NTNU – Trondheim
Norwegian University of
Science and Technology

Applying Learning Analytics in the course TDT4100 at NTNU

Stein Kjetil Sørhus

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

Abstract

In the Object Oriented Programming course TD4100 at NTNU there is a need to get an improved overview of the learning challenges of the students. A better understanding of how the students work, and how they handle the weekly programming exercises, could enable further improvements of both the exercises and the course.

A design science approach was used to implement a Learning Analytics system to be used in TDT4100 and similar courses. The aim was to create a visualisation of the students' progress through exercises, by collecting snapshots of the source code being written. Several metrics were extracted from the snapshots to enable an expression for progress to be created. An experiment was conducted with voluntary students in TDT4100 to assess the proposed progress metric.

The results indicated that the progress of students when completing exercises can be measured. The results also indicate that visualising the progress metric over time enables identification of areas of breakdowns. Inspecting the students' source code at these areas was shown to aid in identifying the challenges of the students.

The implemented system worked according to specifications and enabled research into how students progressed through exercises. The system was flexible and several extensions were successfully made through iterations of development and evaluation. The results of the thesis is seen as a starting point for further research into the challenges of the students in TDT4100.

Sammendrag

I faget "Objektorientert programmering med Java"(TD4100) ved NTNU er det et behov for å få en bedre oversikt over læringsutfordringer til studentene. En bedre forståelse av hvordan elevene arbeider, og hvordan de håndterer de ukentlige programmeringsøvingene, kan muliggjøre forbedringer av både øvelser og faget.

Design science ble brukt som metode for å implementere et Learning Analytics system som kan brukes i TDT4100 og tilsvarende fag. Målet var å lage en visualisering av studentenes progresjon gjennom programmeringsoppgaver, ved å lagre flere versjoner av kildekoden som ble skrevet. Flere parametrer ble innhentet fra kildekoden for å muliggjøre sammensettning av et uttrykk for progresjon. Et eksperiment ble gjennomført med frivillige studenter i TDT4100, for å evaluere det foreslåtte uttrykket.

Resultatene indikerte at progresjonen til studenter som jobber med øvinger kan måles. Resultatene tyder også på at visualisering av progresjonsmålet over tid, gjør det mulig å identifisere områder med "breakdowns". Ved å inspisere studentenes kildekode i disse områdene er det mulig å identifisere studentenes læringsutfordringer.

Systemet som ble implementert fungerte i henhold til spesifikasjonene og muliggjorde forskning på elevenes progresjon i øvinger. Systemet var fleksibelt og flere utvidelser ble implementert gjennom flere iterasjoner av utvikling og evaluering. Resultatene fra avhandlingen regnes som et utgangspunkt for videre forskning på læringsutfordringene til elevene i TDT4100.

Acronyms

API Application Program Interface. 24, 25

HTTP Hypertext Transfer Protocol. 23, 25

IDE Integrated Development Environment. 3, 7, 8, 20, 21, 38, 56

IPC Inter-Process Communication. 23, 24

JSON JavaScript Object Notation. 25

LMS Learning Management System. 10, 20

MQTT MQ Telemetry Transport. 23

REST Representational State Transfer. 25

VM Virtual Machine. 22

Contents

List of Figures

Chapter 1

Introduction

1.1 Problem Identification & Motivation

In the Object Oriented Programming course TD4100 at NTNU there is a need to get an improved overview of the knowledge of the students, and how they progress through the course. There are over 450 enrolled students each semester, and due to varying level of previous programming experience among the students, it is difficult to get an understanding of their expectation and needs of the course. This makes it difficult to assess the perceived difficulty of both the course and the assignments. A further understanding of how the students work and how they handle the weekly exercises could enable further improvements of both the exercises and the course.

A pre-study to this thesis was completed to determine ways of improving the educational environment for the students who struggle the most in TDT4100 [1]. The aim was to identify ways to extend their programming tool to provide guidance and prevent students from getting stuck. The pre-study was a literature review, that examined research into common errors made by novice programmers, editors in use and their implemented functionality to aid programmers, and existing tools in use to aid novices.

The pre-study concluded that more research had to be done to determine the specific challenges of the students in the course, as the evolution of the modern Integrated Development Environment (IDE) had implemented several of the findings in early research. The pre-study suggested that collecting data regarding the specific learning activities of the students could enable further understanding about their specific challenges.

The students in the course are required to complete weekly programming assignments, which provide an opportunity to study the process of the students while programming. Due to the large number of students in the course, an automatic analysis is required to

monitor and examine the students work flow.

The aim of this project is to follow up on the pre-study and use Learning Analytics to examine if it is possible to reason about the students progress through exercises and identify if and when they are stuck. This knowledge could enable further research into why they are getting stuck, and will give the lecturer a better understanding of both the level of the students and what they find most challenging.

1.2 Problem definition

The main objective of the project is to learn more about how the students in TDT4100 work through assignments, and what they find challenging. The pre-study of the thesis concluded that analysis of the evolution of the source code the students write when completing assignments, could be a way to approach this research [1].

Therefore the aim of the project is to collect data about the activities of the students while completing assignments, and create a flexible platform that can be used to analyse this information. The aim is for the platform to aid in the analysis of the students in the course TDT4100 specifically, but also enable extension of the platform for use in other courses and as an overview dashboard to aid lecturers.

1.3 Thesis Structure

This section provides a reading guide for the reader to enable navigation to relevant parts of the document.

The thesis begins with problem identification and concretisation, and follows on to solution specification and architecture of the software created. Then the final software is described, as well as the process of extending the software and performing the research. After this the results and discussion of the research is presented, as well as an evaluation of the software solution. The final chapter contains a conclusion and suggestions for future work.

Even though a design science approach was used throughout this project, the results and the discussion of the experiment is presented using a traditional scientific empirical model. This been done to highlight the results of the research in a traditional and to the point manner. The process of arriving at the results and discussion regarding the process used is found in chapter 5.

Chapter 1 Introduction.

Chapter 2 Detailed overview of the background theory, with an emphasis on Learning Analytics.

Chapter 3 Narrows down and specifies the objectives of the project and the software developed.

Chapter 4 Technical overview of the system, including overview of the software architecture and explanation of the different parts of the system.

Chapter 5 Demonstrates the use of the application by detailing how the experiment with students in TDT4100 was performed, and the design science approach of developing the analysis software.

Chapter 6 Results of the experiment.

Chapter 7 Discussion of the results of the experiment and evaluation of the software and research in relation to the objectives and research questions.

Chapter 8 Conclusion and suggestions for future work.

Chapter 2

Literature Review

The project started with a literature review to examine if there has been any research on similar problems, and to examine the results that has been found.

2.1 Challenges of Novice Programmers

This project was preceded by a literature review[1] intended to identify how the Eclipse IDE could be extended to aid novice programmers. The motivation for this study was to improve the learning environment for the students struggling the most in TDT4100.

The study examined topics such as common errors made, the challenges, and the tools used by novice programmers. The study identified many common errors and challenges encountered by novices in general. However, due to the development of modern IDEs and the lack of recent and specific research on the particular programming language and IDE used by the students, the relevance of the data was questionable.

Even though several suggestions were made to improve the aid provided to students by Eclipse, the data was not adequately reliable. This lead to a conclusion that more research had to be conducted to specifically identify the challenges of the students in TD4100.

A suggestion was made to automatically monitor how the students work through assignments by continuous logging of the written source code. The project also developed a proof of concept logging plugin to demonstrate how it might be done.

This study serves as the foundation for this project. The aim of this project is to use the data collected through such a plugin to perform analysis that can aid in improving the educational environment for the students in TDT4100.

2.2 Learning Analytics

Learning Analytics is a relatively new field of study with roots in several fields including Educational Data Mining, Business Intelligence and Psychology [2]. Learning Analyics uses learner produced data as a foundation for further inference about the situation of learners with the goal of improving the learner environment. The aim is to create analysis models and discover information from the data collected from learner activities that can give further insight into the learners situation [2].

Educational Data Mining has a significant relation to Learning Analytics [3]. Even though there are many overlaps, the most prominent difference is the overall aim of the research. Educational Data Mining is more focused on creating inferences through automatic systems with no human intervention. The aim is often to decrease the workload of the teacher through automatic assessment of students, or outcome prediction.

Learning Analytics on the other hand puts the learner in focus. The aim is to inform and empower learners and educators to improve learning. In previous research this has been done through creating tools that contribute to learning and give insight into the learning environment and state of the students [3].

Since Learning Analytics has roots in several fields, there are many different approaches used in the research. This includes automatic analysis such as machine learning [4], statistics, as well as manual approaches where an abstraction of the data is inspected directly by the researcher[5]. The current tools resulting from Learning Analytics include early warning systems, automatic tutors and recommender systems [2].

The field of Learning Analytics is still developing. Researchers are currently exploring the potential of Learning Analytics for the future, and the tools currently made are only of the first generation. This means that there are minimal experience and frameworks to build from, however it also means that there are many discoveries to be made. Siemens [2] argues that Learning Analytics has a potential for a significant impact in the education systems today, and could transform the system on all levels.

2.3 Code Evolution

Many researchers have looked into ways of gathering code snapshots or inspecting them over time. This includes plugins for various IDEs made for collecting snapshots, and tools created for the specific purpose of analysing the snapshots. However, few have ventured into automatic analysis and metric extraction of the data.

Marmoset[6] was a system developed for automatic submission and testing of student assignments. The data collection was added as a plugin to the Eclipse IDE, and the system also enabled optional snapshots of the students code whenever they saved a file.

Spacco et al. [7] used Marmoset to collect data from 73 students to examine how students learn to develop software. This research however was focused on the correlation between errors reported and error exceptions encountered. The aim of the study was to find new ways of static code analysis to identify errors.

CodeBrowser is a tool for comparing snapshots of code over time[8]. It was created to inspect the work flow of students as they work through assignments. The tool includes a timeline that is used to step through each snapshot of the code.

The usage of CodeBrowser requires manual interaction and does not include any automatic analysis. The purpose is to give instructors and researchers a tool for stepping through the programming process of learners, and manually reason about the work flow.

Analysing snapshots of code has also been done to examine the edit-compile cycle of students [9]. A plugin to jBlue was developed together with a snapshot browser to enable research into how students respond to syntactical errors in the code. Part of the aim of this research was to identify if there was a correlation between the final grade of the student, and how the student dealt with syntactical errors.

There is also some research into visualising metrics from snapshots. SnapViz [10] was created as an online tool to view snapshots metric in the browser. It was made as a proof of concept to show what is possible and only visualised the time of compilation and whether the build failed for several snapshots.

Blikstein [5] used learning analytics to assess the behaviour of 9 students in open-ended programming tasks. The students' source code was collected in snapshots while they performed an open ended assignment of creating a program to model a scientific phenomenon of their own choice. Some metrics were extracted from the snapshots, including size of the code, code compilations as well as error messages and types.

The metrics from the snapshots were then plotted over time to allow the researchers to identify areas of the curve that deviated from the norm. The inspections resulted in the researchers suggesting three coding profiles: "copy and pasters", "mixed mode", and "self-sufficient". These profiles take into account how the students work by looking at how often code is copy pasted in from other places, and how much time is seemingly spent thinking of a solution rather than looking it up.

2.4 Research Grounding

The project shares both the aim and the means of Learning Analytics. The aim is to gain a further understanding of how students work, and the means is through collecting data about their coding activities. The aim is not to make an automatic assessment of the students performance, but to use human interference to analyse the data and find information about the challenges of the students. This information can be used by the course instructors to

gain a better understanding of the needs of the students.

Siemens and Baker [3] advocates usage of diverse data in Learning Analytics. Most of the current tools developed through Learning Analytics research rely solely on data collected from Learning Management System (LMS)s. The reason for this is that it is one of the few platforms where the activity of learners are already monitored and collected. This data usually only measures student participation and engagement in the course, and does not directly measure actions performed in the learning environment.

Programming education provides a unique ability to collect data about the specific learning activity of students. This is possible since all work to complete the mandatory programming exercises is being done on the computer. This provides a more detailed and "closer to the source" view on the learning experience, and has the potential to give a more specific and accurate depiction of the learners process, compared to the data collected from LMSs.

Since Learning Analytics is still a new field, the research conducted thus far are of variable relevance for this project. Many studies have been conducted with a focus on collecting data regarding students activity while programming, however the aims of the research are differing. Few studies are directly related to examining the work flow of students, however the methods used in similar studies may still be relevant.

Chapter 3

Objectives of a Solution

Even though few studies are directly related to using Learning Analytics to examine the work flow of novice programmers , the studies identified has provided a grounding for further research. Both Spacco et al. [7] and Blikstein [5] collected snapshot of students code over time and developed ways to gain significant insight by analysing certain metrics in the code.

The approach used by Blikstein [5] was a significant inspiration for this project. The results show that extracting relevant metrics and plotting them over time can give considerable insights into how students work.

The study only looked at 9 students, which gave the researchers the possibility to examine each of the plots carefully. In this project however, there are more than 450 students, each completing many exercises through the semester. Therefore the exact same method cannot be used as it is not feasible to individually examine the graphs of all students.

The optimal solution would be to find a metric that would give similar insights into how students work, without the need for time consuming examination of each plot. This metric should allow comparisons of students working through the same exercise, to easily identify patterns that indicate the students who are struggling.

The other difference to the study is the aim of the research. Since the aim of the project is to identify the challenges of the students, the metric would need to provide information regarding when students are having issues progressing in the exercise. This was not part of the objective of the study, and only plotting the size of the code over time might not provide the necessary information.

Therefore the goal of the research is to find a visualisation for the progress of students through exercises. Since progress is a general measure that is independent of the type of exercise, visualising progress would allow easy comparison of students, as well as ex-

ercises. The progress metric would also provide identification of challenging parts of exercises, where the progress over time would be low relative to the other parts.

3.1 Research Questions

Due to the broad scope of the project, it was necessary to narrow down the research goal of the project to a manageable level. Through several iterations the research goals were condensed to only a few research questions that encompass the essence of the wanted information about the work of the students.

Q1: How can data collected from students working through programming exercises be used to reason about their *progress* through the exercises?

As detailed earlier, the progress of the students was found to be a metric that could be used to infer information about how the students work. In this project a students progress through an exercise is meant to show how close the student is to completing the exercise at any given time.

The assumption is that progress over time is linear through an exercise in ideal conditions. This implies that for each unit of time spent working on the exercise, the student will progress an equal amount through the task. Due to the context of the project, it is not expected that students will have a linear progress curve. As they are in a learning environment it is natural for them to spend more time on some parts of exercises while learning, this deviation is going to be the most interesting aspect of the progress.

This metric will give insight into differences among students, and the relative difficulty of exercises depending on the slope of the progress curve. Exercises where a significant amount of students have a non-linear progress curve could be specifically challenging, and warrant further inspection.

Q2: How can data collected from students working through programming exercises be used to identify *breakdowns*?

A breakdown is a point in an exercise where a student is stuck or spends a long time progressing relative to the rest of the task. The reasons for breakdowns do not include a lack of effort, rather it is a lack of knowledge of the subject, a misunderstanding, or similar. The goal is to identify situations where students need help, so that these instances can be inspected further. This could include individual breakdowns, or instances where groups of students are seen to have similar breakdowns.

Knowledge of these breakdowns can enable help to be given both individually and on a higher level through course adaptation.

3.2 Methodology

Design Science was used as the methodology in the design and development of the system and the research. Design Science is distinguished from natural science in the goal of the research. Natural science is concerned with understanding and explaining reality, while design science is concerned with the creation of artifacts that serve specified goals[11].

