed in the system?	XV
C.2.5	Is there anything else about the system that you would want to have been handled differently?	XV
C.2.6	If you imagine yourself as a new student in this course, would you consider using this product instead of the current implementation?	XV
D	Performance measurements	XVII
E	Source Code	XXI

List of Figures

2.1	An example of the meta data associated with an exercise.	7
2.2	An exercise from the course Wiki, with superimposed, colorized divisions for the different parts as well as a mock-up of the editor .	8
2.3	An example of the use of content assist with <code>System.out</code>	10
2.4	An example of the use of templates. Left side: Initiated code suggestions. Right side: “main” template chosen.	10
2.5	An error marker will show additional info when the programmer hovers over the marker icon.	11
2.6	Warning marker that are being rendered because the programmer have an unused variable in the code	12
2.7	The page editor, with both standard HTML-elements and a macro for including code blocks.	14
2.8	A simple illustration of how different modules can coexist inside a plugin	15
4.1	Different parameters that must be entered into the plugin	25
4.2	The content of the macro with XEMFS-language. The file structure is described, where the content can either be in-line or downloaded.	26
4.3	The web editor with the different functionalities labeled with numbers	28
4.4	The web editor with all tests passed	29
4.5	All bundles (except <code>no.hal.eclipsky.services.workspace.http</code>) with their components	34
4.6	The <code>no.hal.eclipsky.service.workspace.http</code> bundle with all its components	35
A.1	A overview of the entire implementation	II

List of Tables

- 5.1 Hardware for the testing platform 40
- 5.2 Results from measuring different functions in the web editor 41
- 5.3 P1’s score for the three tasks 44
- 5.4 P2’s score for the three tasks 45
- 5.5 P3’s score for the three tasks 45
- 5.6 P4’s score for the three tasks 46
- 5.7 P5’s score for the three tasks 46
- 5.8 A summary of all participant’s result for each task 47

- B.1 Task 1 VI
- B.2 Task 2 VI
- B.3 Task 3 VI

- C.1 Results from all participants in Task B.1 X
- C.2 Results from all participants in Task B.2 XI
- C.3 Results from all participants in Task B.3 XII

- D.1 Performance of ensuring a project, opening up a ensured project
and changing resources XVIII
- D.2 Performance of running a project with and without run configura-
tion, and testing XIX
- D.3 Performance of doing basic tasks in editor XX

Abbreviations

AJAX	Asynchronous JavaScript and XML. 25, 26
AST	Abstract Syntax Tree. 16
DSL	Domain-Specific Language. 16, 36
EMF	Eclipse Modeling Framework. 16
EMFS	EMF File System. V, VII, 15, 16, 23, 36, 53, 55, 56
HTML	Hyper-Text Markup Language. IX, 14, 21, 26
IDE	Integrated Development Environment. 1–3, 9, 10, 12, 22, 23, 41, 51, 52, 56
JDT	Java Development Tools. 23, 32, 37
MSc	Master of Science. IV
NTNU	Norwegian University of Science and Technology. I, II, IV, V, 1
OSGi	Open Service Gateway initiative. VI, 9, 27, 30–33, 53
PDE	Plug-in Development Environment. 9, 32
RCP	Rich Client Platform. 9
UCD	User-centered design. 18
UI	User Interface. VII, 49, 52
XEMFS	Xtext based EMFS. IX, 16, 23, 24, 26, 39, 51, 56

Chapter 1

Introduction

Over the last years, code editors on the web has become topic of discussion for replacing the traditional desktop code editors and IDE's. These web editors tend to either be very simplistic, e.g. only having syntax highlighting, or to approach a full-fledged IDE experience. However, none are approaching a solution that is a cross between the two counterparts, which can be utilized when teaching students how to develop software.

Learning how to develop software is by itself not an easy task. This is even harder for a student if there is a steep learning curve at the initial stages just to configure the environment. If a student manages to configure a local development environment with a full-fledged IDE, the functionality a student needs will be met, but the IDE has a plethora of other functions that might confuse a student. There exists different cloud based editors already, but they tend to be either to simple or to advanced. For the course TDT4100 - Object-oriented programming at the Norwegian University of Science and Technology (NTNU), the students have to deal with this problem when learning how to write code for the Java platform.

The goal is a web-based code editor for ad-hoc usage for students to work on exercises. The solution should be both functional, with essential features like error markers and code completion, and have the ability to run and test the code. It should also be simple to use.

In order for students to be able to take the exam, they must pass 10 assignments. Each assignments consists of about 5 exercises that the students must work on, which is found on the course wiki. Creating the exercises takes a lot of work, and including the exercises on the course wiki is also time consuming. The instructors need to be able to quickly configure new exercises on the course web site, including dependencies on tests, existing files and libraries, and have both the editor and back-end behave appropriately.

This thesis will cover the architecture, implementation and evaluation of such a system.

1.1 Goals

The main problem this thesis tries to solve is making it easier for students to write code when learning how to develop software. To get a better control of the different aspects of this, the problem description have been divided in to goals that describe what the final implementation will try to achieve.

The goals are as follows:

Goal 1 Student Availability

Students must be able to start working on an assignment from the course wiki

Goal 2 Student Usability

Design the web editor to the responsiveness and usefulness of a local application

Goal 3 Instructor Usability

Implementing a solution that makes it easy for instructors to add new assignments

Goal 4 Extensibility

The system should be implemented in such a way that it supports future expansion/changes

1.1.1 Goal 1 Student Availability

As the users have access to the exercises through the course wiki, the cloud based service must be accessible through here. The web editor should therefore be rendered on the page on request from users, and the exercises must be configured/prepared on the server running a form of Eclipse.

1.1.2 Goal 2 Student Usability

Although syntax-highlighting is supported in most JavaScript editors (Wikipedia, 2015), having more of the functionality that a full-blown IDE offers for users would make the web based editor for this implementation much more useful. The problem is finding out how much features too add, as adding too much could lead to confusion/performance troubles, and adding too little would force the users to go back to configuring/installing a local IDE instead.

The intended context for this product is for students who have gone through the basics of Java, but have done so in a simple text editor. Although the students do not have any experience with an IDE at this point, the functionality of the web editor should reflect those that exist in an IDE for both transferring the students' obtained skills and easing the transition over to a full-fledged IDE later. In order to find out how well this implementation fits the students, user tests will be performed as part of a qualitative analysis.

The most prominent functionalities that an IDE has that a simple text-editor is lacking is context sensitive content assists (hereafter referred to as content assists), error/warning markers, running code and get the output. These functionalities will be explained in section 2.3.1.

1.1.3 Goal 3 Instructor Usability

Since the final implementation will be used on the course wiki, it has to support the different exercises the instructors have created for the students to work on. As these exercises require different technical attributes, the web based editor must be able to handle all the potential exercise descriptions (preferably at run-time).

When instructors create an exercise, they use a lot of time to both write codes and create tests. The usability aspect this goal must fulfill here is that it should be an efficient task for an instructor to incorporate the exercises in the course wiki.

1.1.4 Goal 4 Extensibility

Future exercises could use completely different problem descriptions that the course wiki currently offers. If the implementation is unable to handle a specific type of problem, the work required to fix the problem should be minimal. Separation of concerns and abstractions is therefore a key part of the system, which should satisfy extensibility in the definition of Johansson and Löfgren (2009).

1.2 Structure

This section describes what the different chapters contains and shows the structure of the report.

Chapter 1: Introduction Presents the motivation of the research, and the goals associated with it.

Chapter 2: Background and Theory Discuss the theory and technologies that are used in this research, and the previous work it is based on.

Chapter 3: Method Describes the type of research, and how the research was performed.

Chapter 4: Results Detailed description of the different aspects of the implementation, as well as the requirements and goals it tries to solve.

Chapter 5: Evaluation Outlines how well the implementation answers the research questions

Chapter 6: Discussion Summarizes the results found in chapter 4 and discusses future research

Chapter 7: Conclusion Presents a conclusion to summarize the research. 5.

Chapter 2

Background and Theory

In this chapter, the different quality attributes related to the research is defined. Further, the current system, the previous implementation this work is based on and the relevant technologies are outlined.

2.1 Quality Attributes

ISO standards are used to verify if the implementation meet the goals of the research. The ISO/IEC FCD 25010 standard is for software architecture and is concerned with two main points; “Quality in use” and “product quality”. From product quality, the most relevant definitions are the ones related to *usability* and the *maintainability*. As for quality in use, the *effectiveness* is also important for the implementation.

2.1.1 Product quality

Usability

The degree to which a product or system can be used by specified users to achieve specified goals with effectiveness, efficiency, and satisfaction in a specified context of use.

There is no ISO standard for extensibility, but Johansson and Löfgren (2009) defines it as follows:

Extensibility

The ability of a system to be extended with new functionality with minimal or no effects on its internal structure and data flow.

Extensibility relies mainly on three other quality attributes, namely *modifiability*, *maintainability* and *scalability*. In short, the three quality attributes has these three concerns:

Modifiability

A system is modifiable when a change involves the least number of changes to the least number of possible elements.

Maintainability

To design a system to allow for the addition of new requirements without risk of adding new errors.

Scalability

The ability of a system to expand in a chosen dimension without major modifications to its architecture.

2.1.2 Quality in use

The *effectiveness* of a system refers to the distinction between building the system correctly (the system performs according to its requirements) and building the correct system (the system performs in the manner the user wishes) (Bass, 2007).

Effectiveness

Effectiveness is a measure of whether the system is correct.

2.2 The Course

The course wiki¹ is created for the students to have a site where they can find the curriculum, code examples and all their exercises. These exercises and code examples explains both rudimentary programming concepts (like for-loops) and more advanced subjects (object-orientation and even graphical programming, e.g. JavaFX). The TAs and the course instructor (henceforth, “instructors” will mean both) has access to alter every page by updating the content through a admin interface. In order to be able to take the exam, a minimum of 10 assignments must be solved, where each assignment in average consists of five different exercises that the students will find in the course wiki.

2.2.1 Assignments

Since the assignments are used to verify if the students have passed the course and are able to take the exam, each exercise are given properties that relate to

¹<https://www.ntnu.no/Wiki/display/tdt4100/>

the difficulty and the effort it takes to solve it. For each exercise, meta data describing the curriculum coverage, scope and required fulfilment of the exercise are listed in a table, like shown in figure 2.1. These are mainly used by the instructors when creating assignments.

Sidetype	Dekningsgrad	Omfang	Ferdig
oppgave	50	25	95

Figure 2.1: An example of the meta data associated with an exercise.

2.2.2 Exercises

The exercises for the course are diverse, and a lot of them utilize the standalone jar named JExercise² for testing the solutions. This jar contains the JUnit framework and specific utility classes. On the course wiki, exercises will have direct links to files/classes (hosted on the Github page for JExercise). Students have to download the relevant files into their local project, write code to try to solve the exercises, execute tests designed for each exercise, and verify if they have written their code correctly based on the test results. All of these exercises requires the students to write a few classes on their own, which would be quick to compile and run.

In the exercise seen in Figure 2.2, the first section tells the students to create a class named `Account` with the attributes and methods that are described in the text. In the second section, the student is told to write down a state diagram of how they think the `Account` class will behave under a certain circumstance. In the third section, the rest of the exercise is explained. For Figure 2.2, a mock up of how the editor might look like is superimposed in this section. The different classes, running the code/tests, and a possibility to download the relevant files are not shown in this mock-up.

2.2.3 Tests

The code that is provided for each exercise are the tests that the students need to save inside their projects in order to test their code. As well as the classes that test the code, they can also download files that contains a simple, custom language. This is the language that have been used to generate the aforementioned test class, and is designed so that it's more human readable for the students to understand

²<https://github.com/hallvard/jexercise/>

Tilstand og oppførsel - Account-oppgave Tools ▾

1 Added by [Hallvard Trætteberg](#), last edited by [Hallvard Trætteberg](#) on 09.04.2014 ([view change](#))

Opggaven handler om en **Account**-klasse, som håndterer data i en konto.

Tilstanden i **Account**-objekter er som følger:

- **balance** - et desimaltall som angir beløpet som er på kontoen
- **interestRate** - et desimaltall som angir rentefot som *prosentpoeng*.

Account-klassen har to metoder, **deposit** og **addInterest**, med følgende oppførsel:

- **deposit(double)** - øker konto-beløpet med den angitte argument-verdien (et desimaltall), men kun dersom det er positivt
- **addInterest()** - beregner renta og legger det til konto-beløpet

Del 1 - tilstandsdiagram

Tegn et objekttilstandsdiagram for en tenkt bruk av **Account**-klassen. Velg selv en passende start-tilstand for **Account**-objektet og sekvens av kall.