March and Smith [11] argues that there are two essential activities in design science; build and evaluate. These activities reflect the creation of the artifact, and an evaluation of the artifact to ensure it meet the intended goals.

The research model used in this project is a design science research model developed specifically for information systems research [12]. It was developed to have a standardized conceptual model that Information Systems researchers can use when performing design science. This also benefits the readers as it creates a mental model that can act as a template defining what to expect when reviewing the literature.

The model includes 6 steps:

- **Problem identification and motivation**
  Identifying the problem and justifying the relevance of the solution
- **Objectives of a solution**
  Identifying the objectives of a solution to the problem.
- **Design and development**
  Designing the artifact, including specifications and architecture, as well as the creation of the artifact.
- **Demonstration**
  Demonstration of how the artifact solves the problem.
- **Evaluation**
  Evaluate to which degree the artifact solves the problem according to the objectives.
- **Communication**
  Communicate the value of the problem and the resulting artifact or research.

These steps are followed both in the project and this written thesis. The thesis includes all steps except communication, which is omitted as it is manifested as the thesis itself.

Design science is an iterative process, and often iterates through design and development, demonstration and evaluation. The project went through many iterations of these steps, especially in the process of developing the analytics application. This stage of the development required consistent evaluation of the application according to the objectives and research questions to ensure advances in the research.

3.3 Hypothesis Development

3.3.1 Progress

Progress as a metric

The progress of an activity is an abstract metric. It portrays something about the current state of the activity in relation to the goal. An example of a simple progress metric is the distance remaining for a runner in a race. At any given time, the position of the runner along the track can be used to find the progress of the runner by comparing it to the position of the goal.

By plotting the progress over time, information about both the runner and the track can be identified. The slope of the progress curve in relation to other runners, will identify the relative skill of the runner. A great runner, will run faster than the others, and will progress more rapidly resulting in a steeper progress curve.

If some parts of the track are more challenging than other parts, this can also be seen on the progress curve as a decrease in the rate of change of the progress. If a hill is located along the track, the runners will most likely have a lower average speed when running up the hill, resulting in a lower average progress over time. A hill can then be identified when several runners are showing the same signs of a lower rate of change at the same progress position in the race.

The progress curve can also be used to identify breakdowns, or portions of a race where a runner is seen to stagnate in progress. This can happen if a runner is exhausted and stops running to take a break. The breakdown will then manifest on the progress curve as a portion with a low increase in progress over time.

Identifying progress in programming exercises

Finding the current progress of writing a program is difficult. A program is by definition complete when it fulfills the required functionality. The problem is that there are an infinite number of programs that can be created to fulfill the exact same specifications. This means that there is no given sequence of states to reach the goal when writing a program, as there is when completing a marathon.

Each marathon has a specified track, and deviations from the track are not allowed. This makes it easy to compare the state of the runners to the goal. However, programming can be seen as only giving each runner a starting position and a goal position, and letting them choose their own track. Knowing the position of the runner at any point is not useful, as it is not possible to know how the runner will progress further.

However, if the program specification is accompanied with several tests that ensure the partial working of the program, it makes it possible to estimate partial progress. Each test can be seen as a milestone that has to be reached. If there is a sufficient amount of milestones, the progress can be estimated by the number of remaining milestones.

Code metrics to identify progress

Size of the code

The most crude way of determining the progress of a program is through the size of the code. This metric provides information about the effort made by the student to complete a task, and also that the student has an intention of progress.

However, due to the nature of programming, the size of the source code is a symptom rather than a cause of a working program. This means that the measure cannot accurately predict the current progress of the program, in the same way that the number of pages written cannot determine the current progress of writing a book.

The other issue is that people are different. Depending on how the problem is solved, the size of the code will change. A simple but elegant solution may require a small amount of code, but often the same problem can be solved quicker in a less elegant way with more code. Some people also prefer to have longer variable names, or use more methods. All these issues makes the size of the code an uncertain measure of progress.

However, the common factor in all these issues is that an increase in size of code usually means an increase in progress. The act of writing code is a proof that the student is trying to progress in the exercise. The problem is just that it cannot accurately predict the progress relative to the finished program.

When refactoring code this assumption does not hold. The programmer may simplify parts of the code, reducing the size while keeping the functionality. This means that progress can increase while the size of the code decrease.

However, the assumption is that refactoring is rare in these small student exercises as there is relatively small amounts of code involved. Therefore the hypothesis is that the size of the code is an accurate metric for gauging relative increase in progress and intention of progress, but it does not determine the overall level of progress through an exercise relative to the final progress measure.

Failed tests

There is no way of determining if a program is complete without knowing the function the program is supposed to perform. Therefore a finished program must fulfill all required specifications. This leads to a measure of progress depending on the number of requirements that are fulfilled at any given time.

Several tests are provided for the students to check whether their solution satisfies the

specified requirements of the exercise. The assumption is that if all tests are passed, the program is working according to specifications and the exercise must be finished. This gives a measure of progress of the exercise based on the number of tests that are still left to pass. Once a student starts an exercise, the number of tests left to pass will be the amount of tests accompanying the exercise, and when the student finished the exercise, all the tests will be passed.

The tests do not always account for all the requirements, however most exercises include several tests, and the aim of the tests is to provide near complete coverage of the code. This is possible as the students are not graded on the correctness of the exercises, and are supposed to use the tests to verify that their solution is satisfactory. This leads to the assumption that the tests can be used as milestones most exercises.

An issue with using the number failed tests as a metric for progress, is that it is a discrete measure with few states. This is due to the relatively low number of tests for each exercise. As the goal is for the optimal progress curve to be relatively linear over time, this metric is not sufficient. Therefore the hypothesis is that failed tests over time will track the overall progress of the student through the exercise, but will not give the sufficient detail of the progress over time required to reason about the challenges of the student.

Hypothesis
The resulting hypothesis is that progress through an exercise can be identified by a combination of the number of failed tests and the size of the code written. The failed tests will provide the milestones towards to goal, while the size of the code will provide detail in progress between the milestones.

3.3.2 Breakdowns

Breakdowns are situations where a student needs help to progress further. This means that breakdowns, as defined in this project, is a significant lack of progress over time. This implies that on a graph showing progress over time, a breakdown should be identifiable as a part of the curve where the rate of change of progress is significantly lower than other parts.

Error markers
To aid in the validation of identified breakdowns, the error markers of the code should be collected. This information would help in understanding why the student is struggling, and can provide further research into the common errors and challenges that are effecting the students.

Hypothesis
Breakdowns can be identified on a progress over time graph as a significantly reduced rate of change in progress.

3.4 Objectives of the Artifact

Given the objectives of the research, it is possible to determine more specific objectives of the required software application. These objectives specifies what is expected of a working solution and will be used to evaluate the final artifact.

The aim of the system is to provide a flexible platform for Learning Analytics that can be used in courses like TDT4100. To evaluate and demonstrate the system, the research into identifying the progress of students will be performed using the system. The goal is for the system to allow extensions such as dashboards for course instructors to use to get an overview of the students in the course, and to find other types of measures that can be relevant for Learning Analytics in programming courses.

3.4.1 Functional Objectives

Collecting Data
The system must be able to log the data required to reason about the metrics specified in section 3.3.1. Since there is an experimentation element in the project, the aim is to have a flexible data collection. This means that it is important to keep the original data intact, so that new metrics can be gathered if required by later analysis. The data collected must be as elaborate as possible and should not be processed or abstracted before storage.

Storage of Data
The data must be correctly transfered and stored in a central location where it can be further analyzed. The data must be stored with no modifications or abstractions to prevent loss of potentially important information.

Analysis
The system must process the collected data and extract the identified metrics. It must provide an interface for visualising the metrics in relation to each other.

Experimentation and Exploration
The artifact must allow experimenting with relations of metrics to allow new visualisations based on different combinations. It must also provide interfaces for further experimenting with the data to compare and contrast different visualisations with different levels of detail in the data set.

3.4.2 Non-functional Objectives

Scalability
An important aspect of the solution is the scalability of the system. As part of the project aim is to determine the feasibility of the program in use in a university course, it must be

able to accommodate the requirements imposed by this. This means it must be written to handle large and unspecified amounts of users completing many exercises throughout the semester.

Flexibility
The analysis process and the experimentation tools must be flexible to allow for rapid iterations of growing the application to allow new metrics to be added, and new ways of exploring and analyzing the data set. The application should also allow extensions that can enable the system to be used in other courses, with potentially different programming languages and tools.

Performance
The analysis of the whole data set must be in a relatively timely manner in relation to the size of the data set. There is no real-time requirement for the data processing due to the performance required to keeping a flexible experimentation possible. However the architecture should facilitate expansion to enable continuous monitoring and reporting for future use cases.

Chapter 4

Design and Development

This chapter presents the design of the application, including the major architectural decisions and the workings of the end product. The usage of design science as the methodology requires several iterations between design, development,demonstration and evaluation. In this project the methodology has been key to ensure rigour both in the research and the development of the system.

The iterations in the development where major design elements were changed will be discussed in this chapter, to give an impression of the process and explain reasoning behind trade-offs and major decisions.

4.1 Context - details of the course

The aim of the system is to be a platform for learning analytics that can be used in several courses. However, the current usage for this project will be in the course TDT4100 at NTNU, and therefore the system will be customized for this context while remaining as flexible as possible to allow for integration in other courses.

The course is an introduction to object oriented programming and is the part of the second semester of the first year of the computer engineering degree at NTNU. The aim of the course is to teach students about object oriented programming principles and to teach how to program in Java.

The students in the course are required to complete a given number of assignments to be eligible for examination. The assignments consist of multiple exercises of which a given number has to be completed. The student is usually free to choose which exercises to complete among several possibilities.

Some of the exercises are open-ended and require the student to create custom classes and logic, however most of the exercises follow a specific structure. The exercise specifies several classes to be created, and also specifies which methods should be implemented and their expected input and output. The student is then expected to implement the classes according to the specifications.

The exercises are accompanied by several tests that the student can use to evaluate if their program is working according to the specifications in the exercise. The number of tests, and the code coverage the tests provide vary between exercises. It is also optional for the students to take advantage of the tests, as long as the programs are working as expected when they hand in the assignments. The students deliver the finished assignments on line through an LMS.

In this project the exercises that the students complete will be used as the basis for the analysis. Since the exercises are mandatory, the assumption is that all students will complete the exercises, given enough time. This means that all students should end up with reaching the progress goal. Since the exercises are given with strict specifications, an assumption of the progress through the exercises can be made by examining the current number of failed tests.

4.2 Specifications

Due to the design science approach used in the process of development, the full and detailed specifications of the system are not known in advance. However the aim and some of the required functionality is known and will serve as a starting point for further evaluation and development.

4.2.1 Data Collection

To enable sufficient collection of all required data, the implementation must be tailored for the context of the participants. The students use the Eclipse IDE as a general tool for Java programming. This means that they use it both to complete exercises, as well as any other Java programming they might do. The students choose when to start the assignment, how to structure their work environment and how to finish the exercises. The files are only classified as an assignment at the time of delivery on the LMS. This makes specific data collection regarding the exercises difficult, as it is difficult to determine which files are related to which exercise.

This presents a challenge for the system. The aim of the system is to require as little work as possible from the students, to lower the barrier for participation in the voluntary experiment. Therefore manual classification of exercise files is unwanted as it will require extra work from the participant. Therefore the only way to classify exercises is by identifying

the contents of the .java files, and relate the content to the exercises.

Since the exercises generally have a mandatory structure, where certain classes are required, the contents of files can be used as an identification for exercises. The tests run in Eclipse will also contain a reference to the class files being tested, allowing the test runs to be collected as well.

To enable data collection of test runs and markers from the work, the utility must be implemented as a plugin to the IDE in use. The tests are run by the students through the IDE, which also provides them with the results of the tests.

The Eclipse IDE integrates markers into the gutter next to the code to convey information regarding errors and warnings. These markers stem from the results of the compilation and static analysis of the code, and show where problems are found and includes a message to describe the problem. Collecting these markers can enable reasoning about errors in the code, in relation to the challenges and breakdowns of students.

This means that the data collection plugin must be able to collect:

- File content of all .java files edited, for later classification
- Test results from the tests that were run for relevant files
- All program markers related to the currently edited file

The data must be collected on an interval that gives the sufficient level of detail needed for the analysis. There is a trade-off between the detail of the collected data, and the performance and processing required. Collecting each key-stroke gives incredible detail into how the students work, but results in significant processing overhead. For the purposes of this project, the trade-off favors a lower level of detail due to the time span of each assignment which is in the order of hours.

To get a necessary level of detail with an acceptable level of processing, the data should be collected each time the current file is changed on disk, i.e when the student saves. This allows an assumption to be made that reduces the complexity of state processing; the student assumes the program is in a sufficient working order to warrant a save. Meaning that they are not in the middle of a small change.

The Eclipse IDE will automatically perform a build and generate program markers that can be collected with the state every time a save occurs. This means that the data collection is simplified as program marker will always be up to date.

4.3 Data Analysis

To make the analysis interface as flexible as possible for future use cases, it will be implemented with a browser based front-end. This will allow the application to be used on any platform, at any time. It also makes it easier to create extensions that depend on other editors, as the front-end and server for analysis will be independent.

The data analysis must allow extraction of key metrics from the snapshots collected, and provide the possibility to change the extracted metrics at any time without loss of data. The metric that should be extracted include:

- Size of the code
- Number of failed tests
- Time spent
- Associated markers

The application must provide an interface that can enable researchers to create visualisations of expressions based on the collected metrics. The software must also be easily modifiable and extendible to allow extensions to be made based on the requirements of the research.

4.4 Contextual Constraints

4.4.1 Time limit

Due to the time limit of 20 weeks for finishing the master thesis, there were some consequences affecting the project. The length of the project was short, so the development and progress had to be rapid.

The course TDT4100 began at the same time the master project was started, this made it crucial to start the data collection as early as possible to collect as much data as possible in the time frame available.

4.4.2 Technical

The application had to run on a Virtual Machine (VM) provided by the technical administration at NTNU. This put some limitations on the performance of the application as the VM was given limited computational power.

4.5 Application Architecture

This section will detail the major architectural patterns used in the application, and give overviews of the system in different architectural contexts.

The system uses the microservices architectural pattern to divide the application into several stand-alone services to increase the flexibility of the system, and enable horizontal scalability to accommodate increasing number of students.

The data collected is stored using the event sourcing pattern, to retain a full representation of all events entered in the system. This makes it easy to change the way metrics are extracted and even allow new metrics to be extracted from all collected data at a later time.

More details regarding the architecture of the implementation can be found in appendix A.

4.5.1 Microservices

Microservice architecture is a relatively new architectural pattern that specifies a way to structure large applications as a collection of smaller services [13]. Instead of creating one single monolithic application, each service is itself a small application and by communicating together they form the complete system. The purpose is for one service to be a stand alone entity where the responsible team can freely choose the software and hardware solutions that fits the service, without being constrained by the rest of the application.

Microservice architecture is often compared to componentization, where an application is divided into separate components with limited responsibility. The practical difference between these architectures, is the means of communication between the modules. In a monolithic component based application, the difference components communicate in an inter-process manner using language specific features. In a microservice architecture, the services communicate using Inter-Process Communication (IPC), typically using messaging protocols such as Hypertext Transfer Protocol (HTTP) or MQ Telemetry Transport (MQTT)[14].

The convention is to have smart endpoints and dumb pipes. This means that communication between services should be easy and not rely on a complex message bus, and the services themselves should handle the different types of data structures or requests.

The benefit of microservices is the increased freedom of each service to independently choose architecture and run-time environments. This makes the applications more flexible, and allows for more tailored services. This also makes the application more flexible as services can be swapped out or re-written without changing the other parts of the system.

Scalability is another great benefit of microservices. By ensuring that each service is created as an independent part of the application, the application can be horizontally scaled

by replicating services that require more performance.

A disadvantage of microservices is the communication overhead. Since all communication uses IPC and does not utilize smart message buses, the data often has to be serialized or processed before sending. This can result in decreased performance required if the messages are of significant size.

Another disadvantage is re-factoring across module boundaries. Re-factoring inside module boundaries is simple with micro services, as each service should have a defined responsibility. However, if a re-factor requires changing the responsibilities of the services, the work involved can be significant if it results in changing responsibility from one service to another.

The microservice architecture is applicable to this project to increase the flexibility and scalability of the application. Since the required performance of use is unknown due to factors such as the number participants and the number of assignments, it is necessary to be able to appropriately scale the application if the need arises. By separating major components of the application into services, the potential bottleneck services can be replicated. The architecture also increases the flexibility as services can be more easily changed or replaced as long as the communication Application Program Interface (API) is consistent.

The architecture was used in the application to separate major components into services, see figure 4.1 for an overview. The separation was made to allow replication of the performance critical services in the future. Data storage was separated into a separate service to allow for scalability. Replication could for example be implemented by limiting the number of students stored on each instance, this would allow the storage to scale with the number of students in the system. Processing and metric extraction from the data was also separated into a separate service to enable scalable analysis with high performance requirements.

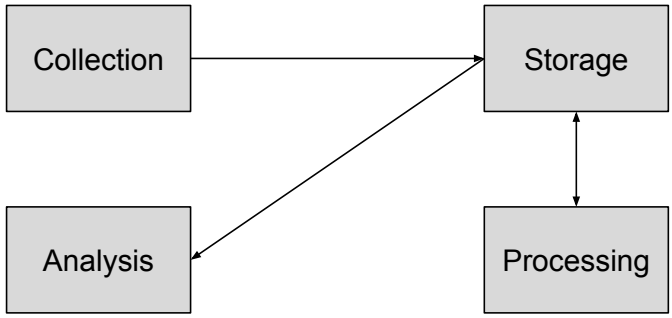

Figure 4.1: Architecture overview in the context of microservices

Communication
The communication between services uses standard protocols and data representations to allow for easy integration and communication with other services. The services communicate through the HTTP protocol via simple Representational State Transfer (REST) APIs to enable standardized interaction, and the requests and responses follow a JavaScript Object Notation (JSON)data structure.

4.5.2 Event Sourcing

Event Sourcing is a pattern concerned with recording action events in an application [15]. The patterns specifies an append-only store to collect operations being performed on a system. The operations are described using events that specify which operation has been performed instead of specifying the resulting state of the event. The store acts as a continuous stack of all events that have happened in the system, and can be used to replicate the current state of the system by playing the events off from the beginning.

Due to the flexibility requirement of the project, it is important that the metrics extracted from the collected data can change at a later time. This requires the system to store all the original data, and not just rely on the current state of the metrics when new data arrives. Event sourcing can enable the collection and storage of the original data, and allow for replaying the file changes if new metrics are to be extracted.

Event sourcing was used to store all the snapshots of the code that was collected. This made the full history of the files available at all times, and allowed the metric specifications and extractions to be changed without impacting the research. When a metric had to be added, the current metric database could be deleted, and rebuilt by running through all snapshots again from the beginning.

Projections were created to keep state for the data processing. The implemented projection was used to store the extracted metrics from the snapshots in a flexible manner. The structure used can be seen in section 4.5.3.

4.5.3 Data Structure

The relational structure of the extracted data can be seen in figure 4.3. The overview makes it easy to reason about specific relations in the data, and shows how each user can be seen as the root of a rooted tree.

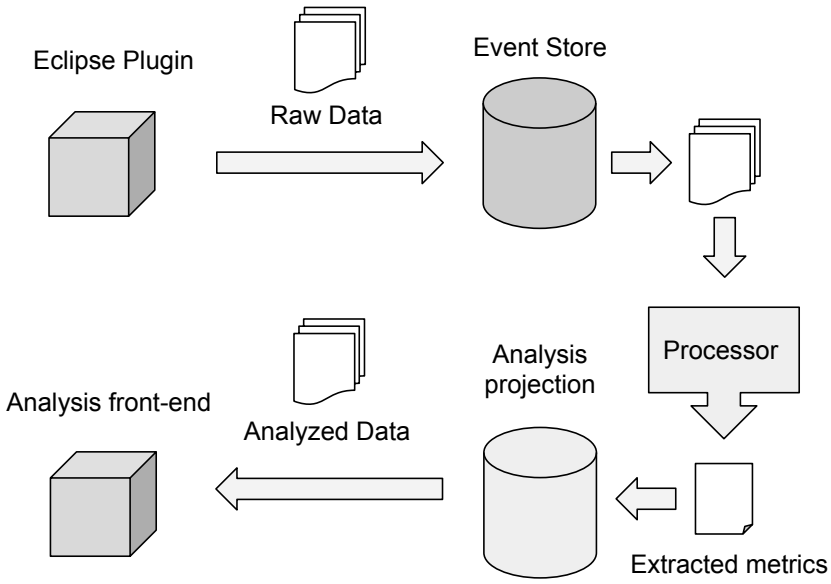

Figure 4.2: Architecture overview in the context of event sourcing

4.6 Data Collection

4.6.1 Collected information

- File content of all .java files edited in a logging enabled directory
- Test results from jUnit tests that were run
- All markers added by Eclipse related to the currently edited file
- Participation status and specified nickname

4.6.2 Eclipse plugin

The core functionality of the plugin is to collect data. The currently edited file and associated markers are collected whenever file change on disk (ie. when the user saves), and test results are collected after tests are run. To install the plugin it has to be downloaded from a plugin repository through the Eclipse plugin installer.

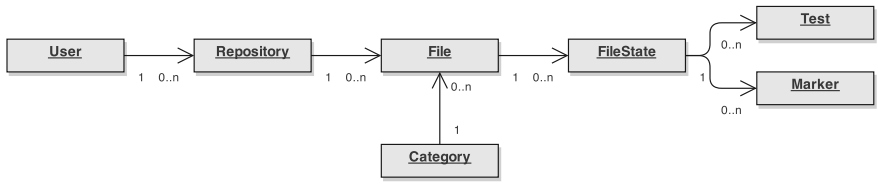

Figure 4.3: Relational structure of the data

Once the plugin is installed a pop-up will notify the user of a successful installation, and present a disclaimer that can be agreed or denied. If the user does not agree to the disclaimer, the plugin will not collect any data and will remain disabled. It can be re-enabled by accepting the disclaimer on the settings page.

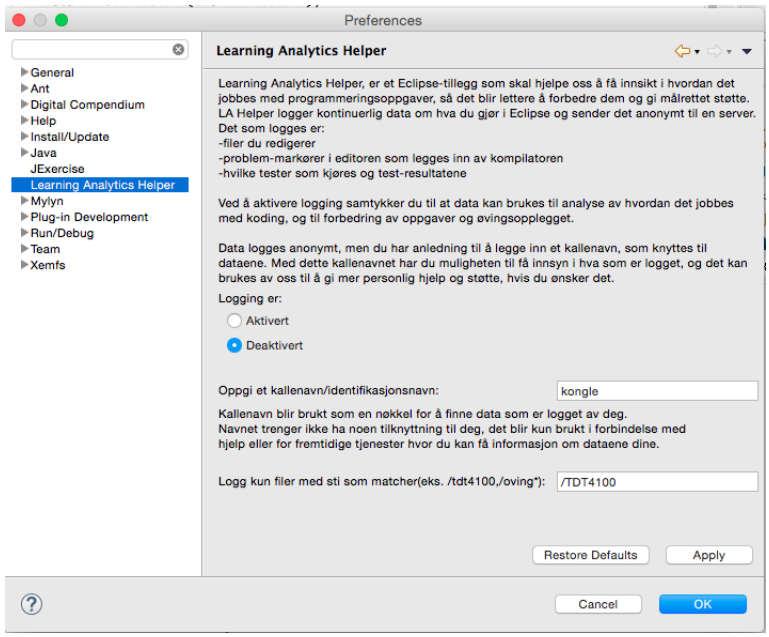

Figure 4.4: Settings page for the Eclipse plugin

The disclaimer is retrieved from the server to allow for changes in the disclaimer and to keep it up to date. If the disclaimer is changed, logging is immediately disabled and a new pop-up will be presented to the user with the updated disclaimer for them to accept again.

On the first install of the plugin a unique client id will be created for the user to keep the participant anonymous. This is the only identification used when transferring the collected

data.

Users also have the option to only allow data collection from specific directories. These directories can be specified on the settings page by writing a comma separated list of regular expressions that validates directory paths.

The user can specify an optional nickname in the settings page that will be stored on the server and associated with the given client id. This allows the user to identify themselves to the server in cases where this could be wanted, such as gaining insight into what has been logged about them. The nickname can be changed or removed at any time.

4.6.3 Data Processing

The file snapshots were stored in an event store, and then processed to extract metrics. The metrics were stored as a projection to the event store to enable a flexible storage of the metrics that could be destroyed and rebuilt if any changes to the processing was made.

Several metrics were extracted from the collected files.

Line count
To extract the size of the code, the number of lines was used. This decision was made to make it easy to process, and because the variability in number of lines is assumed to be lower than the variability in total size of the code.

To count the lines, a primitive count of line break characters was used. The consequence is that lines that do not influence the program is preserved in the count. This includes comments, empty lines and unreachable code. The reason for including this is that even though it is not indicative of a change in the program, it is indicative of a progress in the assignment. The assumption is that any typing, even typing that has no direct impact on the running of the code (such as writing comments), is an intention of progress in the assignment.

Assignment classification
Most of the assignments requires specific classes and interfaces to be created so that the included test cases would run. This enabled the classification of files into assignments. The classification was made based on the name of the class or interface contained in the file, as well as the package name specified in the file. This way of classifying was based on the structure provided in the exercise description where certain class names were required. The package name was included in the associated tests, and the assumption was that students would not change the package name when running the tests.

Number of failed tests
The number of failed tests related to a given snapshot of a file was counted based on the last known test run by the student. The initial number of failed tests, before any tests were run, was assumed to be the total of all later tests run.

Working time
The students were not required to complete each assignment in one continuous session. Therefore idle time had to be accounted for when measuring time between states and the total time spent on an assignment. To account for this, any time step between two adjacent snapshots that was greater than 10 minutes, was assumed to be idle time and was normalized to 2 minutes.

This means that if the student spent more than 10 minutes searching for a solution without saving any files, this would be visualised as a working time of 2 minutes. However, whenever idle time occurred a flag would be set and the length of the idle time was recorded to enable visual cues to be displayed in the visualizations.

4.7 Analysis and Experimentation

The analysis application is a browser-based interface that enables visualisation of metrics in graphs. The expressions for the curves can be created through an interface and can be any combination of the available metrics. This allows experimentation with different expressions for similar metrics, or comparison of different expressions. The application also offers several levels of detail in the data set for different purposes.

Initially the aim was to create an interface that allowed visualizing several expressions for progress based on the metrics collected. The expressions should be created and edited using the interface to support the experimental research. However, since the initial specifications were unknown and dependant on the needs of the research, the development was reliant on several development and evaluation iterations.

4.7.1 Development Iterations

Wieringa [16] argues that design science can be thought of as nested problem solving, where one knowledge question is followed by a practical questions, which is followed by a knowledge question, and so on.

Knowledge questions in this sense are questions with answers that change our knowledge of the world, while practical questions require answers that change the world according to some goal.

This approach was specifically used in the development of the analysis tool. The tool followed several iterations as the design and implementation was changed and expanded. Each change gave rise to new knowledge that lead the research closer to a solution, but also lead to new practical problems in the evaluation. These problems were grounded in the need to further explore the findings and increase the understanding of the data set.

This explorative approach was essential to the work flow as the problem domain was loosely specified due to the pioneering approach used. This meant that the end result was not known in advance, and an incremental process of exploration was needed.

These iterations are specified in this section, where each new view gave rise to new knowledge, that required new views to be created to gain further knowledge.

4.7.2 Explore View

The overview provided by the explore view, was aimed at comparing expressions. The view enabled the user to create custom expressions based on the metrics collected from the snapshots. The expressions were created by specifying an expression for the *x* and *y* axis on a graph. The expressions were parsed and allowed for any mathematical combinations. The expression creation dialog can be seen in figure 4.5.

The explore view created a way to visualise several expressions across a select representative of the data set. This allowed initial comparisons to be made, and made it easier to determine which expressions warranted further inspection and validation. A screenshot of the explore view can be seen in figure 4.6.

Each expression was visualised for a selected participant and exercise, but also for a subset of random users, and an aggregation of all users in a subset of the exercises. This was done to examine how the expression was visualised for different contexts.

As the view provided only limited view of how the expressions could be used to compare students, a need arose to be able to examine in more detail how a given expression effected participants in the same exercise. This view would enable more inferences regarding the ability of the expression to compare different participants under the same context.

Iterations

The initial iteration of the explore view revealed the need to remove idle time in the data set. The initial graphs that were visualized were rendered including all the time between each state. This meant that if a student had a couple of days break between each coding session, the idle time would represent almost all of the graph.

This lead to creating of a new metric, working time, that was normalized to the start of the exercise, and removed any idle time of more than 5 minutes. See section 4.6.3 for details. The idle time was visualised on the graph as a marker that displayed details of the length of the idle time on mouse over. A comparison between the visualisations of idle time can be seen in figure 4.7.