Del 2 - Java-kode

Skriv Java-kode for **Account**-klassen med oppførsel som er beskrevet over.

Lag en passende **toString()**-metode og et [hovedprogram](#), slik at du kan sjekke at oppførselen stemmer med tilstandsdiagrammet (bruk samme start-tilstand og sekvens av kall).

```

1 - public class Hello {
2 -     public static void main(String[] args){
3 -         System.out.println("Hello World!");
4 -     }
5 - }

```

Figure 2.2: An exercise from the course Wiki, with superimposed, colored divisions for the different parts as well as a mock-up of the editor

what the test class is actually doing. Usually, students will develop in an iterative process by alternating between running tests and altering their code to make all tests pass.

2.2.4 Avoiding environment configuration

As described in the report by Rasmussen and Åse (2014) and in the preliminary study (Vikås, 2014), first time setup can be a difficult and time consuming task for a first year student. In order to start writing code, the student must download the correct JDK (Java Development Kit) and configure the system variables/paths.

Then, the Eclipse IDE must be downloaded, and in some cases it has to be configured correctly before it's ready for use. Afterwards, the students have to download a plugin to Eclipse that will perform the unit testing as well as a Java archive (JExercise standalone) that contains the classes that is used for testing their code.

On the course Wiki, the students can find guides to help them configure the development environment. Although the guides are helpful, they are also very long (by necessity). This leaves room for troubles when students have to follow all these steps, as one mistake along the way could lead to the student being stuck and not know how to both find the mistake and correct it. Even if the students can configure the environment, they will still have to add the JExercise archive for each assignment that requires it. This isn't necessarily hard, but it do take some extra time each time.

2.3 Eclipse

Eclipse is an Integrated Development Environment (IDE) that have been created for developing applications. It is used by millions of software developers, and is built upon a modularity infrastructure. By using Eclipse Rich Client Platform (RCP), developers can create new applications by including any modules from the Eclipse base. In order to load these module in to an application, the Plug-in Development Environment (PDE) package must be installed as part of the Eclipse installation. This package is needed for creating bundles that can run in an implementation of the Open Service Gateway initiative (OSGi) specifications. Equinox is an implementation from the OSGi-specifications, which is what Eclipse is built on (McAffer, VanderLei, & Archer, 2010).

2.3.1 Eclipse Java editor features

In Eclipse, there are many features which will help the developers write code faster and with less keystrokes. Considering that the implementation should avoid having too many features that would confuse the students and/or slow down the system, only a subset of these are included as part of the implementation.

Content assist

One of the basic forms of help that Eclipse provides to developers are content assists. It provides the possibility to complete class names, method names, parameter names and more. Code assists also allow for defined keywords to be expanded into any of the items mentioned, such as the keyword "NPE" that can be expanded into `NullPointerException` (Clayberg & Rubel, 2008).

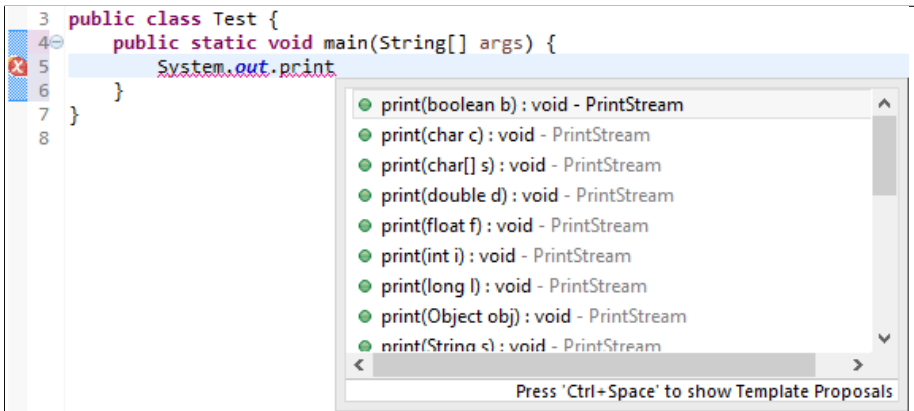


Figure 2.3: An example of the use of content assist with `System.out`.

In figure 2.3, content assist has been invoked on the context of `System.out..`. The output here are a list of the different methods that are available, the type of parameters it accepts, and the value it returns.

Templates

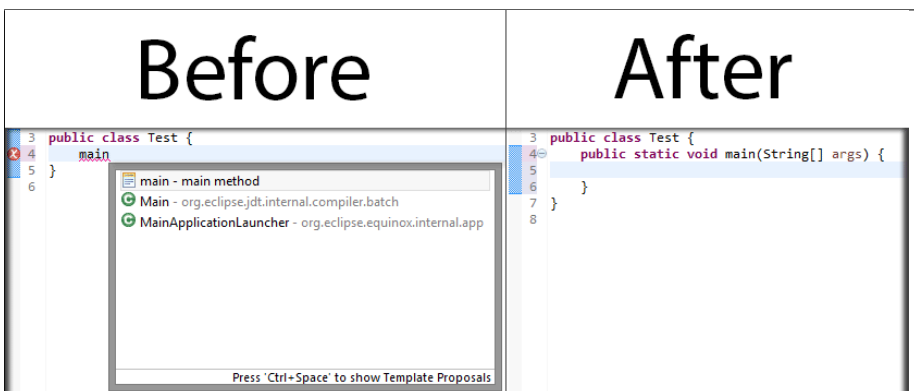


Figure 2.4: An example of the use of templates. Left side: Initiated code suggestions. Right side: “main” template chosen.

The Eclipse IDE also provides templates, which can generate source code patterns directly in the code. A common template keyword is “sysout” which

can generate `System.out.println()` directly in the code. There also exists more advanced keywords, e.g. “main”, which will create a whole method block.

Figure 2.4 show how the state of the code in Eclipse changes when utilizing templates. Both content assist and the code assist features are activated by using the hot-key combination **Ctrl+Space** (Clayberg & Rubel, 2008).

Built-in Java compiler

According to the Eclipse JDT Plug-in Developer Guide³, Eclipse uses a built-in compiler to build code incrementally or in batches. This builder has smart features, such as not only checking if the code is syntactically correct but also semantically. The syntactical analysis can see if there is an error in the syntax, but it doesn’t check for the semantic validity. For example, a programmer could write syntactically correct code that uses a method that should exist in another resource in the class path, but if that method doesn’t exist, the code would fail when it’s executed. Having the check for semantic validity will give feedback to the programmer of the error in the code without having to compile and run it first.

Error/Warning markers

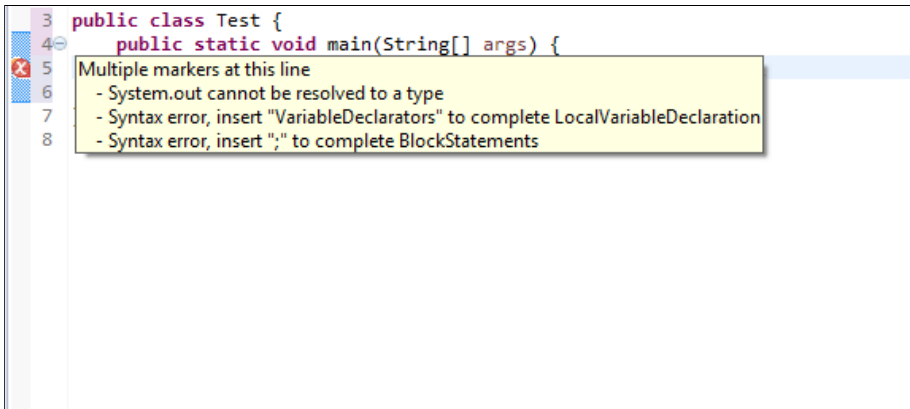


Figure 2.5: An error marker will show additional info when the programmer hovers over the marker icon.

A common feature for IDE’s is to have error checking of the code, which is not often found in text editors. When a programmer creates an error in the code,

³http://help.eclipse.org/indigo/index.jsp?topic=%2Forg.eclipse.jdt.doc.isv%2Fguide%2Fjdt_api_compile.htm

Eclipse will denote the line with the problem and give an explanation of what is wrong (Rasmussen & Åse, 2014). By utilizing the built-in Java compiler in Eclipse, both semantic and syntactical errors can be displayed to the programmer. The markers are invoked every time a programmer alters the code after a set interval from the last key-stroke.

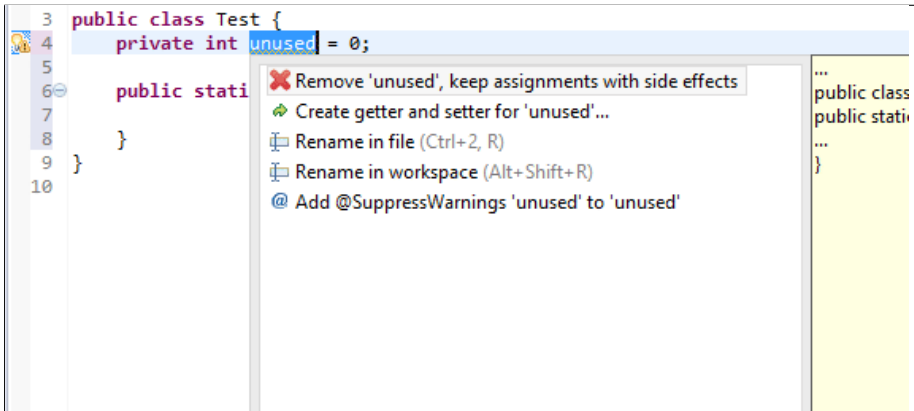


Figure 2.6: Warning marker that are being rendered because the programmer have an unused variable in the code

Another feature that the built-in compiler supports is to give warnings, which is executed in the same way error markers are. These will make the developers aware of any mistake they might have done, but it still means that the code will compile and run. An example of a warning is when the programmer creates a variable which is unused. In figure 2.6, an example of a warning marker from having an unused variable is shown.

2.4 Assignment System prototype

The artifact from the research by Rasmussen and Åse (2014) was a standalone assignment system to allow for a web editor to connect with Eclipse. As a standalone system, it consists of two parts; The web editor itself and the back end that handles the work students do with assignments. The Assignment System prototype have been created as a first step in providing more development features directly in the browser. Communication from the web editor goes through WebSocket on the web page via a Play Framework⁴ web server. The web server initiates the connection with the Eclipse instance through a socket consisting of

⁴<https://www.playframework.com/>

WebSocket and the Akka Framework⁵. This make the web server responsible for relaying all communication between those two endpoints (Rasmussen & Åse, 2014).

The prototype had many of the required features that a final implementation would need, but it had some problems that made it difficult to use the whole system when performing this research. Firstly, it was hard to work with due to its unnecessary complexity and its lack of modularity, which makes it hard to use on different platforms. Secondly, it used the Play Framework for providing a whole web site, which wouldn't fit properly in the course wiki that is built on a Confluence server.

2.4.1 Gamification

The supervisor wanted to support gamification. The prototype included gamification aspects, which was created to encourage learning. An aspect of gamification is to award the work students do, e.g. giving badges to students who have solved exercises and displaying a list of the students who have solved most exercises/assignments.

2.5 Scalability

Scalability is a quality attribute to a system. The two main kinds of scalability are horizontal- and vertical scalability. The horizontal scalability refers to adding more resources to logical units, such as adding more units to a cluster of servers. Scaling up (vertical scalability) refers to adding more powerful components to a unit, for instance adding more RAM to a server. One of the problems with scalability is to do it effectively, which means that the system should be able to scale without having to disrupt operations for the users and gain a measurable improvement of the system (Bass, 2007).

As discussed by Rasmussen and Åse (2014), the latency in communication between the student on the course wiki and Eclipse grew linearly with each new client and isn't a big performance problem. Creating a new exercise would take a lot longer, as the prototype would have to create a new instance of Eclipse for each project. The suggested way of mitigating this problem was to only have one Eclipse instance running on a server and create a new project within the workspace instead.

⁵<http://akka.io/>

2.6 Confluence

The course wiki is built upon the Confluence framework by Atlassian⁶. One of the great aspects of the framework is its openness that allows users to expand upon the initial product (Koplowitz, 2010, p.7). Although Confluence is meant for collaboration between teams in an enterprise environment through wiki pages, it's used for giving relevant information to the students in this course.