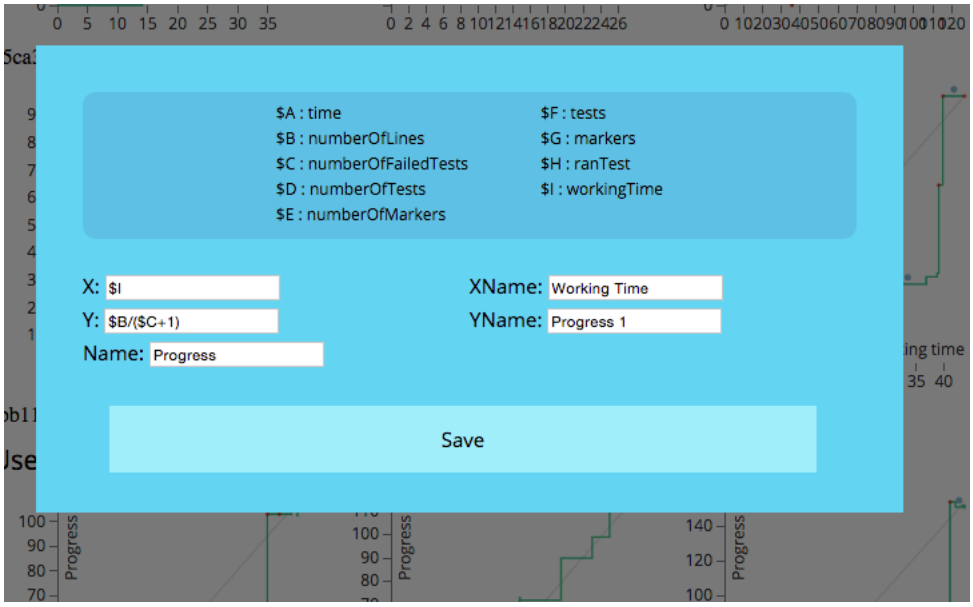

Figure 4.5: Screenshot of adding a new expression

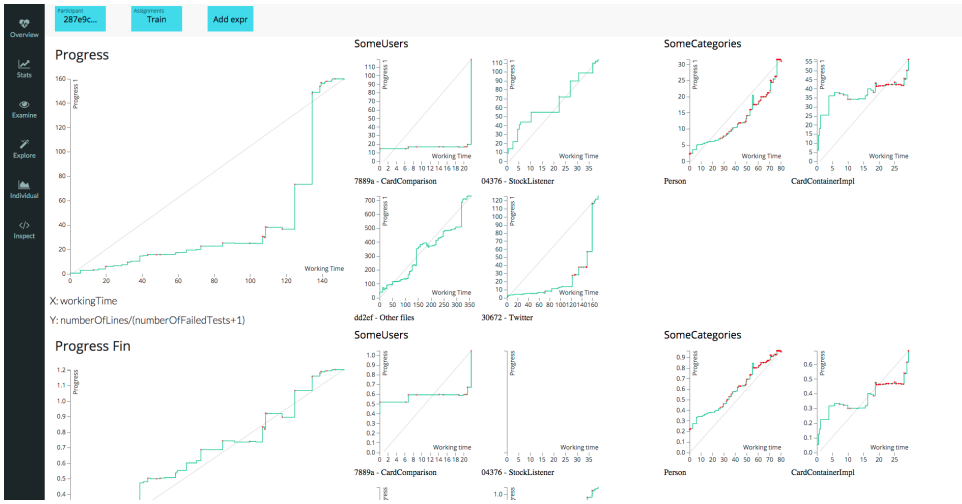

Figure 4.6: Screenshot of the explore view of the application

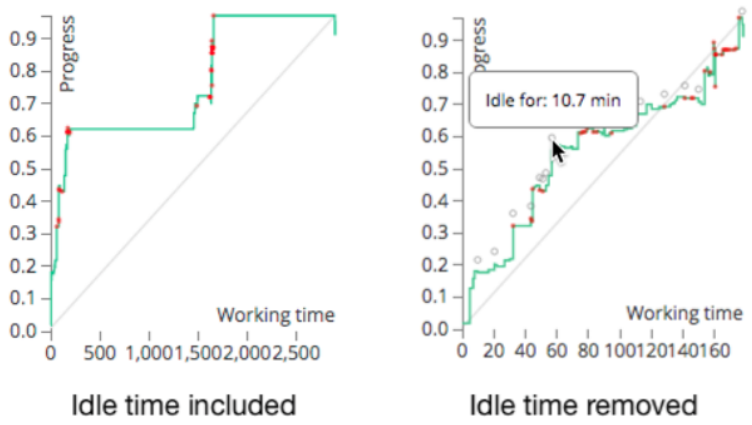

Figure 4.7: Comparison of including and removing idle time

4.7.3 Participant comparison

The broad overview of the explore view lead to a new practical problem of creating a more detailed view that could compare more users in the same exercise. At this point in the research, some expressions had been found that showed potential, but a more detailed examination of how they effected students in the same context was needed. This would also allow easier identification of breakdowns, both in individual students and breakdown trends in exercises.

The view should still be able to display several expressions at a time to determine the differences between the visualisations, but the focus should lie on examining the effect of each expression on students in a single context.

The solution was a view where an exercise was chosen, and each expression was visualised for each participant that had worked on the exercise. The expression was also visualised for the average of all participants. This gave an overview that allowed comparisons to be made between participants, as well as comparisons between individual participants and the average of everyone. A screenshot of the view can be seen in figure 4.8.

This made it easier to spot participants that deviated from the norm, and to identify trends in each exercise. The view also made it easier to spot breakdowns as several participants were displayed at the same time.

4.7.4 Snapshot browser

After having started to spot breakdowns in the progress visualisations of the participants, it was necessary to examine if the identified breakdowns were representative of a student

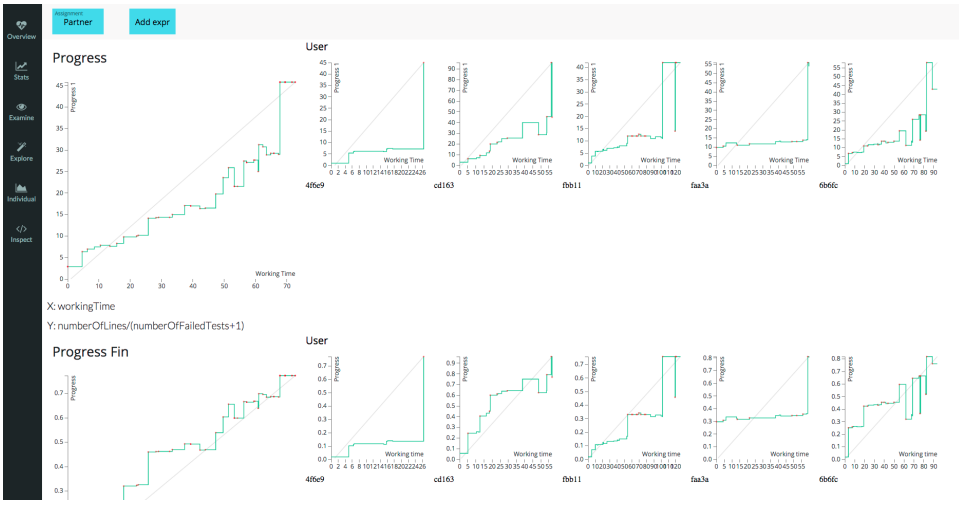

Figure 4.8: Screenshot of the participant comparison view of the application

getting stuck. Initially this was done manually by inspecting the snapshots in a Git snapshot browser, however finding the exact state of the breakdown became an issue since the browsing was not linked to the visualizations.

To aid this need, a snapshot browser was created, inspired by Heinonen et al. [8]. The browser enabled a researcher to chose an assignment and a participant, and would allow the researcher to easily progress through snapshots with a slider bar. The view showed the syntax highlighted code in each file for the current state, the tests results for each file, and showed the visualizations of the created expressions for the participant. A screenshot of the view can be seen on figure 4.9.

When the slider was dragged, the contents of the files were updated, and a marker on the graphs gave an indication of the location of the state according to the curve. The visualisations included a red marker on the graph each time a test was run by the participant. An overview of the most important features can be seen on figure 4.10.

This allowed for much easier inspection of breakdowns as one could immediately identify which snapshot was the start and end of the breakdown. Since the test results were also displayed it was easy to see which tests were currently passing or failing, to identify the result of each change the participant made.

Iterations

The initial iteration of the snapshot browser only included the files and the visualisations with the slider bar. The initial need was only to make it easier to identify the position of

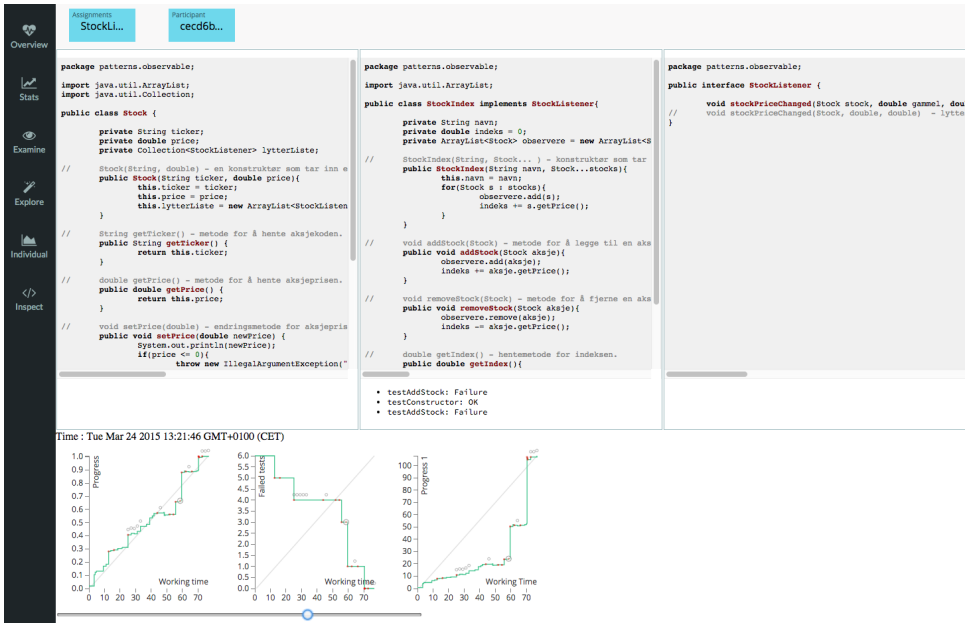

Figure 4.9: Screenshot of the snapshot browser of the application

Figure 4.10: Overview of some of the features in the snapshot browser

the current state on the progress graph.

After using the snapshot browser for a while it was apparent that knowing the current state of the tests would be interesting, to make it easier to identify which part of the exercise that the student was completing at any given time. This lead to the inclusion of the test results for each of the displayed files.

Some breakdowns suggested that participants managed to suddenly pass multiple tests at the same point in time. The assumption was that these students did not perform tests frequently through the exercise, and therefore the failed tests metric was inaccurate in the time leading up to the passing tests.

To ensure that the assumption was correct, it was necessary to add red indication on the graph to display when a participant ran a test. This lead to a conclusion that these cases were usually a result of participants running tests infrequently.

4.7.5 Client Inspector

Due to low participation numbers in the beginning of the study, a need to increase the participation arose.

To increase the number of participants an application was developed that enabled more information to be given to the participants regarding the data collected about them. With inspiration from motivation theory, the application was intended as an incentive for the students to participate in the study. The aim of the application was to increase the students motivation for completing the weekly assignments by improving intrinsic motivation.

Self-determination Theory is a theory detailing factors which influence human motivation. The theory is centered around the concepts of intrinsic and extrinsic motivation. Extrinsic motivation is motivation to perform a task because it is a step towards a greater goal, while intrinsic motivation is motivation to perform a task due to the apparent reward of the task itself.

The theory identifies three important psychological needs that influence intrinsic motivation; competence, autonomy and relatedness [17].

Competence is the feeling of responsibility for the progress made, and makes the person feel that time has been well spent. Autonomy is the need for freedom to chose when and how to perform the task. Relatedness is the need to feel part of a group and have close relations with others.

Endomondo [18] is an exercise application for smart-phones with the goal of increasing motivation for exercising. The application tracks the users position and elevation during running laps and displays the information in a graph over time. This visualization works as a concretization of the progress made by the user, which improves the intrinsic motivation of the exercise.

This directly relates to the users need for competence to increase the intrinsic motivation for each run, as opposed to the extrinsic motivation of running to get fit.

Inspired by applications such as Endomondo, the goal was to satisfy the students need for competence to increase their motivation for completing the weekly exercises. Even

though the extrinsic motivation was to pass the course, the aim was to increase the intrinsic motivation to of each exercise.

The application developed gave each participant an overview of the completed exercises registered, and showed a graph of their lines of code in the exercise over time. This acted as a visualization of their progress over time of each exercise, and gave an impression of the work done. It also allowed them to compete with themselves in the time spent on each exercise and the amount code written over time.

Figure 4.11 shows the application available to the participants. By entering their nickname they got access to the visualisation of data collected from them. The application presents a list of the exercises classified, and allows the participant to visualise the lines of code over time for each exercise. The number of tests failed is also visualised as a red area to show how it decreases over time. The number of markers are also visualised as a yellow area.

The optimal solution would have been to show students their progress over time based on the progress metric created, however at the time the application was developed the research was not yet completed.

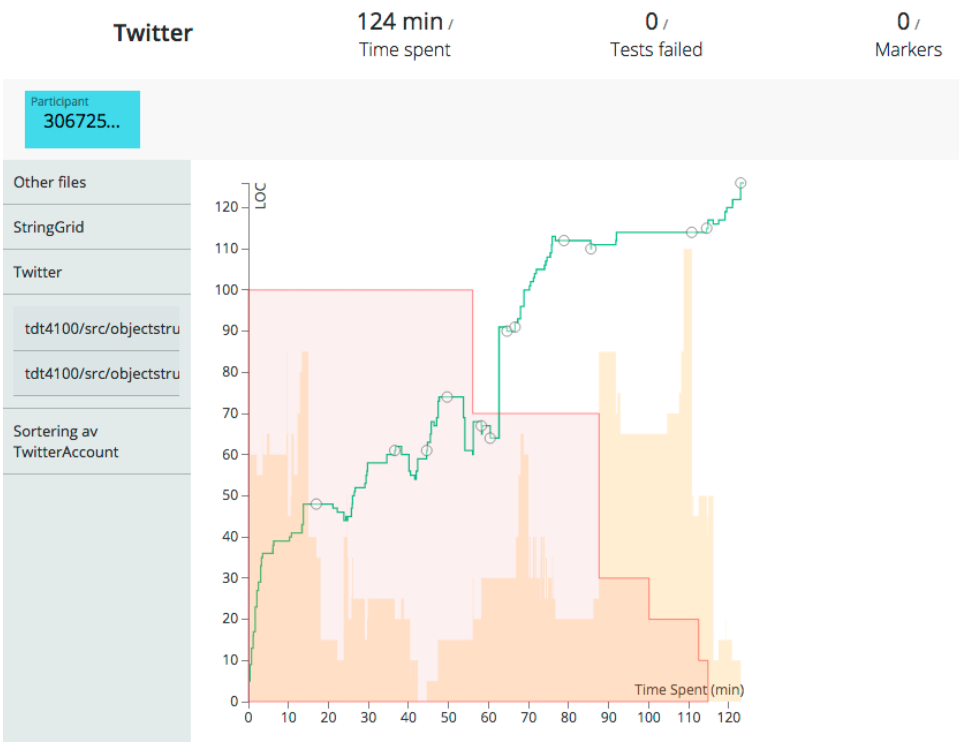

Figure 4.11: View available to participants to inspect the data collected from them and increase motivation

Chapter 5

Demonstration

This chapter provides a demonstration of the usage of the software, to enable further evaluation of the implemented software.

5.1 Student Experiment & Context

To demonstrate the use of the software application, as well as to answer the research questions, an experiment was conducted throughout the project.The aim of the experiment was to answer the research questions given a data set from students in TDT4100. The students were asked to participate in class, and the ones who wanted to take part had to download the Eclipse plugin used to collect data. This section demonstrates the use of the application in the experiment.

5.2 Data Collection

5.2.1 Ethics

Learning Analytics is seen as a benefit for students and instructors alike, however it also comes with a challenge of ethics to ensure that the data collection does not have negative impacts on the students. Several researchers have examined the current ethical challenges of Learning Analytics to ensure that the research is conducted with care towards the privacy of the participants[19, 20].

Slade and Prinsloo [19] highlights three overlapping categories of issues surrounding ethical considerations in Learning Analytics; the location of the data, privacy and informed consent, and classification and management of data

In this experiment several measures have been used to ensure the privacy of the participants. This section will elaborate on these measures in relation to the categories presented.

The location of the data
The collected data was stored on a virtual machine on the NTNU campus. This ensured that there were no issues with 3rd parties using different conflicting regulations, or having access to the data.

Privacy and informed consent
The experiment was opt-in and students in the course TDT4100 that were willing to participate had to manually download the plugin needed. Before any data was collected the participants had to agree to the terms of the experiment. The terms explained the purpose of the experiment, how the data would be used, and what data would be logged.

To ensure that the collected data was anonymous, the participants were given a unique random id for identification purposes when they installed the plugin. No information was stored regarding each participant other than the random id.

To ensure transparency of the experiment, participants could access a list of files that had been collected. Since each participant was anonymous, a separate solution had to be developed to allow this.

Each participant could provide an optional nickname that was paired with the unique id and allowed a temporary identification token that could be used to show the list of files. The nickname could be changed or removed at any time.

Classification and management of data
The students were not categorised based on the results of the data. Only the individual progress of assignments were inspected. This means that no labels were given to the participants in the experiment.

5.2.2 Participants

The data was collected from students in TDT4100 that volunteered to participate in the experiment. The students agreed and downloaded the data collection plugin that was installed to Eclipse; the IDE they use while working on assignments.

Initially only 16 students volunteered to participate in the experiment. This lead to the development of a web-based interface where the participants could get more information about their data, including a visualisation of the lines of code they had written over time through an assignment (see section 4.7.5). This application was developed to create an

Sample	Count
Participants	34
Number of students in course	450
Exercises	29
Exercise files	517
Exercise snapshots	6696
Exercise tests run	3130

Figure 5.1: General statistics over the sample set

incentive for students to participate by giving them motivational value from the collected data. This lead to a significant increase in participation and the final number of participants was 34.

5.2.3 Collected data

Data was collected each time the participants changed, created or deleted a relevant file on disk, as well as any time a test was run. A summary of the collected data can be seen in section 4.2.1.

Summary statistics about the experiment and collected data can be seen in figure 5.1.

5.3 Data Analysis

The data analysis began by creating an expression that could be used as a starting point for further experimentation, and to validate that the metrics in use were relevant. The aim was to start broad by examining how the expression was visualised in several different contexts, including different students working through different exercises, and aggregate data about all users in a given exercise. This was done to get a broad look of how the expression affected different scenarios and how and if these could be compared using the expression.

The initial expression created was used to examine if the hypothesis that progress could be measured from failed tests over time had any merit. The expression was inspired by Blikstein [5] where the size of the code was plotted over time to reveal the work flow of students. The expression used was to plot the number of lines in the code divided by the number of failed tests. The aim was to use the number of lines to get a detailed curve of the changes made by the student, while retaining the metric of progress from the number of failed tests.

$$\frac{TotalNumberOfLines}{NumberOfFailedTests + 1} \tag{5.1}$$

As seen in figure 5.2 this expression cause non linear progress curves in most instances. This is due to the division by the number of failed tests which cause a rapid increase as it goes towards zero. In instances where the student did not perform any tests, the curve just visualised the lines of code over time, and often results in a more linear curve.

The advantage of the expression is that it does show that the student is progressing through the exercise, and that the identifies metrics can be used, at least to some extent, to indicate the progress.

The disadvantage of the expression is that it is not linear, and makes it difficult to assess the details in the progress of the student. It also makes it difficult to compare students. As the curve is not linear, the time taken to pass each test has a significant impact on the shape of the curve and makes it difficult to asses the detailed progress.

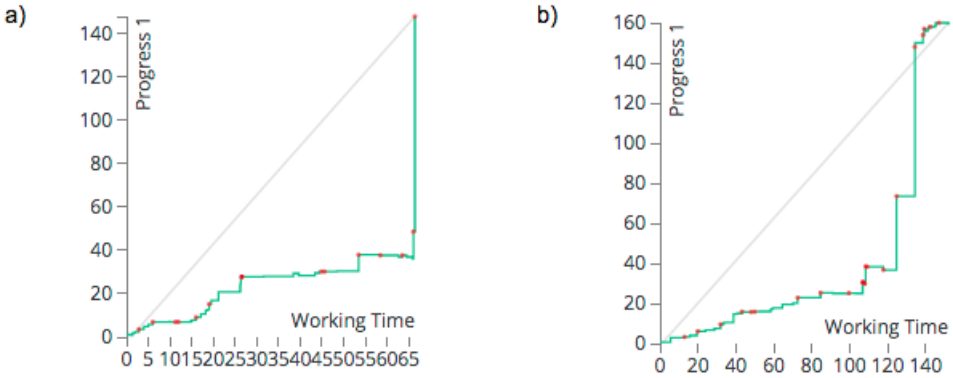

Figure 5.2: Initial expression for progress

To create a linear expression, the number of failed tests had to be moved from the divisor. By moving the term out and replacing it with the percentage of the passed test, it could be used as a scale from 0 to 1 (eqn. 5.2). This would make the progress component more linear and easier to compare to the lines of code.

$$TotalNumberOfLines * \frac{TotalNumberOfTests - NumberOfFailedTests}{TotalNumberOfTests} \tag{5.2}$$

This expression gave a significant increase in the linearity of the provided graphs. However, the problem was still how the lines of code could be used without using it as a base term for the progress. In this expression, the progress is still increasing more rapidly

as both components of the multiplication are increasing. This can easily be seen in the progress curves in figure 5.3 as the rate of change of the progress is rapidly increasing towards the end of the exercise.

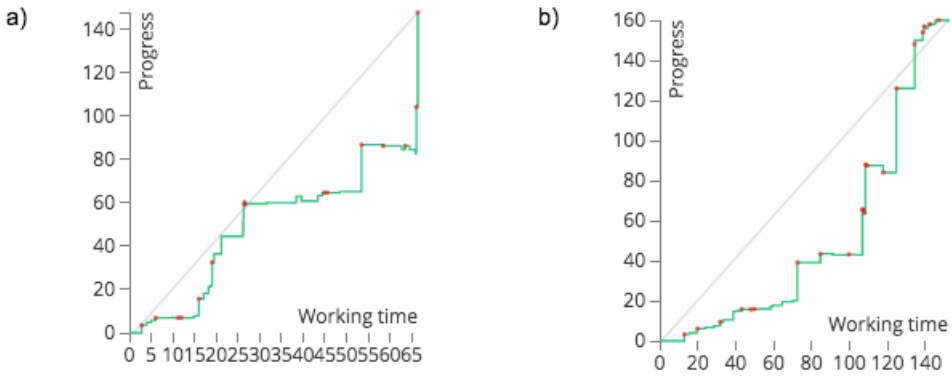

Figure 5.3: Second expression for progress

The other issue is that the progress is not normalized. The range of each participant is different, resulting in progress values that are difficult to compare. If the progress was normalized between 0-1, the progress output would easily translate to the percentage of progress through an assignment.

The problem with creating an expression that is normalized, is that the number of lines of code is unknown until the assignment is finished. One approach would be to only support progress after the assignment is finished, by using percentage of lines compared to the total line count in the finished exercise. However, the problem with this approach is that breakdowns could not be identified until after a student had completed the assignment. This means that future uses such as real-time analysis and guidance is not possible.

To solve the issue it was decided to try to use the total line count in the exercise solution created by the course instructors. This introduces a variability in the expression, as it is unlikely that the students will write the exact same amount of lines. However, due to the size and structure of the exercises, it was assumed that the variations would be relatively small.

Expression 5.3 shows the resulting, and final expression. The expression consists of two weighted components; the percentage of lines written relative to the total in the proposed solutions, and the number of tests passed relative to the total tests ran.

The components are weighted differently, with the test percentage weighted at 60% vs. 40% for the lines of code. This has been done due to the variability in the percentage of lines of code, but also due to the reasons mentioned in section 3.3.1 that the percentage of total tests provide a better measure for the total progress through the exercise.

$$\frac{TotalNumberOfLines}{NumberOfLinesInSolution} * 0.4 + \frac{TotalNumberOfTests - NumberOfFailedTests}{TotalNumberOfTests} * 0.6 \tag{5.3}$$

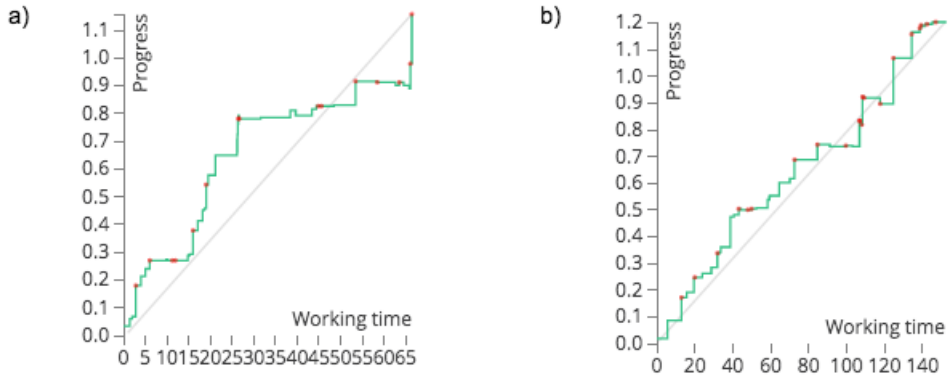

Figure 5.4: Final expression for progress

Breakdowns

When identifying breakdowns, the participant comparison view was used (see section 4.7.3). The assignments with the most participants were examined to identify progress curves that had a significantly low increase in progress over an extended period of time.

When these curves were identified it was necessary to validate that these were in fact breakdowns. This process began as a manual process of looking at each snapshot of the student code and trying to identify the relevant snapshots. This was difficult and lead to the creation of the snapshot browser (see section 4.7.4) which was inspired by the CodeBrowser created by Heinonen et al. [8].

This allowed a more detailed analysis of the snapshots exactly at the point in time they were identified on the progress curve. The browser made it significantly easier to determine the validity of each instance, as the reason for the low progress was easier to identify.

5.4 Experiment with course assistants

To perform a more controlled experiment, where the experience and skill of the participants was known, another experiment was conducted with some of the course assistants.

The course assistants were students that had previously been enrolled in the course, and were hired to help the current students with the weekly exercises.

Since the assistants had already completed the course, the assumption was that their experience level would provide a more controlled environment, where the progress metric could be used as a standard to compare the students with. The aim was also to demonstrate and validate the usage of the progress metric in a different environment.

The experiment was conducted by choosing an exercise to be completed by the assistants while snapshots were collected of their work. The exercise was one of the optional exercises for the students, which allowed a direct comparison to the results of the students.

The participating assistants were instructed to download the plugin, and complete the given exercise at their leisure. Another instance of the Learning Analytics platform was created to store the data for the experiment to ensure that it would not be mixed in with the students. The data was collected anonymously, to preserve the privacy of the assistants.

Three assistants participated in the experiment, and they all completed the same exercise.

Chapter 6

Results

The progress curves are presented together with a curve of number of failed tests over time to highlight the relation between progress and number of failed tests. The red markers on the curves represents tests run by the participant, and the gray circle is used to highlight areas of interest on the curve that are referenced in the text.

6.1 Progress

The final expression found to determine the current progress of a student in an assignment was:

$$\frac{TotalNumberOfLines}{NumberOfLinesInSolution} * 0.4 + \frac{TotalNumberOfTests - NumberOfFailedTests}{TotalNumberOfTests} * 0.6 \tag{6.1}$$

This expression is normalized to be in the range of 0-1 making the progress easy to compare as a percentage. The expression consists of two weighted components; the number of lines written relative to the number of lines in the solution to the assignment, and the number of failed tests in relation to the total number of tests ran by the student.

Figure 6.1 and 6.2 shows some of the progress curves of the participants in the same exercise. These curves are a representative selection of the observed progress curves in the exercise.

Figure 6.1 shows two students with a relatively linear progress over time through the exercise. Both participants perform tests frequently through the exercise, with participant *a* running the most tests. Participant *a* spends about 150 minutes on the exercise and the last progress is measured as 1.2. Participant *b* spends about 100 minutes on the exercise, with

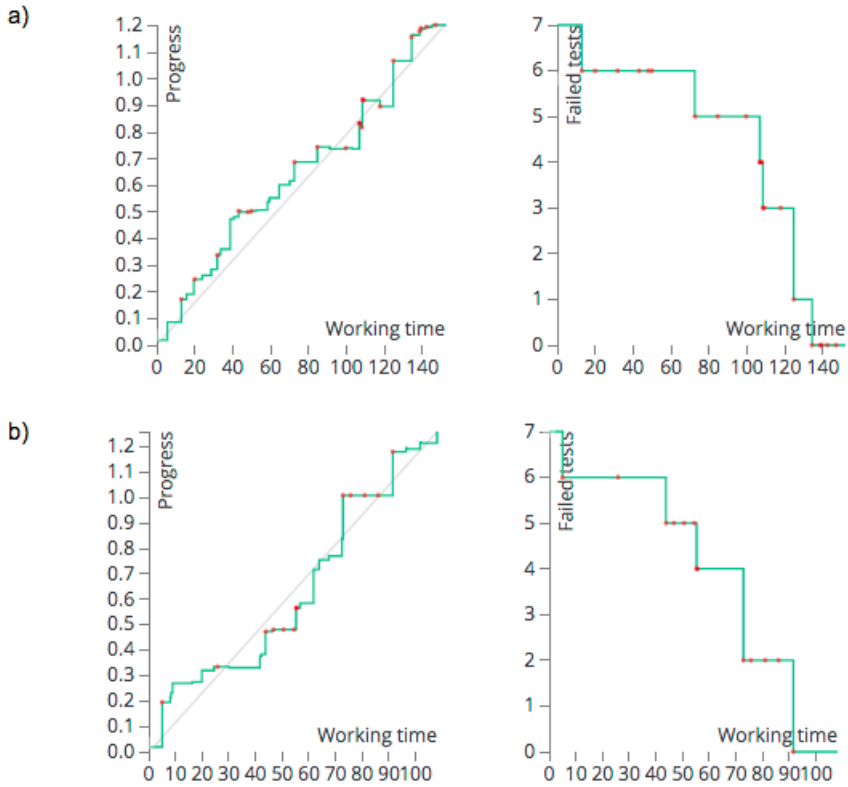

Figure 6.1: Progress curves following relatively linear progress

the last progress measured at 1.2.

Figure 6.2 shows two students with a non-linear progress over time through the exercise. Both participants perform tests infrequently through the exercise. Participant *c* spends about 50 minutes on the exercise and the last progress is measured as 0.8. Participant *d* spends about 40 minutes on the exercise, with the last progress measured at 1.1.

Participant *c* only ran tests once in the assignment. After running the test the progress metric increased by 60%. Participant *d* ran tests 5 times throughout the assignment, however the last test run resulted in an increased progress of about 30%.

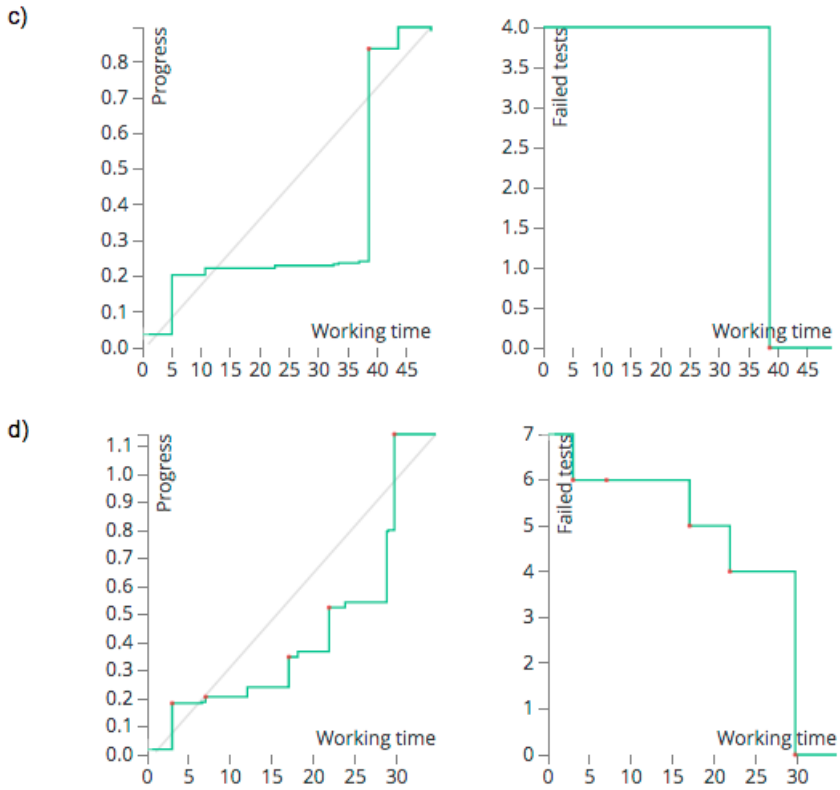

Figure 6.2: Progress curves following non-linear progress

6.2 Breakdowns

On figure 6.3 a possible breakdown is present after 70 minutes, when the rate of change of the progress is dramatically decreased. 30 minutes after this, the progress increases instantaneously by 30% indicating a resolution of the breakdown.