2.6.1 Page editor

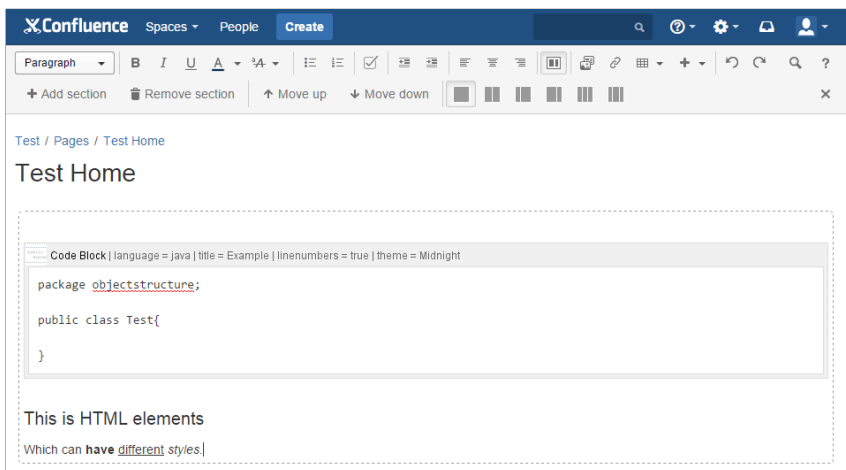


Figure 2.7: The page editor, with both standard HTML-elements and a macro for including code blocks.

The page editor in Confluence is shown in figure 2.7, which include both simple text that has been formatted with HTML as well as the more advanced Code Block macro. Confluence doesn't expect the users to learn markup language for inserting different objects and styles, but relies on a WYSIWYG-approach (What You See Is What You Get). For authorized people who creates and edits pages, this means that they can see the result of the alteration in real time before submitting the change. There are some content that need to be altered when the page is accessed, and for that the Confluence supports macros.

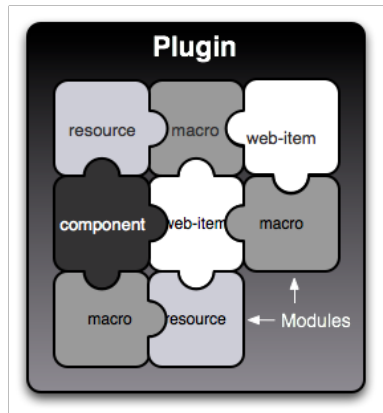


Figure 2.8: A simple illustration of how different modules can coexist inside a plugin

Plugins

To extend Confluence, the easiest and recommended way is to create a plugin. A plugin consists of a combination of modules that each serve individual purposes, which can be tied together to create new functionality (see Figure 2.8). There exists more than 30 different predefined modules that a developer could use as part of a plugin, and one of those are macros. The macros enrich the page content in an easy and modular way by having the possibility to attach independent meta data to every instance of a macro, so that it can render the content in a unique way. It is up to the developer of the plugin to decide what the macro shall render, which means that the raw input data does not need to be shown to the user. The editor in this research will use the content from a macro to configure the project.

Another module that Confluence provides is the REST plugin module. The module provides a HTTP path on the server that plugins and scripts can communicate with, and the module can give a response in the form of JSON and/or XML-data. Since the REST API is an open standard, any web development language can access the REST module.

2.6.2 EMFS

To make the process of downloading the relevant files for an exercise, the students can use an Eclipse plugin that is linked to in the course wiki. This plugin will automatically download the required files and configure the project locally by

⁶<https://www.atlassian.com/>

using EMF File System (EMFS), a project developed specifically for creating new projects in a Eclipse workspace. The EMFS project uses EMF and Xtext, which are tools to generate code from a simple Domain-Specific Language. The DSL-language, Xtext based EMFS (XEMFS), is parsed by a parser inside the EMFS project, and the project contains models that the XEMFS can map to. XEMFS describes the file and folder structure in a project, and can fill the content of the files either inside the description or by providing links to external resources. The XEMFS-language can easily be expanded to support new features, as long as the parser and models in EMFS supports it.

Eclipse Modeling Framework and Xtext

Eclipse Modeling Framework (EMF) is a framework for making model-driven development less time consuming. In short, model-driven development consists of two aspects; Creating models and processing a model, e.g. generate code. Instead of creating UML-diagram, XML-representation of the model, and the Java interfaces for the data model, developers can create a EMF-model and generate the other artifacts from the model. The goal of EMF is to be able to both create the code and the model at the same time, as well as being the middle ground between the extremes of having full model-driven development and “the code is the model”-approach to development (Steinberg, Budinsky, Merks, & Paternostro, 2008).

Xtext is a framework that supports the creation of textual Domain-Specific Language (DSL), which can be used to generate code. By using Xtext, developers can create a grammar that can be parsed and mapped to models and properties in the form of an Abstract Syntax Tree (AST). This AST can then in turn be used to map to a previously created model (Efttinge & Völter, 2006).

In short, developers can use Xtext to create a custom language that can be parsed to generate code.

Chapter 3

Method

This chapter will discuss the method used for this thesis and implementation project.

3.1 Development method

Due to the fact that the final result from this project is an artifact/product to be used with IT technology, the "Design and Creation"-method have been chosen as the strategy for this thesis. The strategy requires either *constructs* (entities, objects, data flows), *models*, *methods* (algorithms, patterns) or *instantiations* (a working system that demonstrate the constructs/methods/etc.), but usually it's permutation of all of these (Oates, 2006).

To make sure that this project is not only a solution to a technical problem, the research have to include academic qualities such as analysis, explanation, argument, justification and critical evaluation. The project must also, in some way, contribute to the knowledge of science (Oates, 2006).

The strategy is typically a problem-solving approach to science, which uses an iterative process in five steps:

Awareness Recognizing and articulate the problem

Suggestion Proposing a way to solve the problem by creativity

Development Implement the suggestion

Evaluation Asses the worth/deviations of the implementation according to the initial goal

Conclusion Summary of the design process, all new knowledge and (potential) loose ends

3.2 Literature

In order to obtain relevant articles, the Google Scholar¹ service was used. Since this service uses search queries to give a list of articles, different search terms have been applied in order to find the relevant articles. One of the plus sides of using Google Scholar is that it's easy to find articles about mainstream subjects, but on the other side it makes it harder to find articles in new fields. This is due to the Google Scholar ranking system (Beel & Gipp, 2009).

3.3 Data gathering

In order to obtain data for this project, usability testing were performed to analyze different aspects of the implementation. Such aspects include the usefulness, speed of operation, and how easy the artifact is to use for the participants. As the artifact's target group is students that are considered "beginners" in computer programming, the participants in the user testing were all first year students from the masters study in computer science. These students have recent experience in working with the assignments on the current system, and is therefore in a good position to compare the old system with the new implementation. According to Whitehead (2006), five users should be sufficient to check whether a feature is a problem or a success. Due to constraints in time and resources, only five students were included.

3.4 Usability testing

In order to assess the usability of the final product, a usability test was conducted. (Rubin & Chisnell, 2008) explains what usability is in the following way: "In large part, what makes something usable is the absence of frustration in using it.", and goes on to define usability as "[when] the user can do what he or she wants to do the way he or she expects to be able to do it, without hindrance, hesitation, or questions". Usability testing are used to collect empirical data by testing the product on a representation of end users while they do realistic tasks. There are two main approaches to usability testing; Checking the system with formal tests to see if the system confirm/refute specific hypotheses or use an iterative cycle of tests (User-centered design (UCD)) in order to mold the product in to a usable final product.

¹<http://scholar.google.com/>

3.4.1 Types of testing

There are three types of testing that are part of usability testing; Exploratory (or formative), assessment (or summative) and validation (or verification). For the exploratory testing, (Rubin & Chisnell, 2008) states that this type of testing is performed at an early stage to see “whether the user intuitively grasps the fundamental and distinguishing elements of the interface?”. Exploratory testing is in regard to the high-level aspects of the system to see how the user perceives the effectiveness of using the product. This means that user only needs to be exposed to enough functionality to be able to test the system, which can be a mock-up/prototype of the product and all its functionality. It is encouraged to have an open dialog with the participant so that the participant might, for example, come up with ideas on how to improve the product. This helps form the final design.

Assessment testing is about testing if the user can perform a full-blown realistic task, rather than just exploring the intuitiveness of the product. It evaluates the usability of the lower-level functions of the system and see how efficiently it has been implemented (Rubin & Chisnell, 2008).

For this research, the usability testing approached the exploratory testing type with aspects of assessment testing. The product is a working prototype, but all the functionality of a final system has not been implemented. This means that the testing was performed as an exploratory test, but the tasks the user had to solve was related to realistic tasks (assessment testing).

3.4.2 Performing usability test

User testing was conducted with an observer present to guide the users through the testing. As no personal information is required in this research, the users that participate are anonymous. The way the testing was performed was by first introducing the users to what this implementation tries to achieve. Then, they were presented with different tasks that they tried to complete. Even though an observer was present, the observer would only give small hints if the user was stuck, and only answered questions that did not reveal how to solve the whole task. For each task, the observer wrote down notes on how well the user handled the task and any useful points that occurred in the dialog with the participant.

3.4.3 Post-testing survey

After all the tasks had been completed, an evaluation of the system was performed in the form of questions from the observer. These questions was in the form of open questions on the different aspects of the system to (possibly) get more

information than what was written down during the task-solving phase. The following list contains all the different questions asked to the participants:

- Was there any problems/difficulties when using the product that prevented you from solving the tasks?
- Did you find the product easy to work with? If so, why?
- Was it easy to understand and use the test panel for solving the tasks?
- What did you think about how the resources was managed in the system?
- Is there anything else about the system that you would want to have been handled differently?
- If you imagine yourself as a new student in this course, would you consider using this product instead of the current implementation?

The usability testing was performed in Norwegian, but all the results have been translated in to English.

Chapter 4

Results

4.1 Overall design

This chapter will describe the overall design and flow of the system, which involves the interaction between the front end, back end and all other related services that makes the implementation work.

The system is split in two main parts; The implementation in the course wiki as a plugin (named *Confluedit*) and the implementation which serves the web editor (named *Eclipsky*). *Eclipsky* is the main back end solution that handles all the operations related to exercises, but it also provides a presentation layer by serving the web editor. The following lists shows how the different responsibilities is divided in the system:

Confluedit

- Retrieving a unique user identification based on either user id (if logged in) or session id
- Storing exercise description to be sent to the back end when establishing a connection
- Determining which Eclipse instance to send the request to.
- Inserting an `iframe`¹ on the page that the user uses to interact with the backend.

¹http://www.w3schools.com/tags/tag_iframe.asp - A HTML-tag that indicates a inline-frame on a web page

Eclipsky

- Converting exercise description to project resource
- Handling HTTP-requests, XMLHttpRequests and WebSockets
- Compiling and running code based on requests from the user
- Provides a graphical web interface for each project

For a overview of the system, figure A.1 shows the flow of operations when a student wants to work on an exercise.

4.2 Requirements and constraints

All the requirements and constraints for the implementation in this research are discussed here. Since the requirements/constraints have different aspects, they have been divided in to these groups:

EF Editor Functionalities

EP Editor Performance

IC Implementation Constraints

4.2.1 Editor Functionalities

As described in Goal 2 Student Usability, several IDE-features should be part of the editor. The list below are labeling the different editor functionalities (hereby named EF) that have been selected to be part of the system:

EF1 The editor shall support syntax highlighting

EF2 The editor shall support content assist

EF3 The editor shall support error markers

EF4 The editor shall support warning markers

EF5 The editor shall provide a resource navigator

EF6 The editor shall provide a “run code” button

EF7 The editor shall provide a console output

EF8 The editor shall provide a “run test” button

EF9 The editor shall display test errors

EF10 The editor shall display failed tests

EF11 The editor shall display passed tests

4.2.2 Editor Performance

It's not enough that the editor has the functionality of a local IDE, it must also feel like a local editor when it comes to the performance. Since there are many factors that can impact the performance of an application over the Internet, the measurements listed here are for the context of running the application locally. Still, some resources (like the test classes) needs to be downloaded and will have some impact on the performance.

Another factor that isn't part of the requirement is the effect of the system with many users. A future system should support scaling and be able to start new servers dynamically if the load on the system is too high.

EP1 Starting a project should take less than one second.

EP2 Basic editor tasks should take less than *250ms*.

EP3 Running code/tests should take less than *500ms*.

4.2.3 Implementation Constraints

Since NTNU wishes to use this implementation and develop it further, a set of constraints about the technologies to be used in this implementation must be met.

IC1 The server side code must be written in Java

IC2 To create projects, EMFS must be used.

IC3 Eclipse's JDT must be used to manage the workspace and editor functionalities.

IC4 Students must be able to access the web editor from a Confluence server.

IC4 Instructors must be able to insert XEMFS into a Confluence macro.

4.3 Confluedit – Instructor’s viewpoint

The Confluedit part of the system must be installed in a Confluence environment. From this environment, the instructors in the course are able to enter information inside a Confluence macro when editing a page, which students can see when visiting the page and optionally start the exercises from a button on the page.