On figure 6.4 a possible breakdown is present after only 10 minutes. In the following 50 minutes the increase in progress is fluctuating, however the overall increase is negative until the exercise is finally solved after a total of 60 minutes.

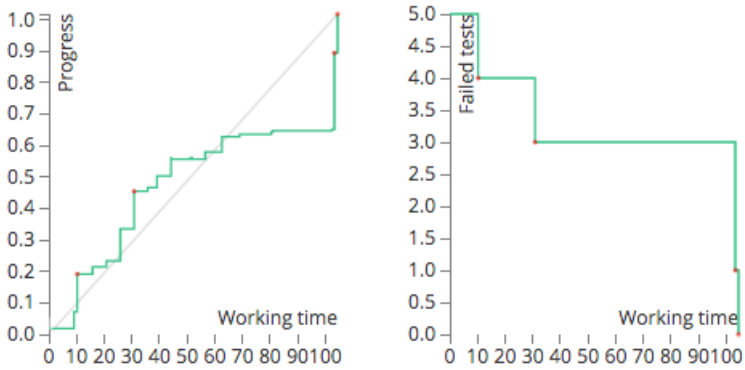

Figure 6.3: Breakdown identified after 70 minutes

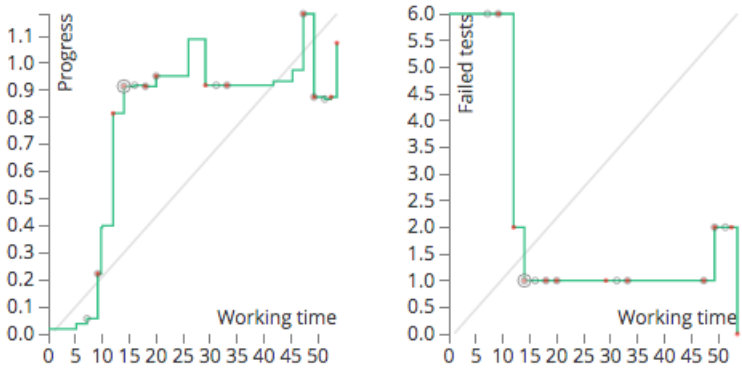

Figure 6.4: Breakdown identified after 10 minutes

6.3 Experiment with course assistants

Figure 6.5 shows the progress curves of the experiment conducted with the some of the course assistants. There is a significant variation in the progress curves between the assistants, with increasingly linear curves.

Assistant *a* spends 25 minutes, and only run tests twice. In total only 1 distinct test is run and passed, out of a total of 6 for the assignment. Assistant *b* spends 35 minutes, and performs tests three times. In total all 6 of the exercise tests are run, and 5 of them are passed at the last state. Assistant *c* spends 45 minutes and performs tests frequently. In total 8 out of a possible 6 are run, of which only 3 are passed at the end state.

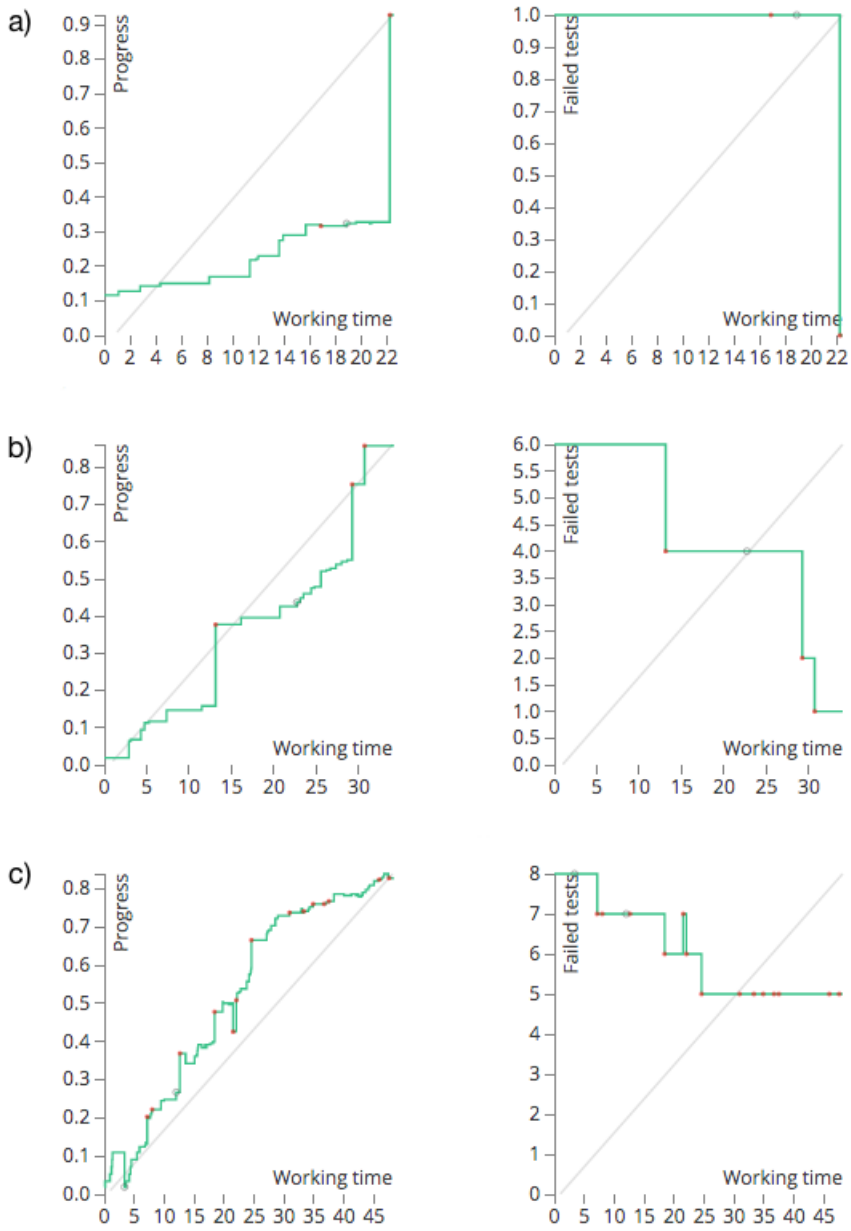

Figure 6.5: Overview of the progress curves of the assistants while working through one of the exercises in the course

Chapter 7

Discussion & Evaluation

7.1 Discussion

7.1.1 Progress

The results indicate that the hypothesis is correct, progress through an assignment can be estimated through a combination of the lines of code written and the number of failed tests. All curves have an increasing progress over time, with most ending at a progress measure of 1 +- .2% which was expected due to the variability of the number of lines of code compared to the solution.

The results have provided several interesting observation regarding the progress measure, and the activity of students.

Number of tests

The results indicate that the total number of tests in an exercises impacts how accurate the progress can be measured. A low number of tests result in more significant instances of instantaneous progress, however these instances decrease with a higher number of tests in the exercise.

This can be seen on figure 7.1, where participant *a* is working through an exercise with many tests, and participant *b* is working through an exercise with fewer tests. The difference in the magnitude of instantaneous progress is significant between the two, where the instantaneous progress of participant *a* is 8.5%, while participant *b* has an instantaneous progress of 15%.

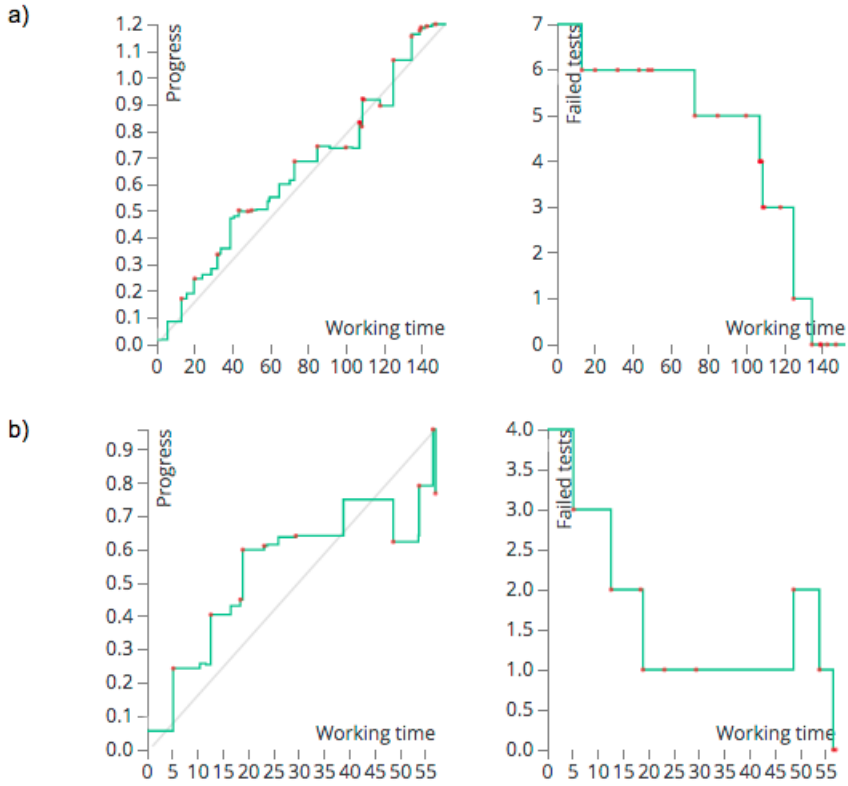

Figure 7.1: Comparison of exercise with many tests, and exercise with few tests

This indicates that the initial assumption is correct; increasing the number of milestones in the exercise will increase the accuracy of the progress measure.

The reason behind this is the weighted expression used to calculate progress. Since passing all tests always impact the progress by 60%, each test passed impact the progress with a given percentage. Increasing the total tests accompanying the exercise, results in a decrease in the percentage of progress gained per test. This is necessary as discussed in section 3.3.1 due to each test representing a milestone towards the end goal.

Type of tests

In exercises where the tests covers overlapping methods or logic, the progress is seen to be fluctuating, and often contains sections of decreasing progress. This is prominent in assignments where a single method is incrementally improved to handle increasingly

specific and complex inputs. The result is that previously passed tests may fail when code is re-written or extended to make new tests pass. This results in a decrease in progress, as the program no longer functions according to the requirements of the previous test.

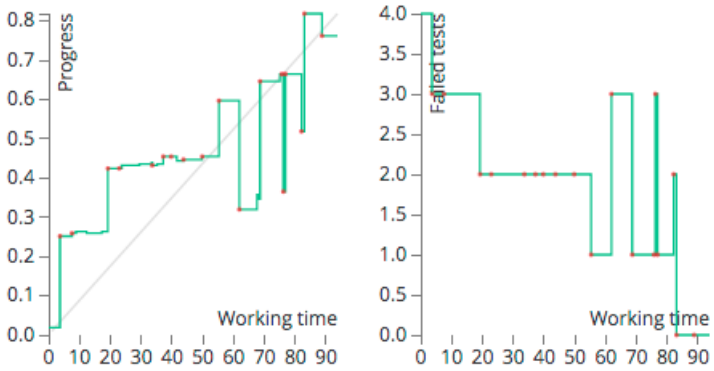

Figure 7.2: Exercise where code had to be re-written to pass later tests

An example of this scenario is seen in figure 7.2 where the exercise was focused around one specific method, and the test cases were testing increasingly complex input to the method. The result was that to pass the later tests, the student had to re-write code that the previous tests covered. The change in the program resulted in the earlier tests no longer passing. The impact on the progress curve can be seen after 60 minutes when the progress starts to decrease.

Even though the progress curves resulting from these types of tests show decreasing progress, it does not necessarily mean that they portray incorrect information. Since the only way of determining the correctness of a program is through testing functionality, it could be argued that since the tests previously passed are starting to fail it must mean that progress is decreasing. This is because required functionality of the program has been removed, or is not working as it should, even though it was working previously.

One can also argue that the progress should still be positive as the student is still progressing through the assignment, however that must mean a redefinition of the progress metric and would possibly rely on different metrics, such as time spent or effort made.

Therefore it is debatable if these issues should be corrected through a change in exercise design, or if it is an accurate representation of the progress.

Structure of exercise

It is also seen that the structure of the exercise has an impact in the progress portrayed. If the exercise is structured as several independent steps towards a final solution, the progress is seen as more linear.

This is especially noticeable when the exercise is divided into several files that can be completed one at a time. In these cases it is seen that the progress is more linear, and students are testing more regularly.

The observation indicates that the forced structure and modularity is effecting how the student work with the assignment. One possible reason is that the exercise is seen as a combination of several small independent steps, and therefore it is natural for the student to test that each step is correct after each part is finished.

Frequency of test runs

The frequency of test runs by the participants is seen to have a major impact on the accuracy and representation of the progress metric. Fewer test runs are seen to lead to more variability in the progress metric, with significant instances of instantaneous progress. When the frequency of test runs increase, the progress visualisation is seen to be more linear and with a lower variability in the rate of increase in progress.

The most problematic instance happens when multiple tests pass at the same time. This produces an instantaneous and significant increase in the perceived progress of the student. This reported progress is often not an accurate reflective of the progress of the student, as inspection of the code has shown that the students are having a more linear progress before the tests are run.

An example of this can be seen in figure 6.1 and 6.2 where participant *a* is running tests frequently through the assignment, while participant *d* is only running tests a few times. The effect is that participant *d* gets an instantaneous 34% increase in progress the last time the tests are run, when four tests are passed at the same time.

The underlying issue seems to be that the tests are manually run by the students at their discretion. This means that the metric for number of failed tests will be increasingly inaccurate the more time passes between test runs. This results in the inability to rely on the number of failed tests as a measure for the correctness of the program, which leads to an unknown progress state.

An ideal solution to this problem would be to run the tests automatically for each snapshot in the exercise. This would ensure that the metric would always be accurate, and as a result the progress metric will more accurately reflect the state of the program.

Relative number of lines

The results indicate that the relative number of lines is a decent metric to estimate an increase in progress in the exercise. The argument against using the total number of lines in the solution as a target for the students is centered around variability. Since there are an

infinite number of ways to create a program, it is assumed that relating the students work to the solution would be very inaccurate.

However, the results show that the number of lines of most of the students program are within 50% of the solution. This seems like a high value, however it is still low enough to use this measure as an increase in progress as it only has a maximum effect of 20%, due to the weight of the expression component.

The negative aspect of this measure is that the progress metric will have an error element. If the student writes less code than the solution, the progress will never reach 100%, and if the student write more code than the solution, the progress will surpass 100%. In the experiment it was seen that most students completed their exercises with a progress metric of 100% +- 20%.

For the purposes of this experiment this error measure is acceptable, as the benefits of the incremental progress outweighs the disadvantage. The exact progress metric is not required as the aim is to examine the progress over time. If the error measure was higher it might have been an issue as it would have a significant impact on the progress caused by passing tests, and would potentially render the passing tests negligible.