One of the responsibilities for the Confluedit plugin is to find an Eclipse instance to pass a request to when a student wishes to work on an exercise. In the current implementation, it only looks for a Eclipsky instance in local host.

4.3.1 Creating the macro

When creating the macro for Confluedit, the following information must be entered in for it to work:

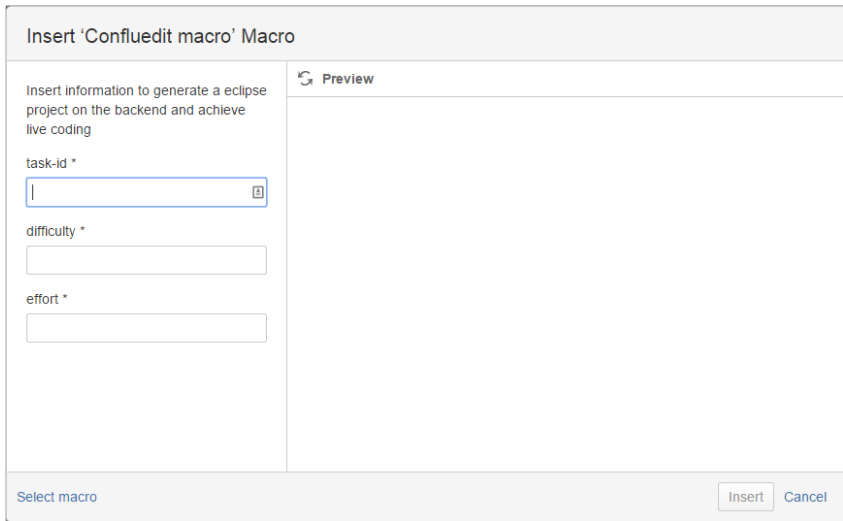
1. taskId
2. difficulty
3. effort
4. xemfs

The meanings of these properties are discussed in section 4.6.

From the figure 4.1, the different properties that are part of the Confluedit plugin (except *xemfs*) are shown. After the instructor have entered relevant information and clicked the “insert” button, the macro editor with an empty body is displayed. In figure 4.2, the content of the body has been filled with a valid XEMFS description. The purpose of the XEMFS description is to accommodate both the Goal 3 Instructor Usability by offering an expressive language to define a exercise. The macro body does not provide syntax-highlighting for writing in XEMFS, but it does not need to. Eclipse supports the possibility of writing the XEMFS code and verify that it is correct before pasting it in Confluedit, and its how the instructor writes the code now. The language itself is further discussed in section 4.6.4.

4.4 Confluedit – Students’ viewpoint

For students, the Confluedit plugin will only appear when they are visiting a page in the course wiki that the instructor have included to the page. When a student enters the page, the only visible part of the macro is a rectangle with button(s) inside it. A button to start the project is always present, but if the user isn’t



Insert 'Confluedit macro' Macro

Insert information to generate a eclipse project on the backend and achieve live coding

task-id *

difficulty *

effort *

Select macro

Insert Cancel

Figure 4.1: Different parameters that must be entered into the plugin

logged in to the system, a log-in button will be displayed as well. As it is now, all properties that macro have are rendered on the page with hidden elements. The difficulty and effort properties are not used, but are there for later development.

One of the requirements of the implementation is that a student should be able to work on an exercise without having to log in. If a student only want to work on an exercise without it being related to their account, the project in Eclipsky will be associated with the students current session id in Confluence. The problem with session id in this context is that the student might not get back what they worked on at a later time (see section 4.6.1).

To ensure that a student can work on a project that have been previously worked on, it is recommended to be logged in. Every student in the course has an account in Confluence, with a unique id attached to it that Confluedit can fetch. This unique id will never change for students' accounts, so they are guaranteed to get their previous work back.

4.4.1 Initializing project on Confluedit

When the log-in button is clicked, a JavaScript-script will execute and get the hidden information from the macro that is rendered on the page. This information is then passed on to a REST-service within the Confluedit plugin via an AJAX-call. Inside this service, it will check if the user is logged in, and if so, load the user id from Confluence. If the user isn't logged in, the user's session id will be



```

emfs $testspath="no.hal.jex.collection/src-gen/objectstructures"
/
src/objectstructures/
Card.java*
8<---
package objectstructures;
public class Card {
}
--->8
CardDeck.java*;
CardHand.java*;
;
;
tests/objectstructures/
CardTest.java @ git@github.com:hallvard/jexercise/$testspath/CardTest.java;
CardDeckTest.java @ git@github.com:hallvard/jexercise/$testspath/CardDeckTest.java;
CardHandTest.java @ git@github.com:hallvard/jexercise/$testspath/CardHandTest.java;
;
;
.project: dot.project #java;
.classpath: dot.classpath
source-folder src
source-folder tests
source-folder /jexercisestandalone
class-container org.eclipse.jdt.launching.JRE_CONTAINER
output-folder bin
;

```

Figure 4.2: The content of the macro with XEMFS-language. The file structure is described, where the content can either be in-line or downloaded.

loaded instead. Now that all the information required is present, a HTTP-call to an instance of Eclipsky is initiated. Eclipsky will then respond with a redirect to a URL that contains the requested project, which Confluedit will send back as a response to the AJAX-call. On the users page, the current HTML for the Confluedit macro will be replaced with an HTML `iframe`. The `iframe` will be directed to the URL that was returned from the AJAX-call, so the user can see the content from their project in Eclipsky inside the course wiki.

4.5 Eclipsky

This section is in regard to everything that happens on the back end, which involves every responsibility mentioned in section 4.1.

Eclipse have been chosen based on the previous implementation discussed by

Rasmussen and Åse (2014) and Vikås (2014), which this report is based on. By having Eclipse as the base platform, the supervisor required the use of OSGi to create the implementation discussed in this report. The previous implementation relied on other technologies, such as a standalone web server to communicate with Eclipse, but these technologies have been not been included. Instead, a web server have been included as a OSGi bundle (amongst other bundles) to create a simpler system to manage, since everything resides in the same application.

There is an important distinction to make her between the words *editor* and *web editor*. The *web editor* means the whole page that is served, and it includes the *editor* (and other elements). The *editor* is where the student can write code on the web page.

4.5.1 Serving web page

The web server included in Eclipsky is Jetty, which has the design motto “Don’t put your application into Jetty, put Jetty into your application.”. It supports many new web features (such as WebSockets), and is designed to have a small memory footprint². As Eclipsky only need to serve one page, the web editor, a small and simple web server is an appropriate solution. This should also help with EP1 (start performance) if a Eclipsky instance need to be started on a new server in a cloud solution.

When students starts exercises from Confluedit, the web editor page that will be presented to them look like the example shown in figure 4.3. After the exercise is completed, it might look something like figure 4.4.

Editor

The editor functionalities EF1 (syntax highlighting), EF2 (content assist), EF3 (error marker) and EF4 (warning marker), as seen in figures 4.3 and 4.4, are all part of the editor. The base editor is the Ace editor written in JavaScript. It is recommended and used by Rasmussen and Åse (2014), which have been utilized to accommodate the requirements for this implementation.

When a student presses **Ctrl+Space**, Eclipsky will give the student content assist (EF2) about which methods are available in a list. If the student has errors in the code, an error marker (EF3) is displayed, while warnings will display a warning marker (EF4). By hovering over this item in the editor gutter, information about the error or warning is displayed to the user, as seen in figure 4.4.

As discussed in the background regarding error/warning markers in Eclipse, the code will only be evaluated after a set interval from the last keystroke (see section 2.3.1). This concept is also used in the editor. The code is only transferred

²<https://webtide.com/why-choose-jetty/>

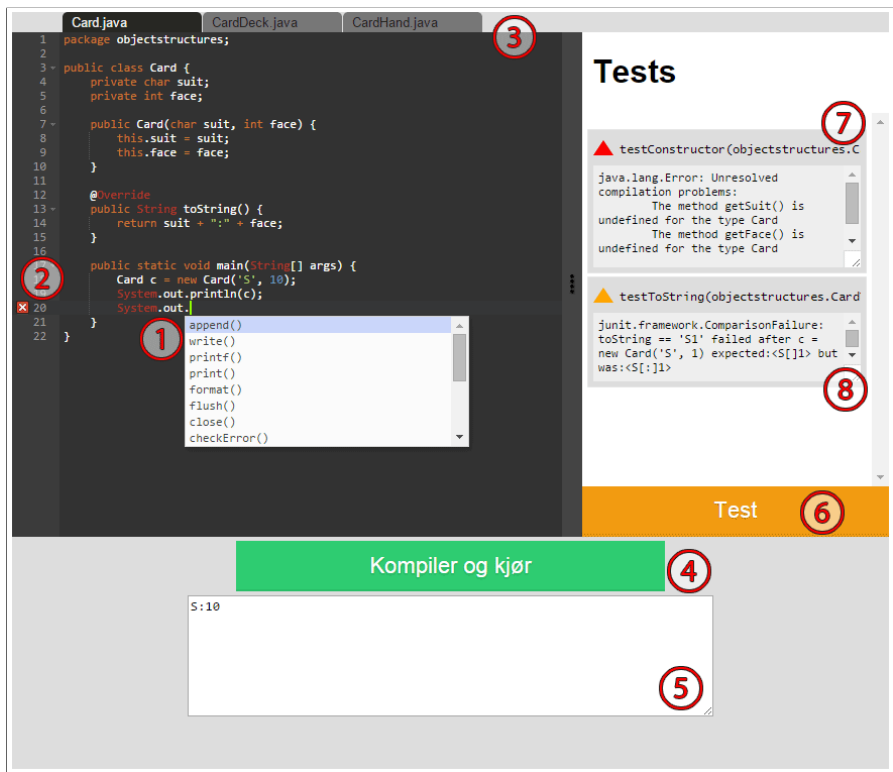


Figure 4.3: The web editor with the different functionalities labeled with numbers

to Eclipsky after a interval of $500ms$ from the last keystroke, and Eclipse will generate any potential error/warning markers and return it to the editor.

Resource navigation

As the exercises requires the student to work in multiple files, the resource navigator is located at the top of the editor. This is to reflect the interface of the Eclipse editor which also manages opened files above the editor.

Test panel

In order for students to verify that the code they have written is correct, they will need to be able to run tests and see the output. When a student clicks the test button, the test panel will be filled with tests results. Each of the individual tests



Figure 4.4: The web editor with all tests passed

will return in one of three states; *error* (EF10), *failed* (EF11) and *passed* (EF11). For error and failed, the result can be expanded to display a message about what went wrong.

Running code/testing

If students wishes to run code in the current resource, pressing the run button (EF6) will display the result inside the console output (EF7), as shown in figure 4.3. In order to run code from the different resources, the student must switch to the desired resource and press the run button from that state.

The testing works much in the same way as running code. This means that each test run is done in each resource, and that the result displayed in the test panel is related to the current resource.

Connection

When the page is loaded, a connection to the back end is attempted. If the browser doesn't support WebSockets, it will use a fallback to `XMLHttpRequest` to handle the communication. All communication is handled by a JavaScript-

module that implements the pub/sub (publish/subscribe) pattern, as well as FIFO (First In, First Out) stack for communicating with the server.

By having a pub/sub pattern, any module can subscribe to the events and handle it in different ways. For instance, when a student runs tests and the result comes back from Eclipse, the editor module will cache the result while the interface module will render it on the page.

Since the web editor can't know about the state in Eclipse, a FIFO stack have been implemented to try to avoid the two systems having different states. The stack contains all the latest commands to be sent to Eclipse, but the next command in the list won't be sent until the last command returns. For example, if a user alters the resource and runs a test right after, the test command won't be sent until Eclipse returns a list of error/warning markers (potentially an empty list). This should increase the chance of the commands running at the same state on Eclipse as it happened in the web editor.

4.5.2 Compiling and running code

In order to run code in a Java class, it needs a `main`-method. If the student wishes to run the code in the current resource they are working with on the web, this method needs to be implemented (or else an error message will return in the console output). This is not necessarily the case with other programming languages, so no checks on the existence of a `main`-method is included.

4.5.3 Towards cloud service

As part of the Goal 1 Student Availability, the project should be available for all students that access the course wiki. Although a fully scalable system is not implemented, some of the base functionality is in place. The bundles `git` and `mqttlogging` (with prefix `no.hal.eclipsky.services.`) are part of the approach to scalability and establishing a cloud service. Unless otherwise stated, the bundles will be written in the report without the prefix.

A bundle is package of binary code that can run independently in a OSGi-environment, yet be dependent on other bundles or have other bundles dependent on it. OSGi, in short, is a framework with an application that manages bundles (more on OSGi in section 4.5.4). For the bundles that exists in an OSGi-application, accessing other bundles can only be done by an interface that they expose outwards. The bundle itself does not have any *main* program, but can start threads and access both internal (the OSGi-application) and external resources, e.g. databases or network communication. During the lifetime of a application, the bundles can be installed, uninstalled or updated at any time. This means that the bundles themselves must implement ways to deal with these events in the application.

There were 514 students registered in the course in spring 2013, according to the preliminary study (Vikås, 2014). In order to have a system that responds to Goal 2 Student Usability, the server shouldn't have too long response times that would affect the usability of the system. For that reason, a requirement for the system is to be easily made horizontally scalable in the future by adding new servers if the load on a server is too high. To communicate what the current load of a server is, and to be able to determine which server to use when a student starts working on an exercise, the MQTT-protocol was selected by the supervisor. In its current state, the MQTT-client in Eclipsky only submits events to the broker.

MQTT is a lightweight, binary protocol that implements the classic pub/sub (publish/subscribe) pattern with a central broker. The protocol is based around the concept of *topics*, which clients can either subscribe to (i.e. receive updates from other clients) or use them to publish updates (Collina, Corazza, & Vanelli-Coralli, 2012).

One instance per server

As discussed in section 2.5, the most important thing to fix was to have one instance of Eclipse running on the server, and have projects sharing that instance. This is the way the system works now as a OSGi-application (see section 4.5.4).

By having just one instance of Eclipse, all projects will exist in the same workspace. One of the benefits of having this solution is that there only need to exist one project in the workspace that contains all required files that the projects needs. As well as making the first time creation of projects faster when Eclipsky don't have to download these files, it will also take up less space on the storage drive.

Persistent storage

As discussed in the preliminary study (Vikås, 2014), the code from the exercises that the students work on should be stored on the server so that they are able to get it back when they open up the exercise again.

If Eclipsky runs in different servers, every time the student want to work on a previous exercise, code that the student has already written must be loaded into the server the broker selects. No implementation of a persistent storage is currently in this implementation, as it isn't required for running a single instance of Eclipsky.

4.5.4 OSGi-runtime

In the workspace of Eclipse, a target platform defines all the bundles for which the code will compile with. Which bundles to be included under compilation can be selected from Eclipse, which, by deselecting unwanted bundles, can give speed improvements when compiling the code. To be able to build bundles in Eclipse, any packages that include the Java Development Tools (JDT) or Plug-in Development Environment (PDE) can be used (McAffer et al., 2010).

The JDT implements both the UI-components in Eclipse and the underlying core components. When developing a plug-in for Eclipse, these bundles becomes available to the developer.

In order to use these tools, the code for the back end consists of multiple bundles that can run inside the same OSGi-application. These bundles, with full name, include the following:

1. no.hal.eclipsky.services.emfs
2. no.hal.eclipsky.services.git
3. no.hal.eclipsky.services.monitoring
4. no.hal.eclipsky.services.mqttlogging
5. no.hal.eclipsky.services.workspace
6. no.hal.eclipsky.services.workspace.http

Each bundle have their own responsibility in the system, and the internal communication is handled by Equinox (the OSGi-application).

Bundles

In section 4.5.3, the properties of a bundle was explained. Since bundles can be replaced during run time, new versions of the components can be added without having to stop the service. The following list will explain briefly what each bundle's responsibility is:

no.hal.eclipsky.services.emfs

The emfs-service provides an interface to import resources in to the project, either from the file system, network communication or from a description of the content inside the Xtext description.

no.hal.eclipsky.services.git

This bundle is used to retrieve and store resources to git.

no.hal.eclipsky.services.monitoring

Receives information about request and responses when requests are sent to the communication components (see section 4.7).

no.hal.eclipsky.services.mqttlogging

Receives the same events as the `monitoring-bundle`, and communicates these with a MQTT broker.

no.hal.eclipsky.services.workspace

Provides an interface to interact with the Eclipse workspace

no.hal.eclipsky.services.workspace.http

Uses a Jetty Web Server to communicate with HTTP/WebSocket, and provides the communication components defined in section 4.7.

Each bundle contains description of the different components inside that the OSGi-application uses for managing the system. These components usually provides an interface class that other components inside the bundle (or other bundles) can use, and a component-specific implementation of that interface. For all the bundles shown in figure 4.5, a relation arrow points to the different components it provides and the name of the interfaces. Both `monitoring` and `mqttlogging` uses the `ServiceLogger` as an interface to receive events, and handles them in their own way. For the `EmfsService`, the `addImportSupport()`-method accepts `ImportSupport`-objects in the none-to-many relation. This relation tells OSGi that the interface in the component can be called many times with `ImportSupport`-objects. There is also a `remove` method in negation with every `add`-method, except for methods that start with `set`.

Since the `workspace.http`-bundle contains 12 components, a second model has been created. Figure 4.6 shows the components, with the same way of describing each as with figure 4.5. An exception to the interpretation of the description is that some of the components have the same interface, so the implementation class has been used instead. This is the case for all the grey and blue components, which all have the `SourceEditorServletService`-interface. The blue components (the bottom row) are used with running the project, and the grey components (row above the blue components) are related to the basic editor tasks. Another set of components that exposes the same interface are those in purple, located in the topmost row. These implement the `ServiceServlet` interface. Another exception to the interpretation is the `WorkspaceHttpService`-component in dark orange, which doesn't use an interface but exposes itself.

In order to support Goal 4, the architecture of the workspace-bundle uses generic project elements that are extended to be specific for Java. These generic elements could be used to create new project types in the future.

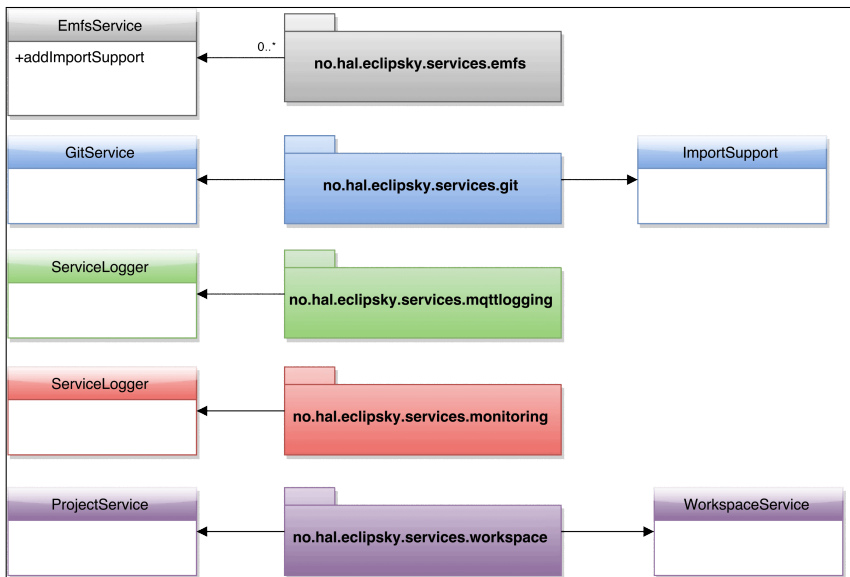


Figure 4.5: All bundles (except `no.hal.eclipsky.services.workspace.http`) with their components

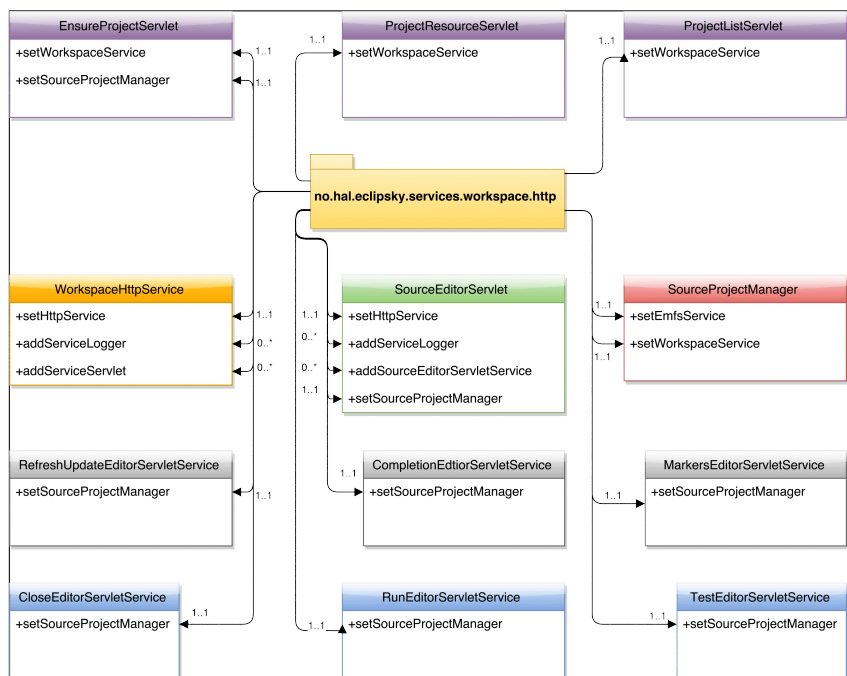


Figure 4.6: The no.hal.eclipsky.service.workspace.http bundle with all its components

4.6 Properties of a project

Each project has properties associated with it that have different responsibility. The *task id* is used to identify projects, and the *xemfs* is used for generating a project. *difficulty*, *effort* and *editable* are properties that is not related to the project files in Eclipse, but related to the exercise.

4.6.1 taskId

As the system needs to provide the possibility for students to open up a project that they have previously worked with, a unique identifier must be generated for each student on each exercise. If the student doesn't want to log in to the course wiki, the current session id for the student will be chosen instead. The unique project name will be constructed by concatenating the students id (either from session or from being logged in) with the task id using a hyphen (-), represented as `$userId-$taskId`.

4.6.2 difficulty

Each exercise has a *difficulty* property to it which the instructor must assign a value. This value can be changed without affecting the back end, as it is meant to give the student visiting the course wiki an idea of the difficulty of the task. The value is in the form of a percentage.

4.6.3 effort

To give the students an estimate about the time required to solve an exercise, the effort property tells the student what time the instructor thinks the student would need.

4.6.4 xemfs

The xemfs property mentioned in section 4.3.1 is part of the EMFS project discussed in the earlier work (section 2.6.2). It provides a language and the ability to convert from the DSL-language to instances of classes, but the specifics won't be discussed further in this report. The purpose of using EMFS is to accommodate Goal 3 Instructor Usability by offering a flexible way to construct exercises from the web, which can automatically create a project in Eclipse based on the description.

4.7 Communication components

All components that handle the communication between the Eclipse workspace and the web editor exist inside the `no.hal.eclipsky.services.workspace.http-bundle`. Even though each component have individual tasks, they can utilize the other components to complete their task.

4.7.1 Refresh/Update component

As the name implies, this component handles refresh and updates. When a student starts an exercise or switches between resources, a **refresh** message is sent to the back end containing the name of the requested resource. Whenever a student alters the content of the resource in the editor, an **update** message with the name of the altered resource is sent to the back end. The response from the back end is a list of error/warning markers for the resource in its altered state.

4.7.2 Close component

In order to switch between resources and build/run them, the resource must be closed. This is because Eclipsky creates a `WorkingCopy` of a resource, which is an object in memory with the content of the file. This service is used to save the `WorkingCopy` of a resource to the file system, and is used by the Run and Test components.

4.7.3 Completion component

This component provides the content assist suggestions to the web editor for a given offset in the resource. The way the implementation uses content assist is by getting proposals from the JDT through a `WorkingCopy` (see section 4.7.2). These proposals exposes, among other things, the name, relevance, modifier (the context of a proposal) and the signature (parameter(s) and return type). The signature is in the form of a string that must be parsed to become human readable, and has not been included in the implementation.

4.7.4 Run component

When the run button is pressed on the web editor, this component will get called and run the current resource that is open in the editor. It will look for a run configuration of the current resource, and generate one if it doesn't exist, before running the code as-is. The component will respond when the process is done running.

4.7.5 Test component

For running tests, a test runner is required. The test runner need to know the test suite and the test class to run the code on, and is independent of each students' exercises. In the workspace of Eclipsky, there is a reference to a project which contains all test suites, the JUnit framework and a test runner (see section 4.5.3). When a student presses the test button on the web editor, a similar process to the run component will be executed in Eclipse. The difference is that instead of launching the resource, the test runner will be executed with the relevant test suite as well as the class to run the test on passed in as arguments.

Chapter 5

Evaluation

In order to assess how successful the implementation both satisfies the requirements and the main goals of the research, an evaluation was performed. It's mainly divided in two parts; Analytical evaluation and qualitative user testing. The analytical evaluation will look at how well the implementation satisfies the requirements when considering performance, while the qualitative user testing will consider the usability quality attribute.