7.1.2 Breakdown

Several instances of low increase in progress over time were identified, and further inspection showed that in most of the cases these situations were in fact breakdowns. This suggests that it is possible to identify breakdowns by inspecting the progress curve of the students through exercises.

This section will provide further details of the breakdowns identified in the results, to further validate the presence of a breakdowns.

Correctly identified breakdowns

Incorrectly scoped variables

Figure 6.3 shows a possible breakdown after 60 minutes. On closer inspection of the participants code it is seen that this is a correctly identified breakdown. The student is struggling to correctly implement a function that counts all the passengers, as well as a function that counts the weight of all the cargo in a train.

Through inspection it is seen that the source of the error is the scope of the total count variable. The student has defined the variable in the class scope and does not reset the count each time the function is called. This results in the count increasing each time the counting function is called.

Instead of repeatedly running the test cases given by the assignment, the student is producing a custom output to verify the correctness of the code. For several minutes the students is changing the output and possibly trying to identify the location of the error through manually verifying the output of each command. This can be seen in the code as the student writes comments containing the additions made, ex: totalWeight = 2000 + 800*80 = 18000

After this the student tries to change the variable names of the counting variables. The variable names are now identical to the ones used in the assignment solutions given to students. However, this still does not work.

The next state is recorded the next day, after a 20 hour long break. After one minute a state containing an update that changes the scope of the variables is posted, and results in the 30% increase in the progress. After another minute the final state is recorded, correcting a reference to one of the counting variables, and resulting in a completed assignment with ~100% progress.

This is a logic problem that is difficult to automatically identify and provide automatic aid for the student. Whether the issue lies in the students understanding of object oriented principles, or if it is a simple oversight is difficult to determine.

The fact that the final state includes an immediate solution to the problem, and is recorded a day after the initial breakdown, could suggest that the student had to seek help to solve the problem. This is a further validation that a significant breakdown was identified.

Misspelled method name
On figure 6.4, a breakdown is identified after about 10 minutes due to the overall low progress for the next 50 minutes. Even though there are peaks in the progress curve in this area, the overall progress is low over a significant length of time.

Inspecting the code in the snapshot browser reveals that the source of the breakdown was a misspelled method name. The student was implementing an interface that required one method. The issue was that the student had misspelled the method name of the interface in the class. This resulted in a failed test case.

The peaks seen in the progress curve, represents the student commenting out significant parts of the program, possibly to try to isolate the problem.

After 50 minutes, it is seen that the student has found the error as the name of the method in the interface is corrected. This resulted in the resolution of the breakdown, and the completion of the exercise.

This particular breakdown is interesting, as one would think that the Eclipse IDE provides aid in circumstances like this. However, the markers accompanying the files suggests that there was no aid being given as to the misspelled method name. However, through further inspection, no errors were found in the logic of the students' code. The problem was that the student had misspelled the name according to the required name in the exercise. This

resulted in the test calling an undefined method, and thus resulting in an error.

The test was requiring a specific name for the method, as defined in the exercise text, however the student had misspelled the name when implementing the interface. The classes implementing the interface were all using the misspelled name, therefore no markers were displayed since the logic of the program was correct.

The results show that the student has spent a significant amount of time trying to solve a problem that is in essence non-existent. The issue was not in the students logic, the issue was in the naming of the required methods in the exercise, causing the program to not meet the required specifications. These errors are impossible for Eclipse to find as the specifications of the exercise are not known to Eclipse.

7.1.3 Incorrectly identified breakdowns

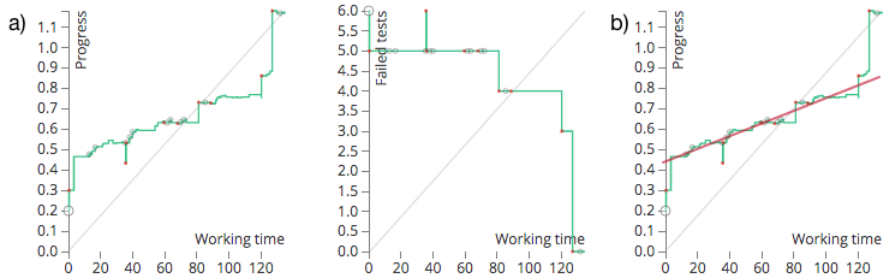

Figure 7.3: Incorrectly identified breakdown, with a linear plot that shows the estimated linear progress of the student

A case where a breakdown was incorrectly identified is shown in figure 7.3. The student has several areas of low progress leading up to an instantaneous progress of 30%. Since the student has performed tests relatively frequently, it suggests that the progress measure is accurate in the time leading up to the sudden increase.

Inspection of the code however, shows the reason for the sudden increase. The student seem to be progressing evenly, although at a relatively low rate compared to other students. However, the reason for the instantaneous progress is that several tests are being completed at the same time. Inspection of the tests reveal that even though they all test different aspect of the program, they are all relying on one specific method to be implemented. The student delays the implementation of this method to the end of the exercise, thus completing several tests at the same time.

The student also starts out with already completed code. The first state of the exercise shows that the student start with 30% progress. This increase in progress at the beginning, coupled with the increase in progress at the end, makes the curve appear less linear. However, by manually adding a regression line that ignores the instantaneous progress, it is

seen that the student does in fact have a linear progression through the exercise. This can be inspected on curve *b* in figure 7.3.

Reasons for the initial progress may be that the student did not enable the data collection plugin until the exercise was already started.

7.1.4 Work flow

The result of the experiment have given a significant insight into the different work flows of the students. The shape of the progress curves and the number of tests run have been seen to give an indication of how the students work.

Manual testing

Inspection of the code at identified breakdowns reveal that some students use the tests as a way to debug when they encounter errors. Instead of performing custom debugging, they are seen repeatedly running tests until it works. This is not an advocated work flow as it implies that the reason for the error is unknown. Tests should be used as a verification of the program, and other types of debugging should be done to locate errors.

Frequency of testing

The results indicate a significant variation in the frequency of running tests. Some students are running tests frequently throughout the exercise, and some are only running tests at the end of the exercise when they believe they are finished.

The advocated work flow should be to run tests on each intermediate milestone in the exercise. The tests should be used as a way of verifying that the current solution is on track to become a fully working solution. By only testing at the end, the students risk that the whole program has to be changed if they misunderstood the exercise text.

Due to the way the data is collected and categorised, it is not possible in the view to inspect supplementary files used by the students when completing the exercises. This makes it difficult to assess whether the students that only test at the end of the exercise use custom debugging and error reports when working.

7.1.5 Experiment with course assistants

The three assistants participating are seen to have very different progress curves. The reasons for the changes, and the relevance to the progress metric will be discussed in this

section.

Difference in total number of tests

The assistants are seen to run a different number of total tests. Assistant *a* only had a total of one test, while assistant *c* had a total of 8 tests.

The number of test are seen to have a significant impact on the progress metric, with participant *a* gaining an instantaneous 60% progress after completing the one test, while participant c only gains 7.5% per completed test.

Through inspection of assistant *c*'s code the explanation for the 8 total tests was seen to be that the assistant had started an optional part of the exercise after finishing the mandatory part. This made the progress metrics for this assistant difficult to directly compare with the others, as the mandatory part is finished after around 25 minutes out of a total 50 minutes spent on the exercise.

The reason for assistant *a* to only run one test is unknown, however a potential reason could be that the assistant was certain that the program was functioning according to the specifications and did not care to run all the tests.

The differences in the total number of tests is seen to be reflected in the accuracy of the progress metric, which is the same result as the student experiment discussed in section 7.1.1.

Not passing all tests

Only one of the assistants passed all the tests, however this was assistant *a* which only had a total of one test to pass.

Assistant *c* was seen to start on an optional part, and inspection of the code suggests that the optional part was not completed. This rendered many of the tests incomplete as modifications to the mandatory part of the exercise was needed to complete the optional part.

Assistant *b* was the only one that ran all tests for the exercise, however on the last state one test was still failing. The reason for this is unknown however it is possible that the last state of the exercise was never saved and therefore not recorded in the system.

By not passing all the tests, the progress metric never reaches the end progress state. This indicates that none of the exercises were finished according to the progress visualised. This was a different result from the student experiment where most participants completed all tests. The most likely difference is the motivation for the exercise, the students are required to complete the exercise according to the specifications, while the assistants did

not have any extrinsic motivation to ensure the correctness of the program.

Differences in frequency of test runs

The assistants are seen to have a varying frequency of test runs. This indicates that the assistants are not following the advocated work flow of running tests after each step is completed.

A reason for this could also be that the assistants find the exercises trivial, and do not see the need to test their program frequently. The time spent on the mandatory exercise is about 30 minutes for all assistants, which may also explain the infrequent testing as the duration over all is short.

The effect of the different frequency of test runs is the seen to be the same as discussed in section 7.1.1, where infrequent testing leads to an inaccurate measure of progress.

Relation to progress metric

The findings of the experiment suggests that the progress metric of more experienced programmers are effected by the same factors as the students. However, due to the variation in frequency and number of tests for each assistant it is difficult to make direct comparisons to the students.

The assumption that the progress curves could be used as a standard to measure students progress does not seem to be true, however the most significant reason for this is due to the inconsistent testing done by the assistants. If the assistants had frequently performed all given tests, the results might have been able to provide a better measure for comparison.

7.1.6 Significance of results

The student experiment had a low sample size compared to the population of students in the course. As seen on figure 5.1, only 34 out of more than 450 student participated. This means the participation rate was only 7.5%. Although a higher participation rate would be preferred, the results still provide significant insight into how it is possible to reason about the progress of students. Due to the low participation rate, it is difficult to use the results to reason about the population in the course, however the methods and techniques show significant potential to be used for these purposes.

7.2 Evaluation of Data Collection & Processing

The system created has been seen to work according to the specifications, and has enabled significant research into how Learning Analytics can be used in courses such as TDT4100.

Due to the low participation rates, there was no need to increase the performance of the system by scaling, however this also suggests that the performance of the system is on par with expectations and that the system can be used for the intended purposes.

7.2.1 Assignment classification

Even though the system collects data according to the specifications. It is seen a problem with classification of files into assignments. The results indicate that students are not following the suggested structure of the exercises, leading to issues with classification of files into exercises.

By further enforcing the project structures of the students the data collection could be improved and the accuracy of the classification would increase.

The most prominent problem seemed to be that students choose to change the package name. A common pattern is to change it to an identification of the current assignment e.g "assignment9". Since the system relies on the package name to identify the exercises, it is difficult to accurately determine their correct classification. Therefore, some files were not correctly classified into an exercise, and were not registered in the application.

7.2.2 Manual testing

The system implemented relies on the students manual test results as the basis of the number of failed tests metric. This was seen to be an issue as it results in inaccurate state metrics when the students do not test frequently. The test results are used to assess how well the program behaves according to the exercise specification, and is thus a measure of how near completion the program is. Therefore relying on student tests, means that snapshots where tests are not run might not show an accurate representation of the total progress in the exercise.

This is a problem as it makes the progress curves dependent on the work flow of the student. It also makes it more difficult to spot breakdowns, as false positives may be found if a student does not test frequently.

Another problem with this approach is that the number of tests is not constant for all students. The results indicate that many students do not run all tests given for each exercise, this makes the total number of test count differ between students in each exercise.

As discussed in section 7.1.1; increasing the number of tests run is beneficial, as it results in a more linear progress curve, as more milestones are recorded. This means that the progress for students not running all given tests are not as accurate as it could be.

7.2.3 Collected information

The information collected was on par with the specifications presented. All .java files, including the relevant markers, as well as test runs were stored. The storage solution allowed for complete reproduction of all events coming in to the system, which enabled a very flexible solution that made it possible to re-create the metric database when new metrics were required.

Even though the data collection was implemented according to specifications, it is seen that more information would be beneficial. Information regarding idle time activities, such as if the application focus is still on Eclipse, or if there is still mouse movement or other interactions. This information would allow more detailed representation of idle time, and allow researchers to distinguish between a student spending time trying to solve a problem, and a student having a break.

7.2.4 Frequency of collection

The frequency of collection was sufficient for the analysis that was done, however during breakdown inspection it was noted that some students would have long intervals between saves. This resulted in several changes being made from one state to the next, making it more difficult to follow the work flow of the student.

This was not an issue in this project, but more detailed analysis of breakdowns could benefit from a higher frequency of data collection to ensure a more detailed event log.

7.3 Evaluation of Data Analysis and Exploration

Even though the specifications of the data analysis and exploration part of the system were vague due to the experimentation aspect of the project, the resulting application was successful. The aim was to make a flexible system that could create and compare expression and allow for further extension.

The demonstration in section 5 clearly shows that the system accommodated to the experimentation, and resulted in several extensions to the application to allow more detailed exploration.

Chapter 8

Conclusion

8.1 Research questions

Q1: How can data collected from students working through programming exercises be used to reason about their *progress* through the exercises?

A metric for the current progress was found and evaluated. The results indicate that the initial hypothesis were correct, however further concretization of the metric was required to arrive at an adequate expression.

It is indicated that the students progress through exercises can be seen through a combination of the failed tests relative to the total tests, and the line count relative to the line count in the exercise solution.

The results are however inconclusive due to the low participation rate (7.5%) as well as the wide spread in participants on each exercise, leaving 13 students as the highest number of participants in the same exercise (see appendix B).

The circumstances of the data collection also impacted the value of the results. The inaccuracy of the failed tests metric made the progress for some students inaccurate, due to the metric relying on students manual testing. Also the difficulty in classification of the files made some students assignments unable to be classified, resulting in less amounts of data.

Therefore the results can only be used as an indication and starting point for further research, however the results show significant potential.

Q2: How can data collected from students working through programming exercises be used to identify *breakdowns*?

The results indicate that the progress curves can be used to identify breakdowns. Breakdowns can be identified by examining areas of significant lack of progress over time.

The results also suggest that there are some false positives using this technique. The reasons for this seems to result from the lack of automatic testing, and the resulting inaccuracy of the progress metric. This is especially problematic for breakdown identification as passing test are seen as significant progress in the exercises. By delaying the inclusion of passing tests, the progress is reported as lower than reality, resulting in the visualisation of breakdowns that are non-existent.

Overall, there is a significant indication that progress curves can be used to identify breakdowns in exercises, and these can be further inspected to gain detailed knowledge about the challenges of students. However, no conclusive statements can be made due to the low sample sizes as reported in the previous section. There is however a strong indication that this is a possible method of identifying breakdowns.

8.2 Implemented system

The system implemented has served its purpose . The functionality was satisfactory according to the initial specifications, and research was conducted as a result of the system. The experimentation and comparison of expression based on collected metrics was satisfactory and allowed for easy extension and iterations of improvement.

The results indicate some areas that could be improved, however these improvement suggestions came as a result of the usage of the application, and can be seen as the next iteration in a larger design science approach.

The architecture of the application was created to make it flexible and allow easy scaling and extensions. Even though there was no need to scale the system for this project, implementing replication of the services should be straight forward. Extending the application was done several times through the various iterations of development and evaluation, and demonstrated the flexibility of the application.

8.3 Suggestions for future work

The results from the experiment were only able to provide an indication of the validity of the hypothesis due to the small sample size. Future work should focus on performing a similar experiment with a higher and more varied sample of students. Future experiments should also validate the data through closer interaction with students.

Future work could also look into automatic identification of breakdowns based on the analysis data, to provide potential automatic guidance to students.

Further utilization of the collected markers would be beneficial. A closer look into the errors of the students to get an overview of the reasons for breakdowns be done. This information could be used to find common challenges for the students, both on an individual level and collectively. Further uses of this information is to identify exercises that are particularly challenging, or contain tricky parts.