5.1 Conluedit

As the Conluedit plugin was only used in a development environment, a thorough evaluation of its approach to quality attributes has not been conducted. Evaluating it in this context would be irrelevant for the students as well.

From the perspective of an instructor, creating a macro would look pretty much identical on the course wiki and the development environment. The goal for an instructor, Goal 3, is in regards to the usability of adding/altering an exercise. Due to the implementation constraint IC4, simply implementing a macro that accepts XEMFS would cover the usability aspects.

5.2 Performance of Eclipsky

In order to validate the performance of Eclipsky, the different functionalities of the web editor was compiled in to a list of tasks. The different tasks that are defined as follows:

Ensure project The first time a project is configured

Opening After a project has been set up locally, access it again.

Resource Changing between resources in the web editor

Run existing Run code in the web editor for the first time

Run new Run code after a run configuration is created

Test Perform tests in the current resource

Cont. assist Perform content assist in the current resource

Markers Updating resource and get error/warning markers

To get a data set to evaluate, 20 measurements was performed for each task. The measurements was done by seeing the load time of the web page and comparing the time stamps of WebSocket frames. As discussed in section 4.5.1, editor events will have a delay of $500ms$ before being transmitted. This delay is not part of the measurements, only the time it takes to send the request to Eclipse and get a result back has been recorded.

5.2.1 Test platform

Section 4.2.2 contains requirements for the performance of the implementation. In order to verify that the implementation actually performs as required, the different functionalities was tested and measured. The testing was performed on a desktop PC with the specifications listed in table 5.1.

Component	Description
CPU	Intel i5 2500k, 6MB Cache, 4 cores, 3.30GHz
Memory	8GB DDR3 1333Mhz
Storage	SSD: OCZ Vector 150 256GB
OS	Windows 8.1 64-bit

Table 5.1: Hardware for the testing platform

5.2.2 Results

All the results from the measurements are listed in chapter D, and a summary is presented in table 5.2. With the exception of Rel. std. dev. (relative standard deviation), the values are written in milliseconds. The standard deviation, denoted as σ (sigma), is the amount of variation around the mean value of a data set. The relative standard deviation shows the variation of the mean in the form

of a percentage value. *NB!* All values have been rounded to remove decimals according to the precision of the measurements.

Task\Result	Average	Std. dev.	Rel. std. dev.	Max	Min
Ensure project	632	67	2%	788	525
Opening	594	52	9%	680	492
Run existing	120	10	8%	136	109
Run new	175	35	20%	273	137
Test	210	4	2%	221	203
Cont. assist	24	25	106%	103	5
Markers	9	5	50%	21	4
Resource	4	5	135%	17	0

Table 5.2: Results from measuring different functions in the web editor

5.2.3 Starting project

Since some of the exercises require Eclipse to download some files, the Internet connection on the testing platform are relevant to the measurements. With a download speed of about 30Mbps on the connection, downloading a relevant file takes roughly 50ms. This measurement could be even lower in a server park.

The performance requirement EP1 required a maximum delay of one second for starting a project. Both the tasks *Ensure project* and *Opening* show an average below this requirement, but the max value for starting a new project is relatively close to the maximum. Even though the system had to download files, it seemed to have a low impact on the actual load time of the system, considering the average for *Ensure project* is only 38.45ms greater than *Opening*.

5.2.4 Editor tasks

After the editor has been loaded, a WebSocket connection is established with Eclipse. The measurements of these tasks should be as low as possible to imitate a local IDE, and according to EP2 (basic editor tasks) the requirement is that each task should take less than 250ms. The basic tasks in the editor consists of *Cont. assist*, *Markers* and *Resource*, and all of these have a average measurement below 25ms.

The standard deviation shows a small fluctuation over the average value, but the relative standard deviation is quite high. Considering that these are very low values, it doesn't require much variation in order to result in a high relative standard deviation. Since changing resources takes between *0ms* (min) and *17ms* (max), it is a negligible performance hit. However, the content assists have a significant difference between the lowest and maximum values compared to the error/warning markers and switching resources.

5.2.5 Run/Test performance

Measuring run/test performance is hard to do since the amount of operations included in a run time can vary greatly from exercise to exercise, and even between students. The code that was measured included simple object instantiations and printing to standard output, which requires very little computation power. The measurements are therefore a baseline of what kind of performance the system might provide.

Per the requirement EP3, running code and tests should take less than *500ms*. With the simple code written for the tasks *Run existing*, *Run new* and *Test*, each task used less then the requirement. Running a project for the first time does use some more milliseconds and with a greater standard deviation, but the measurements from *Run existing* and *Test* are the ones that is actually relevant for working on a project.

5.3 Scalability

In order to be able to start up Eclipsky as fast as possible, a headless run configuration was crated. This configuration will start Equinox itself, and it will only include the bundles that Eclipsky need in order to complete its operations. When executed on the test platform with the hardware described in table 5.1, Eclipsky would be available to use within 3 seconds.

This is probably fast enough if a new Eclipsky instance is started up before the other instances has filled up its capacity, and direct any new students to the instance when its ready.

5.4 Usability testing

In order to verify how well the implementation of the web editor satisfies the product quality (usability) and the quality in use (effectiveness) aspects of the quality attributes, a usability test was performed. 5 students, hereby participants, who volunteered was selected to be part of the usability test. The goal of the testing was to see if:

1. The system responded to the expectation of the participants
2. The system offers efficiency and satisfaction to the participants

Before performing the usability tests, a test plan was made (see appendix B). In the test plan, the different research questions and tasks that would be performed in the usability testing was outlined. After the testing with the participants was performed, all of the data was compiled together to different tables and diagrams that offers an insight about how well the product is perceived from a user's perspective. As the participants must remain anonymous, they will be given the pseudonyms P1 to P5.

5.4.1 Task performance

Each of the tasks include a *Description*, *State*, *Success Criteria* and *Benchmark*. The *Description* is what the observer explains to the participants about what they have to try to achieve. *State* is what the starting state of the task must be, and the *Success Criteria* is what the user must end up with before it is completed.

The benchmark describes what is used to rate the usability of some part of the implementation with each participant. In the test plan, a list of research questions have been made based upon Goal 2, which is related to the usability aspect of the students. Each of these questions have been given an equal weight when it comes to measuring the participants' satisfaction level with the system.

The research questions have different spans; Some span through the whole system while some are just for some of the tasks. For each task, the research questions that are related to that task (and not the whole system) are given a score. If a research question is not part of task, the result will display a hyphen instead.

5.4.2 All task scores

Each participant and their score for each task are laid out in the tables 5.3 to 5.7. The task's result have been calculated by testing if that for each participant, the system accomplish the research question related to the task. For each related task, a possible score of either 1 or 0 ("yes" or "no", respectively) have been assigned. The final result score for each task has been calculated by the following equation:

$$\frac{\sum_{i=1}^n t_i}{n} \quad (5.1)$$

where t_i is the value of each research question, and n is the number of research questions related to that task. In the case where a research question wasn't answered in the test, the question will not be included in the equation.

Research Q.\Task	T1	T2	T3
Finished	1	1	1
Test panel	-	1	0
Running vs. testing	1	1	0
Code completion	0	-	-
Response time	1	1	1
Result	75%	100%	50%

Table 5.3: P1's score for the three tasks

Research Q.\Task	T1	T2	T3
Finished	1	1	1
Test panel	-	1	0
Running vs. testing	0	0	0
Code completion	0	-	-
Response time	1	1	1
Result	50%	75%	50%

Table 5.4: P2's score for the three tasks

Research Q.\Task	T1	T2	T3
Finished	1	1	1
Test panel	-	0	1
Running vs. testing	1	1	1
Code completion	0	0	-
Response time	1	1	1
Result	75%	60%	100%

Table 5.5: P3's score for the three tasks

Research Q.\Task	T1	T2	T3
Finished	1	1	1
Test panel	-	1	0
Running vs. testing	1	1	1
Code completion	0	-	-
Response time	1	1	1
Result	75%	100%	75%

Table 5.6: P4's score for the three tasks

Research Q.\Task	T1	T2	T3
Finished	1	1	1
Test panel	-	1	0
Running vs. testing	1	1	1
Code completion	0	-	-
Response time	1	1	1
Result	75%	100%	75%

Table 5.7: P5's score for the three tasks

From looking at the tables 5.3-5.7, some patterns can be found. An obvious observation is that all participants were able to finish their tasks and did not get hindered by the response time of the system. Therefore, the parts of the system that must be further analyzed are related to the test panel, running vs. testing code, and code completion.

5.4.3 Combined results

All of the scores from every participant have been used to individually calculate mean averages for participants and tasks, as shown in table 5.8. In order to see if there is any outliers in the research, a standard deviation for the results of each task was calculated. When used with the average of a data set, denoted as $\langle T \rangle$, the range, $\langle a, b \rangle$, of the mean variation values can be calculated. By using a data set from one of the three tasks, T , the range was calculated from the equation 5.2:

$$[a, b] = \langle T \rangle \pm \sigma \quad (5.2)$$

Participant	Task 1	Task 2	Task 3	Average
P1	75%	100%	50%	75%
P2	50%	75%	50%	58%
P3	75%	60%	100%	78%
P4	75%	100%	75%	83%
P5	75%	60%	100%	78%
Avg.	70%	79%	75%	75%
Std. dev.	11.18%	20.12%	25%	9.6%

Table 5.8: A summary of all participant's result for each task

5.5 Issues

From the results of both the performance evaluation and the usability testing, performance did not occur to be a problem. The issues are therefore related to the other functionalities of the web editor, and will be discussed further in this section.

5.5.1 Running code

In order to find the lowest value in the mean variation for *Task 1*, the standard deviation of this task was subtracted from the mean average. When taking the values from table 5.8, the lowest value becomes 58.82% ($70\% - 11.18\%$). The

only participant below this score is P2, with a score of 50%. The reason for this participant's deviation from the mean should be explored.

When comparing table 5.4 (The result of P2) with the other participants (table 5.3, 5.5, 5.6, and 5.7), the only outlier is the research question related to running vs. testing. The reason why the observer gave P2 a 0 mark on that research question comes from the fact that P2 attempted to run the code by using the *Test*-button instead of *Run* at the first run, as stated in table C.1.

5.5.2 Content assist

As is seen for the notes in table C.1, all participants expected that the editor would support templates. After realising that the editor didn't have support for it, they continued to use the system without trying the code assistant. This meant that most of the participants never actually used the feature, as it was not as functional as the one they were used to in Eclipse.

Only one of the participants tried to use the content assist feature to get a method, but since it only list method names and not the parameters, the participant didn't know what argument to put inside the method.

5.5.3 Individual run time

A recurring problem with the participants was that they didn't intuitively understand that the run time only executed on the active resource. One of the events was that P4 ran the test in resource 1, and understood that an error was caused by an error in resource 2. P4 then changed to resource 2 and fixed the code, executed a test in resource 2 and then went back to resource 1. Inside resource 1, the old test output was still present (with an error). The participant thought that there must still be an error in the code, even though it was fixed. However, the participant also stated that after knowing how it worked it was easy to understand.

The way testing was implemented in JExercise allowed for students to run tests on all resources in the project, and it was therefore expected that the same behaviour was implemented in Eclipsky. P1 suggested that if the test was executed on each individual resource that the resource name should be part of the title for the test panel. If it was possible, P1 would prefer it if all tests related to a project got executed when testing was initiated.

5.5.4 Test result

Participant P1, P2 and P3 wanted the test results to include more information about the error message. P1 also wanted to see a stack trace of the error message to see what line caused the error, as well as a more descriptive name for the

method to understand what it was testing. Another feature request from P1 was the ability to choose explicitly which tests to execute, instead of having all tests for a resource run as in the implementation.

5.5.5 Altering the UI

The participants also expressed some changes that they would like when it came to the User Interface (UI). As stated in section 5.5.3, P1 suggested that the test panel should include the name of the resource as a title. P1 also wanted that the console had the title *Console* above it, as it wasn't immediately understood what it was.

All participants, except P2, would like to have the ability to customize the design. These customizations include the ability to change the text size (for readability) and different color themes in the editor.

5.5.6 Conclusion

Through the test results, many problems with the usability was pointed out. However, at the end of the usability testing, the participants was asked if they would have used the web editor if they were back at the start of the course without the knowledge they now have. Most replied that they would have preferred to use this system at the start of the course and for solving exercises. P4 had even skipped doing exercises at the start of the course, because of the troubles of configuring the development environment. This participant noted that using the web editor was much simpler to use.

Chapter 6

Discussion

Through this research, an implementation of a web editor for students was created. While the research was based on previous work (Rasmussen & Åse, 2014), the goals, constraints and requirements have changed to include new aspects. As well as analyzing the use of the implementation for students, the instructor's usability have been included based on the preliminary study (Vikås, 2014).

In order to cover Goal 1 Student Availability and Goal 3 Instructor Usability, a Confluence plugin was created that can easily be included in the course wiki. From the perspective of an instructor, adding or altering an exercise is done by using XEMFS, a language already used for the creation of exercises in JExercise. This means that the instructor can use the descriptions that was previously written in XEMFS for all exercises when adding web editor to a page.

To satisfy Goal 2 Student Usability, the implementation Eclipsky was created. Eclipsky serves the web editor and handles the IDE-functionality that the users see in the web editor. To see how well Eclipsky satisfied the usability aspect, performance of the system was tested by measuring response time and the effectiveness was tested by performing usability tests with student.

6.1 Usability testing

When performing the usability testing, several usability problems was discovered. Before discussing the problems that was uncovered, it is important to note that the participants is not the target demographic of the system, which is students that have only gotten through the basic programming course.

The participants in the usability testing was all familiar with the use of Eclipse to solve assignments using JExercise, and was used to the features available from that IDE. Students that will be using this system in the future will most likely only have used a regular text editor, where they would have to manually

compile/run code. This means that the expectations of the participants for the system might be higher than the target demographic. Another thing to note is that the sample size was quite small (5 participants), and that the results would have probably been very different with more participants.

6.1.1 Content assist

For the final implementation, the content assist should at least display the parameter and return type of a method, and preferably include templates as well. Templates might not be that important for the students that will be using the final system, but considering that all participants expected the feature to be there it might be a good feature to add. It could help ensure that the students will use the implementation longer before switching over to an IDE.

6.1.2 Run time

There was some confusion for the participants about which resources that was part of a run time. Since the confusion could be affected by the participants' use of JExercise and an IDE, this problem might not occur with the target demographic. As P4 stated, after knowing how it worked it was easy to understand. It would therefore probably be sufficient to introduce the implementation and explain how it works to all students during a lecture.

6.1.3 Testing

Even though all participants managed to use the tests eventually, some had problems with the content of the tests not being informative enough. Having too much information could potentially be overwhelming for the target demographic, but having more understandable test names and descriptions would most likely be beneficial.

6.1.4 UI

Most of the User Interface did not cause any usability problems, but the participants still wanted to customize it to their liking. Including the ability to change the text size inside the editor should probably be implemented, since the implementation could be used by students with reduced eye sight. Having the possibility to change between different themes could be beneficial for the same reason, but also satisfy the students' personal preferences.

6.2 Cloud

Since the final implementation will run in the cloud, the implementation as it is now must include more functionality related to the cloud before it is ready. In a cloud solution, servers goes up/down at different times. If a student worked with a project on one server, the same project files should be available to the other servers if/when the first one goes down. To achieve this, a persistent storage solution must be available so that a server running Eclipsky can both store files there and retrieve them at any time. One solution is to use Git, which Eclipse has built in support for and some of the included bundles already use.

Some Cloud services can offer the possibility to run as an OSGi-bundle, which could help reduce load time as it doesn't need to boot an entire virtual machine. As of now, the hardware that the final implementation will run in is not known, but having a system that has a low start up time is recommended. When used locally, it is at least much more performant compared to the previous work by Rasmussen and Åse (2014).

6.3 Tested exercises

Even though there are over 30 different exercises listed on the course wiki, only a few was tested in the implementation. Considering that EMFS will be used to create the project files, which is the same as in JExercise, it is assumed that they will work in this implementation as well.

Since the implementation runs on the web, creating java window objects is not possible. Exercises that require students to use a window could therefore not be tested with this implementation.

Chapter 7

Conclusion

Configuring a local development environment and using sophisticated software can be a barrier for students learning to program computers. A cloud based web editor opens up the opportunity for students to write code right away, and to do so without the barriers of configuring the local development environment.

After the prototype was created, usability test was conducted with 5 students to check its viability. The participants both solved tasks and discussed the product with the observer, which resulted in qualitative data. Even though the participants met obstacles when working with the product, everyone completed the assigned tasks.

In regards to Goal 1 Student Availability of this research, the Confluedit plugin was created to work with Eclipsky. Though many of the goals have been met, Goal 2 student usability was not complete. While the performance was never a problem during the performance measurements and usability testing, the functionality wasn't always what the participants expected.

7.1 Future research

As have been discussed previously in this chapter, more usability testing should be performed to verify the usability problems of the system. This is also relevant for the features will be fixed or added into the final implementation.

7.1.1 Including EMFS in Confluence

The preliminary study (Vikås, 2014) discussed several ways of introducing the students to the web editor on the course wiki. One of the suggestions was to have the code available when the student opened the page for an exercise, but had to perform an action (e.g. press a button) to initiate the connection with the

Eclipse instance to configure the project. In order to achieve this, a possibility is to include EMFS parser into Confluence and use the functionality of EMFS to create the editor with static resources.

In order to achieve this, a web editor should be rendered on the page from the Confluedit plugin with the resources described in XEMFS. After the student activates the connection with Eclisky, Eclipsky will have to configure the project locally. When Eclipsky is done, the editor could be either be swapped out with an `iframe` that contains the web editor from Eclipsky (as it is implemented now) or just replace the content in the editor with the files in Eclipsky.

7.1.2 Learning Analytics

Learning analytics is about collecting data for analysis and reporting about learners and their contexts. The goal is to get a greater understanding and to optimise learning. In order to analyze how students solve exercises, Sørhus (2015) created a Eclipse plugin to capture file changes, markers and tests that students do in the Eclipse IDE. One of the purposes is to analyze what the students does in each exercise, but a problem arose in trying to distinguish the specific exercises. This is due to students altering or creating their own project and package names for the exercises. By having a cloud based solution, all of the project, package and resource names will be predefined and can not be altered by the students. If the analytical plugin was included in Eclipsky, distinguishing exercises should be trivial, and the final solution would be quite robust.

Bibliography

- Bass, L. (2007). *Software architecture in practice*. Pearson Education India.
- Beel, J. & Gipp, B. (2009). Google scholar's ranking algorithm: an introductory overview. *Proceedings of the 12th International Conference on Scientometrics and Informetrics*. Retrieved from http://www.sciexplore.org/publications/2009-Google_Scholar's_Ranking_Algorithm_-_An_Introductory_Overview_-_preprint.pdf
- Clayberg, E. & Rubel, D. (2008). *Eclipse plug-ins*. Pearson Education.
- Collina, M., Corazza, G. E., & Vanelli-Coralli, A. (2012). Introducing the qest broker: scaling the iot by bridging mqtt and rest. In *Personal indoor and mobile radio communications (pimrc), 2012 IEEE 23rd international symposium on* (pp. 36–41). IEEE.
- Efftinge, S. & Völter, M. (2006). Oaw.xtext: a framework for textual dsls. In *Workshop on modeling symposium at eclipse summit* (Vol. 32, p. 118).
- Johansson, N. & Löfgren, A. (2009). Designing for extensibility: an action research study of maximising extensibility by means of design principles. Retrieved October 6, 2015, from https://gupea.ub.gu.se/bitstream/2077/20561/1/gupea_2077_20561_1.pdf
- Koplowitz, R. (2010). Enterprise social networking 2010 market overview. *Forrester*. Retrieved November 19, 2014, from http://webjam-upload.s3.amazonaws.com/UKIm0Swakr_Enterprise%20Social%20Networking%202010%20Market%20Overview.pdf
- McAffer, J., VanderLei, P., & Archer, S. (2010). *Osgi and equinox: creating highly modular java systems*. Addison-Wesley Professional.
- Oates, B. J. (2006). *Researching information systems and computing*. SAGE Publications Ltd.
- Rasmussen, C. & Åse, D. (2014). A web-based code-editor.
- Rubin, J. & Chisnell, D. (2008). *Handbook of usability testing, second edition: how to plan, design, and conduct effective tests*. Wiley Publishing, Inc.
- Sørhus, S. K. (2015). Applying learning analytics in the course tdt4100 at ntnu.
- Steinberg, D., Budinsky, F., Merks, E., & Paternostro, M. (2008). *Emf: eclipse modeling framework*. Pearson Education.

- Vikås, L. B. (2014). Integrating a web based editor in a course wiki for tdt4100 object-oriented programming at ntnu.
- Whitehead, C. C. (2006). Evaluating web page and web site usability. In *Proceedings of the 44th annual southeast regional conference* (pp. 788–789). ACM.
- Wikipedia. (2015, May). Comparison of javascript-based source code editors. Retrieved May 11, 2015, from http://en.wikipedia.org/wiki/Comparison_of_JavaScript-based_source_code_editors

BIBLIOGRAPHY

Appendices

Appendix A

Workflow

In order to explain the entire flow of the system, figure A.1 contains a BPMN-diagram that explains the work flow for a student who wants to solve an assignment.

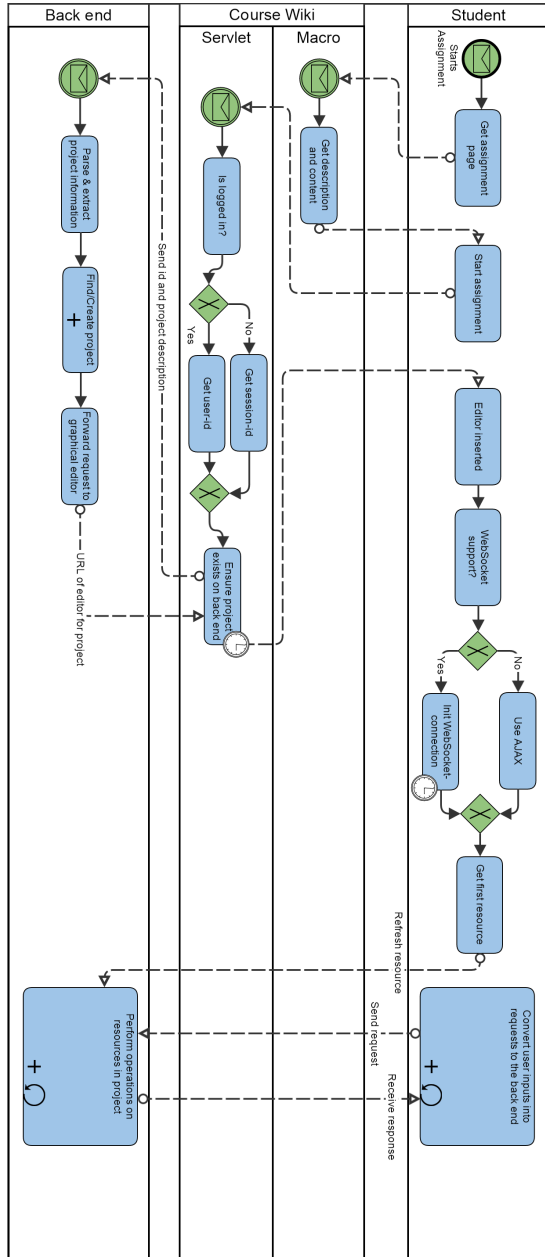


Figure A.1: A overview of the entire implementation

Appendix B

Test plan

This plan outlines the different aspects of performing the usability test. The tests and the test plan was constructed based on the structure described by Rubin and Chisnell (2008).

B.1 Purpose

The purpose of conducting this usability test is to see if the participants perceives the functionality of the artifact in the way it was designed, as well as finding out if the participants would find the artifact useful enough to be preferred over the way assignments are solved now.

B.2 Research Questions

For the usability testing, the only goal that has been taken into consideration is Goal 2, which is related to the student usability of the system (See section 1.1.2 in the thesis for more information). The research goal have been expanded into the following questions:

1. Are any of the participants unable to finish a task?
2. How easily does the participant understand how to use the test-panel when solving assignments?
3. Are the participant able to understand the difference between running the main-method and the tests?
4. How easily does the participant understand how to change between resources?

5. How easily and successfully does the participant figure out how to perform code completion?
6. Does the response time from the server cause frustration for the user?

B.3 Participant Characteristics

Since the implementation is designed for a very specific target group, the participants will all have the same characteristics. All participants are part of the first year for a MSc degree in computer science at NTNU, and have approximately the same skill set. This means that they all have some programming experience, and have recently used the current implementation to solve assignments. These are also the same assignments the new implementation is designed to work with. Since the participants have recent experiences with the current system, they should have a good insight when it comes to comparing the new implementation with the old one.

B.4 Method

For acquiring participants, the supervisor of the research will send out a request to all students currently in the subject and ask if they want to participate in the test with a free dinner as a reward. Considering the timing of the test is close to the students finals, the reward might help encourage them to participate. The expected amount of volunteers is low, so the required group size has been set to five.