Further metrics

Collecting further metrics from the code could provide an even greater overview of the work flow of the students. By examining metrics of the code unrelated to progress, such as metrics for code quality, further expressions could be created that complements the progress expression to provide more detail.

This could for example give further insight into reasons for a decrease in progress by indicating if the decrease was accompanied by an increase in the code quality. This would suggest that the student merely re-factored parts of the code.

Bibliography

[1] S. K. Sørhus, "Aiding novice programmers by extending Eclipse," Dec. 2014.

[2] G. Siemens, "Learning Analytics The Emergence of a Discipline," *American Behavioral Scientist*, vol. 57, no. 10, pp. 1380–1400, Oct. 2013, 00036. [Online]. Available: http://abs.sagepub.com/content/57/10/1380

[3] G. Siemens and R. S. J. d. Baker, "Learning Analytics and Educational Data Mining: Towards Communication and Collaboration," in *Proceedings of the 2Nd International Conference on Learning Analytics and Knowledge*, ser. LAK '12. New York, NY, USA: ACM, 2012, pp. 252–254, 00128. [Online]. Available: http://doi.acm.org/10.1145/2330601.2330661

[4] C. Piech, M. Sahami, D. Koller, S. Cooper, and P. Blikstein, "Modeling How Students Learn to Program," in *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education*, ser. SIGCSE '12. New York, NY, USA: ACM, 2012, pp. 153–160, 00053. [Online]. Available: http://doi.acm.org/10.1145/2157136.2157182

[5] P. Blikstein, "Using Learning Analytics to Assess Students' Behavior in Open-ended Programming Tasks," in *Proceedings of the 1st International Conference on Learning Analytics and Knowledge*, ser. LAK '11. New York, NY, USA: ACM, 2011, pp. 110–116, 00070. [Online]. Available: http://doi.acm.org/10.1145/2090116.2090132

[6] J. Spacco, D. Hovemeyer, W. Pugh, F. Emad, J. K. Hollingsworth, and N. Padua-Perez, "Experiences with Marmoset: Designing and Using an Advanced Submission and Testing System for Programming Courses," in *Proceedings of the 11th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education*, ser. ITICSE '06. New York, NY, USA: ACM, 2006, pp. 13–17, 00067. [Online]. Available: http://doi.acm.org/10.1145/1140124.1140131

[7] J. Spacco, J. Strecker, D. Hovemeyer, and W. Pugh, "Software Repository Mining with Marmoset: An Automated Programming Project Snapshot and Testing System," in *Proceedings of the 2005 International Workshop on Mining Software Repositories*, ser. MSR '05. New York, NY, USA: ACM, 2005, pp. 1–5, 00044. [Online]. Available: http://doi.acm.org/10.1145/1082983.1083149

[8] K. Heinonen, K. Hirvikoski, M. Luukkainen, and A. Vihavainen, "Using CodeBrowser to Seek Differences Between Novice Programmers," in *Proceedings of the 45th ACM Technical Symposium on Computer Science Education*, ser. SIGCSE '14. New York, NY, USA: ACM, 2014, pp. 229–234, 00004. [Online]. Available: http://doi.acm.org/10.1145/2538862.2538981

[9] M. C. Jadud, "Methods and Tools for Exploring Novice Compilation Behaviour," in *Proceedings of the Second International Workshop on Computing Education Research*, ser. ICER '06. New York, NY, USA: ACM, 2006, pp. 73–84, 00072. [Online]. Available: http://doi.acm.org/10.1145/1151588.1151600

[10] E. Balzuweit and J. Spacco, "SnapViz: Visualizing Programming Assignment Snapshots," in *Proceedings of the 18th ACM Conference on Innovation and Technology in Computer Science Education*, ser. ITiCSE '13. New York, NY, USA: ACM, 2013, pp. 350–350, 00002. [Online]. Available: http://doi.acm.org/10.1145/2462476.2465615

[11] S. T. March and G. F. Smith, "Design and natural science research on information technology," *Decision Support Systems*, vol. 15, no. 4, pp. 251–266, Dec. 1995, 02444. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0167923694000412

[12] K. Peffers, T. Tuunanen, C. E. Gengler, M. Rossi, W. Hui, V. Virtanen, and J. Bragge, "The design science research process: a model for producing and presenting information systems research," 2006, 00231.

[13] "Microservices," http://martinfowler.com/articles/microservices.html, accessed: 2015-06-06.

[14] "Mqtt," http://mqtt.org/, accessed: 2015-06-06.

[15] "Event sourcing pattern," https://msdn.microsoft.com/en-us/library/dn589792.aspx, accessed: 2015-06-06.

[16] R. Wieringa, "Design Science As Nested Problem Solving," in *Proceedings of the 4th International Conference on Design Science Research in Information Systems and Technology*, ser. DESRIST '09. New York, NY, USA: ACM, 2009, pp. 8:1–8:12, 00141. [Online]. Available: http://doi.acm.org/10.1145/1555619.1555630

[17] R. M. Ryan and E. L. Deci, "Self-determination theory and the facilitation of intrinsic motivation, social development, and well-being," *American Psychologist*, vol. 55, no. 1, pp. 68–78, 2000, 13429.

[18] "Endomondo personal training application," https://www.endomondo.com/, accessed: 2015-06-06.

[19] S. Slade and P. Prinsloo, "Learning Analytics Ethical Issues and Dilemmas," *American Behavioral Scientist*, vol. 57, no. 10, pp. 1510–1529, Oct. 2013, 00050. [Online]. Available: http://abs.sagepub.com/content/57/10/1510

[20] A. Pardo and G. Siemens, “Ethical and privacy principles for learning analytics,” *British Journal of Educational Technology*, vol. 45, no. 3, pp. 438–450, May 2014, 00012. [Online]. Available: http://onlinelibrary.wiley.com/doi/10.1111/bjet.12152/abstract

Appendix A

System Implementation

A.1 Architecture Overview

The overall architecture of the application follows the micro services architecture. The application is partitioned into small self-sustaining services that communicate to form the complete application. This allows for a more flexible and easily expandable application as services can be extended or changed with little effort. The architecture also offers increased scalability through designing services in a way that enables several instances of performance critical modules. This means that the application can be scaled according to demand simply by running more instances of the services where performance must be increased.

A.1.1 Application Services

All services were written in JavaScript and are ran with Node.js. This solution was chosen due to JavaScripts dynamic typing and functional aspects that results in rapid development and flexibility for change. Node.js was chosen for the server application due to its asynchronous nature, making it easy to accommodate the variable responss times needed due to I/O actions. Express.js was used for defining the API due to the simple and flexible solution it provided.

A.1.2 LA Helper - Client

The Eclipse plugin is written in Java as a standard Eclipse plugin and is downloaded from a plugin repository site through a plugin interface in Eclipse.

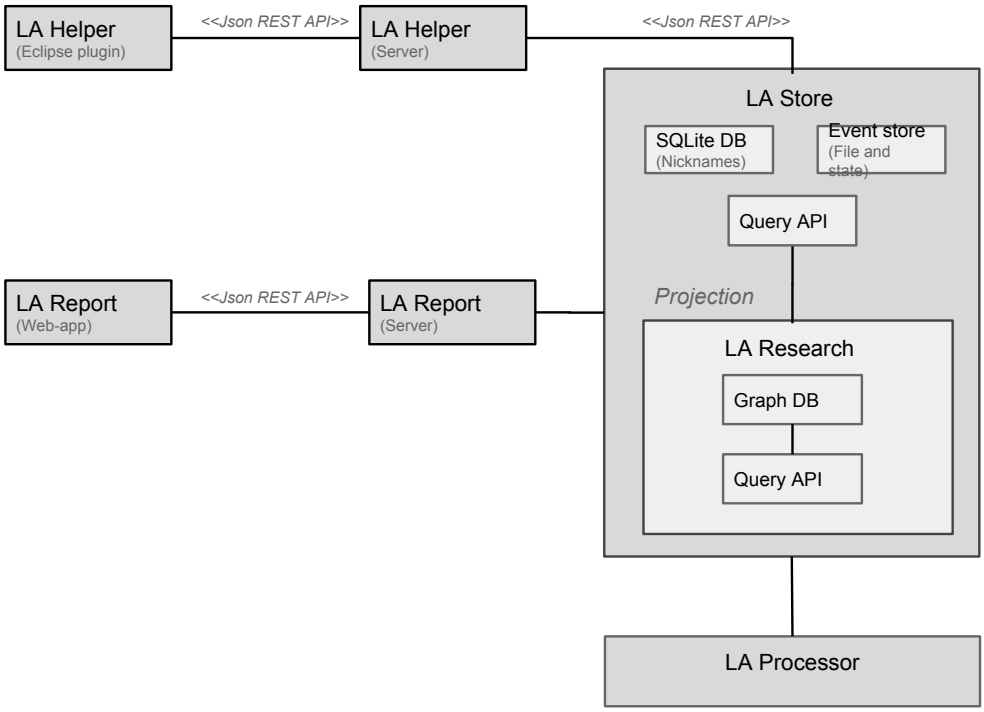

Figure A.1: Overview of the artifact architecture

The core functionality of the plugin is to collect data. The plugin listens for file changes on the file system, ie. when the user saves. Whenever a file is changed it is checked if it is eligible for collection. The user can specify in the settings page which directories in the project to enable data collection. All .java files in the specified directories will be monitored.

When a change happens to an eligible file the contents of the file is logged together with all the markers that Eclipse has associated with the file. These include both error markers and information markers. The marker information is serialized to a meta file and both files are transfered to the server for storage.

The plugin also sends information about the change that occurred, detailing if the change was in file content, file creation, moving or deleting.

The plugin subscribes as a listener to jUnit which enables it to log the results of tests that are run by the user. Whenever a test is performed, the result is immediately transfered to the server in a .tests file, to signify the running of the test.

In the event of an error exception firing in the plugin the error message is recorded and sent

to the server. This is done to monitor for faults in the plugin partly due to the anonymous collection which makes bug reporting problematic, but also to ensure bug reports.

The plugin checks the server if a new disclaimer is present based on the timejtamp of the latest downloaded disclaimer. If a new disclaimer is found, logging is stopped and the disclaimer is presented to the user for acceptance or denial.

Retrieving the disclaimer from the server allows for changes in the disclaimer without requiring the participants to download a new version of the plugin. This makes it easier to ensure that ethics are followed and that any updated to the disclaimer is presented to the user, who can chose to not agree to the new terms.

On the first install of the plugin a unique client id will be created for the user. The plugin requests a new user from the server, and is given a unique id. This is the only identification used when transferring collected files.

A.1.3 LA Helper Server

LA Helper Server is a server-side service that handles the data coming from the Eclipse plugin. It can be seen as an entry-point for the data collection and processing. The service communicates with the storage service to enable data collection, as well as setting nicknames and creating new users.

The communication with the service is done through an API relying on JSON data structures. For future scalability uses, this service could be a broker between multiple storage service instances to decrease the performance requirement of each storage service.

The service accepts a list of files together with the unique id of the user for data collection. It also has API points for setting nicknames, updating participation status and error logging.

The disclaimer popup text is also handled by the service. This allows the users to be notified when the disclaimer is changed.

A.1.4 Storage Service

The storage service stores the collected data in an event store, and uses projections to allow storage of processed information from the store.

Event Store
The event store uses the Git version control system. Git enables fast and easy storage of several versions of a file. It is space efficient and stores only the differences between each revision of a file. This reduces the space requirement while keeping a simple interface to extract different versions.

Git is used due to the space efficiency and due to previous experience with the software. Since the file snapshots collected often shares a significant amount of information with the previous snapshot, only storing the difference is a significant storage benefit.

When a new participant is registered, a new folder is created with an empty git repository initialised. Whenever a file is received for collection, the file is saved in the directory in the location it had in the project path on the client. This ensures that each time the same file is collected it will overwrite the previous snapshot and always keep an up to date representation of the state of the file.

When a collection request is saved all files are added to the git repository and an automatic commit is performed. This ensures that marker information is added in the same commit as the file changes.

A.1.5 Processing Service

The processing service handles extracting metrics from the collected data. It is separated from the store to allow for performance demanding operations on the data. The service exposes an API that can be used to send data for processing.

A.1.6 Analysis Projection

The first projection from the event store to be made was the analysis projection. The aim of this projection was to have a flexible representation of the metrics that were relevant for analysis. After files had been processed the resulting metrics were stored in this projection.

Due to the relational structure of the data set, and the flexibility requirement, a graph database was used to store the metrics. The graph database used was Neo4j which is schema less, meaning that additional fields and relations could easily be added on demand

The benefit of this was apparent already after one iteration of design and implementation, as it was seen that the complexity of the data structure could be reduced by removing some relations.

Another benefit was the querying language used which allowed for a simple and declarative interface to extract wanted data from the database. This made it easy to examine new ways of querying the data set.

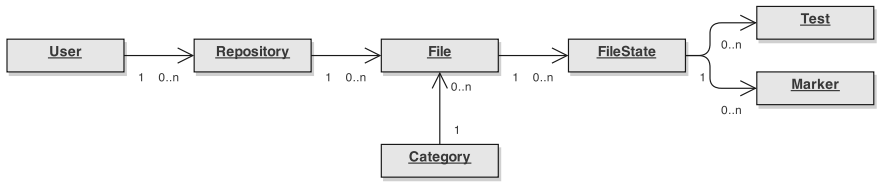

Figure A.2: Relational structure of the data

A.2 Analysis and Experimentation

The analysis and experimentation part of the application was developed with a client server pattern. It consists of a browser based client that requests the collected metrics information from the server. The client is used to visualise the collected data. The client also allows experimentation with different combinations of metrics used in the visualisations.

The client was created in Javascript and HTML5, with React.js as the framework for rendering the user interface. React was chosen due to the de-coupling with the model and the ability to create re-usable components.

The graphs were visualised using d3.js, as it enables a very flexible library for visualising information. This allowed completely customised visualisations that included several different data sources.

Appendix B

Experiment Details

Figure B.1: Number of participants that have done each exercise

Assignment	Type	Participants
Card	Objektstrukturer	13
Train	Arv	13
Logger	Delegering	13
StockListener	Observatør-teknikken	12
CardComparison	Interface	12
Named	Interface	11
CardContainer	Interface	10
StringGrid	Interface	9
SavingsAccount	Arv	8
CardContainerImpl	Arv	7
Twitter	Objektstrukturer	7
Partner	Objektstrukturer	7
LineEditor	Tilstand og oppførsel	5
Account	Innkapsling	5
Person	Objektstrukturer	5
Person	Innkapsling	5
Sortering av TwitterAccount	Interface	4
UpOrDownCounter	Tilstand og oppførsel	4
TicTacToe	Innkapsling	3
Asteroids	Arv	3
TheOffice	Delegering	3
Nim	Innkapsling	3
Rectangle	Tilstand og oppførsel	3
Account	Tilstand og oppførsel	2
RPN-Kalkulator	Innkapsling	2
HighscoreList	Observatør-teknikken	2
Location	Tilstand og oppførsel	1
Kalkulator	Arv	1
Sudoku	Innkapsling	1

Appendix C

Source Code

The source code is divided into 5 git repositories. One for each of the services in the system, as well as one for the Eclipse plugin. The tag used for the current state of the project is v.0.3.0.

The source code can be found on the following urls:

- LAHelper Plugin: https://github.com/steinso/LAHelperPlugin
- LAHelper Server: https://github.com/steinso/LAHelperServer
- LAStore: https://github.com/steinso/LAStore
- LAProcessor: https://github.com/steinso/LAProcessor
- LAReport: https://github.com/steinso/LAReport