Since the numbers of participants is low, a *within-subjects* design has been chosen. This design lets all participants try to accomplish every task, instead of splitting them up to groups where each solve different tasks. A problem with this approach is the potential of transfer of learning, e.g. a participant will find it easier to solve task B after first solving task A. A mitigation of this problem is to have the participants solve the tasks in different order, but since some of the tasks depend on each other, the order of the tasks can't be randomized for each participant. However, this problem is also mitigated by having participants with the same type of prior experience, which is the case in this research.

B.5 Session outline

Each test session will last for 20 minutes, which includes introducing the software, participants performing tasks and post-test questions. During the introduction, the moderator will inform that the participants will be anonymous and that they

will not be tested on their programming proficiency, as well as information on the test procedure.

The test procedure includes the moderator observing the participants while they try to think out loud every step of the way. This is to help the moderator with estimating the benchmarks afterwards. If the participants ask questions, the moderator should not give them direct information on how to solve a task unless needed to go to the next task.

B.6 Test Environment

For the testing procedure, a room at NTNU have been booked for two hours where each participant takes their turn in trying to solve the tasks. When it comes to equipment, the participants will use their own computers to perform the tasks. As the solution is web-based, all modern web browsers should be able to run the site without problem.

B.7 Moderator Role

The moderator will have the role of an observer, and sit in the room with the participants to describe each task they have to solve. As the participants perform their tasks, the moderator will be available to answer questions and offer some assistance. Some follow-up questions might be asked to help with the benchmark if the participants reading aloud is not enough. After the test phase, the moderator will ask the post-test questions. Along the way, the moderator will take notes of the participants responses and behaviour.

B.8 Tasks

In the following tables, the different tasks have been outlined with the different components. The *Description* gives a brief explanation of the task the participant have to achieve, while the *State* explains the state the project has to be in before the task begins. A *Success criteria* explains what the final result should be, while the *Benchmark* outlines what is used to evaluate the participants performance of the test. The benchmark are compiled from the research questions in section B.2.

COMPONENTS	DESCRIPTION
Description	Create a main-method that will test the <code>Card.java</code> 's <code>toString()</code> -method, and run it.
State	Start in <code>Card.java</code> with the default content.
Success criteria	The participants get an output from the <code>Card.java</code> 's <code>toString()</code> -method in the console
Benchmark	Participants understands which button to press to run the code, and not run tests. Participants also identifies that it's the console that will display the output.

Table B.1: Task 1

COMPONENTS	DESCRIPTION
Description	Complete the <code>Card.java</code> -class and have all tests for it pass.
State	Start in <code>Card.java</code> with the default or previous content.
Success criteria	Participants are able to write code and run tests to ensure that the class passes the tests
Benchmark	The participants are able to understand the problem markers and make use of the test panel to see if there are any errors in the current state they are in. They must also be able to conclude when all tests have passed.

Table B.2: Task 2

COMPONENTS	DESCRIPTION
Description	Have all tests for the assignment pass.
State	Start in <code>Card.java</code> with state from task B.2.
Success criteria	All of the tests related to the assignment should pass.
Benchmark	Participants are able to make use of the test panel to both see if there are any errors in the current state they are in, and to conclude that all tests have passed.

Table B.3: Task 3

B.9 Post-test questions

The post-test questions have two purposes; Get (more) information from participants that wasn't written down during the tests and get the participants' true evaluation of the product. This gives the participants the possibility to, amongst other things, explain their view of the product and express their personal preferences.

The first question is an open one about what participants thought about the product, so that they have the possibility to express what they really think without being constrained by the questions. If there was any problems during the test that wasn't (completely) answered, the participants will be asked to explain why that was. Afterwards, the rest of the questions will be related to the questions in section B.2.

During all these questions, the focus will be on problems and difficulties that the participants experienced and not so much on solving those problems. However, if the participant expresses a solution, it will be written down as it could be useful later.

Appendix C

Test result

C.1 Task notes

During the tasks, the observer took notes about what the participants told the observer and the participants interactivity with the editor. This led to a lot of useful notes, but all of them was not related to the current task. The unrelated notes have therefore been placed in the post-question section C.2 under appropriate and relevant questions. For this section, the *notes* for each task/participant only mention the main information used for determining the benchmark.

Task 1: Create a main method

Description Create a main-method that will test the `Card.java`'s `toString()`-method, and run it.

Benchmark Participants understands which button to press to run the code, and not run tests. Participants also identifies that it's the console that will display the output.

Participant	Notes
P1	Tries to use templates, which is not implemented. Ctrl+S when trying to run.
P2	Tries to use templates, which is not implemented. Pressed <code>test</code> instead of "compile and run" the first time
P3	Tried to use templates
P4	Tried to use templates
P5	Tried to use templates and Cmd+S shortcut to run the program

Table C.1: Results from all participants in Task B.1

Task 2: Complete Card.java

Description Complete the `Card.java`-class and have all tests for it pass.

Benchmark The participants are able to understand the problem markers and make use of the test panel to see if there are any errors in the current state they are in. They must also be able to conclude when all tests have passed.

Participant	Notes
P1	Used code completion correctly. Didn't understand the problem as it was stated in the problem marker (mostly skill related).
P2	Acknowledge error markers in the code, but doesn't hover over to check the content. Clicks "compile and run" before clicking "test".
P3	Utilized the information from a test run to solve the task. Used a long time beforehand by switching between resources before realizing where the test panel was.
P4	Read the information in the error marker in order to solve the task, solved the tests on the first try.
P5	Runs tests first and use that information before trying to solve. Doesn't utilize code completions.

Table C.2: Results from all participants in Task B.2

Task 3: Complete Assignment

Description Have all tests for the assignment pass.

Benchmark Participant are able to make use of the test panel to both see if there is any errors in the current state they are in, and to conclude that all tests have passed when they actually are.

Participant	Notes
P1	Confusion about whether to click "compile and run" first before clicking "test", as the "test" button doesn't have a "compile and test" description. Confusion about tests being related to each file, and not seeing all the tests at the same time.
P2	Doesn't realize that the tests are for each individual class. Utilized test message to reason about solving the task, and whether it was concluded. Clicked "compile and run" before "test".
P3	Understood that the test was a per-class basis.
P4	Used the Test panel's drag functionality. Expected that the tests had been performed on every class.
P5	Runs test first, but doesn't check information inside the error. Runs tests in all resources to ensure task is completed.

Table C.3: Results from all participants in Task B.3

C.2 Post-test question

After the test phase, the participants was asked some open questions about the system. All the questions with each participant's answer is listed below.

C.2.1 Was there any problems/difficulties when using the product that prevented you from solving the tasks?

- P1 Tried templates, which doesn't work. The console should have a title so that it's easier to understand what it is and what it does.
- P2 Missing templates and the lack of showing possible parameters for methods.
- P3 Not really. Would prefer if there was templates. Experienced a bug where trying to use code completions, where it removed text due to there being errors in the line above.
- P4 Noticed that templates didn't work. Thought that all tests for all files would run on "test", and not just the current one. Would prefer longer error messages.
- P5 No, but was less efficient due to the lack of templates.

C.2.2 Did you find the product easy to work with? If so, why?

- P1 Completely fine to work with. Very useful with semi-filled classes.
- P2 Likes that a closing-bracket is automatically added when writing a opening-bracket. The lack of available options made it very easy to work it, as the other stuff could cause confusion. Very user friendly in that regards.
- P3 Liked the structure with having both tests and possibility to run the code on the resources.
- P4 It felt intuitive and it was easy to start coding. This was a big contrast to the other system which had resulted in skipping assignments at the start of the year.
- P5 Due to the lack of possibilities, the product felt minimalistic. This made it easier to work with.

C.2.3 Was it easy to understand and use the test panel for solving the tasks?

- P1** Finding out how to run tests was more intuitive than in Eclipse. Would rather have the test details arrow pointing towards the text when it's collapsed and pointing down when open. Would also want a description in a test run that states which line numbers in the code that caused the test to fail. Didn't like that the test panel changed size automatically. When tests succeed, it would be nice to see what the test actually checked. Not intuitive if the code will be compiled when running tests, or if you have to do that manually. Would want the possibility to choose what tests to run. To indicate that the tests are on a per-resource basis, the title of the tests panel should include the resource name. Tests in the current resource that depended on other resources should not appear, as it looks like it's the current resource that has the error (Alternatively run all tests every time on all resources). Preferably, the test details should save the open/closed state when switching between resources. Tests that has an OK status should not have animations on click. The name of each test didn't give enough information about what it was trying to achieve.
- P2** Confused about the meaning of the Test at first, and thought the first error meant that the test was ok (The method was supposed to return an Exception on illegal arguments).
- P3** Didn't quite understand what the test panel and the "test" button did. Understood right away that the tests where on a per-resource basis, but would like more information in the dialog (both on errors and successful tests).
- P4** Yes, due to the cleanliness and availability of the panel. Made it easy to perform tests often, and resizing the panel was useful.
- P5** Yes. The color and symbols was also easy to understand.

C.2.4 What did you think about how the resources was managed in the system?

P1 Liked that the resources was in tabs, but had some confusion about the tests.

P2 Understood right away, easy to use.

P3 Liked the structure with the split into "running code" and "testing code". Good thing that you can't close a resource.

P4 At first, somewhat confusing, since it was expected that tests would be ran on all files. However, it's not really a problem but something you need to be aware of.

P5 Content with the way things are now, as it makes it easier to have an overview of the system.

C.2.5 Is there anything else about the system that you would want to have been handled differently?

P1 Tabs should have a brighter color for readability. The color theme and text size of the editor should be changeable by the user. Code completion should include parenthesis on new objects.

P2 Adding templates and parameter for methods.

P3 Possibility to change text size in the editor.

P4 More customizable design/layout, like themes and text size.

P5 Would prefer a different theme, preferably a bright one.

C.2.6 If you imagine yourself as a new student in this course, would you consider using this product instead of the current implementation?

P1 Not for own projects, but would use it at the start of the course to solve assignments. Liked that it removes the need for setting up either Eclipse, Jextest or the project resource files manually. Would personally transfer over to an IDE after a while, as that is what you would have to use when getting a real job.

- P2** Would use this product for the assignments, and Eclipse for other things like own projects.
- P3** See the possibility of having this as a platform for having programming exams, but would rather use Eclipse as it gives freedom to structure own projects.
- P4** Due to the experience with the current implementation, yes. In bigger systems and for normal use, Eclipse would be preferred. For the assignments, this solution would be much simpler to use.
- P5** Yes. It would make it much easier to start with an assignment, and it's much harder to make any mistakes with this system since there is only a limited amount of modules that is accessible.

Appendix D

Performance measurements

This chapter contains all the raw data from performing performance measurements on the web editor from Eclipsky.

Meassure\Task	Ensure Project	Open	Resource
1	697	522	17
2	604	628	7
3	563	557	1
4	628	582	1
5	788	585	3
6	648	658	1
7	620	666	1
8	661	631	0
9	684	558	7
10	529	574	2
11	619	589	11
12	676	492	1
13	641	656	1
14	667	542	1
15	629	527	2
16	527	634	1
17	725	596	1
18	611	612	1
19	525	581	2
20	597	680	17

Table D.1: Performance of ensuring a project, opening up a ensured project and changing resources

Measure\Task	Run existing	Run new	Testing
1	110	137	221
2	111	168	213
3	113	166	211
4	111	161	209
5	113	168	209
6	109	167	209
7	112	165	208
8	110	163	209
9	110	177	210
10	113	164	215
11	136	175	210
12	117	156	210
13	118	158	209
14	130	181	209
15	133	163	209
16	127	165	212
17	136	160	207
18	129	153	210
19	130	273	207
20	128	271	203

Table D.2: Performance of running a project with and without run configuration, and testing

Meassure\Task	Content assist	Markers
1	103	7
2	12	21
3	12	5
4	7	11
5	8	19
6	78	10
7	8	4
8	13	8
9	48	4
10	21	12
11	19	11
12	22	8
13	23	13
14	15	11
15	25	4
16	20	4
17	19	7
18	7	8
19	5	12
20	8	8

Table D.3: Performance of doing basic tasks in editor

Appendix E

Source Code

The implementation that was created in this research is located in public repositories on Github. Since the base implementation of Eclipsky was created by the supervisor, a separate forked repository was created. Confluedit, however, resides in a single repository.

Eclipsky - Base <https://www.github.com/hallvard/eclipsky>

Eclipsky - Fork <https://www.github.com/bvx89/eclipsky>

Confluedit <https://www.github.com/bvx89/confluedit>