



Norwegian University of
Science and Technology

Gamifying TDT4100

Syver Bolstad

Master of Science in Informatics

Submission date: June 2018

Supervisor: Hallvard Trætteberg, IDI

Norwegian University of Science and Technology
Department of Computer Science

Summary

There have been major advancements in information technology lately, and therefore an ever growing need for good programmers. This leads to more programming students, and an increasing pressure on the educators. The course TDT4100 Object-oriented programming at Norwegian University of Science and Technology(NTNU) makes use of teaching assistants who read the students' code and provide feedback on code quality, which is time consuming for both students and teaching assistants. Advances in information technology have also led to an increased interest in computer games and introduction of the term gamification, which describes the use of elements from games in non-game contexts. This study addresses the development of a prototype of a learning system that through gamification attempts to supplement the coursework of Object-oriented programming, and explores the benefits introduced by the prototype. The learning system has an additional focus on the wide range of programming skills relevant to the course.

The first part of the study provides a detailed introduction to the term gamification, where both positive and negative sides are explored. In addition, a small introduction to code quality is provided, and how code quality information can be retrieved from the code automatically. These findings are then adapted to Object-oriented programming, and are used together with the feedback from iterative user test as the basis for the development of the prototype.

The next part of the study handles simulating the usage of the system using previously delivered code. The results of the simulation show that the prototype is not adequately customized and that it has many possible improvements, but also introduces some benefits to the exercise plan. The results also show that the simulation in itself is useful both for customizing the award model of the prototype, and providing relevant information about the coursework to the lecturer.

Sammendrag

Det har blitt gjort store fremskritt i informasjonsteknologien den siste tiden, og det er derfor et stadig voksende behov for gode programmerere. Dette fører til flere programmeringsstudenter, og et økende press på utdanningspersonellet. Faget objektorientert programmering ved NTNU benytter seg av studentassistenter som leser gjennom koden til studentene og gir tilbakemelding på kodekvalitet, noe som er tidkrevende både for studenten og studentassistentene. Fremskrittene i informasjonsteknologien har også ført til en økt interesse for dataspill og introduksjonen av begrepet gamification, som beskriver bruken av elementer fra spill i andre sammenhenger. Dette studiet tar for seg utviklingen av en prototype av et læringssystem. Som ved bruk av gamification forsøker å supplere øvingsopplegget til objektorientert programmering, og utforsker hvilke fordeler som blir introdusert av prototypen. Prototypen har et ekstra fokus på det store spekteret av programmeringsferdigheter som er relevant for faget.

Den første delen av studiet gir en detaljert utredning av begrepet gamification, hvor både positive og negative sider blir utforsket. I tillegg gis en liten innføring i kodekvalitet, og hvordan informasjon om kodekvalitet kan hentes ut fra koden automatisk. Disse funnene blir deretter tilpasset objektorientert programmering, og de blir sammen med tilbakemeldingene fra en iterativ brukertesting benyttet som grunnlag for utviklingen av prototypen.

Den neste delen av studiet går ut på å simulere bruken av systemet, ved hjelp av tidligere innlevert kode. Resultatene fra simuleringen viser at prototypen ikke er tilpasset tilstrekkelig og har flere muligheter for forbedring, men også at den introduserer flere fordeler til øvingsopplegget. Resultatene viser også at simuleringen i seg selv, er til hjelp både for å tilpasse prototypen, men også for å gi nyttig informasjon om øvingsopplegget til foreleser.

Preface

This thesis was completed throughout the Fall of 2017 and Spring of 2018, with the deadline set at June 1, 2018. The report was written by Syver Bolstad. Because the application is a prototype of a gamification and learning aid in the course TDT4100, the target audience will mainly be students, student assistants, or teachers involved or participating in TDT4100. Although the application is rather specified, some topics in the thesis may also be relevant for people interested in learning technology, gamification, or education in programming.

Table of Contents

Summary	i
Preface	iii
Table of Contents	vii
List of Tables	ix
List of Figures	xii
1 Introduction	1
1.1 Motivation	1
1.2 Problem Description	3
1.3 Goal	4
1.3.1 Objectives	4
2 Research methods	5
2.1 Literature Review	5
2.2 Prototype	6
2.3 Implementation	6
2.4 Validating prototype using student data	6
3 Literature Review	9
3.1 Gamification	9
3.2 Motivation and Rewards	10
3.2.1 Redeemable points	10
3.2.2 Where’s Wally	11
3.2.3 Badges	11
3.2.4 Team leaderboards	12
3.3 Code quality	12
3.3.1 ISO/IEC 25010	12

3.3.2	SOLID	13
3.4	Code Metrics	14
3.4.1	Existing tools	14
3.4.2	Specific metrics	15
3.5	Case study - Khan Academy	17
4	Gamifying TDT4100	19
4.1	TDT4100 - Object oriented programming	19
4.1.1	Coursework	20
4.1.2	Relevant metrics	20
4.2	Badges	21
4.3	Reward model	22
4.4	Simulation as method	24
4.5	Validating the use of gamification	24
4.5.1	Badge award simulation	24
4.5.2	Overall Metrics Accumulated	26
4.5.3	Metrics sorted by task difficulty	28
4.6	Tailoring metrics for assignments	30
4.6.1	Method for configuring relevant metrics	30
5	Design	33
5.1	Rapid Prototyping	33
5.2	Tools	33
5.3	Iterations	34
5.4	Testing	35
5.4.1	Usability testing	35
5.4.2	Survey	36
5.4.3	Final changes	37
5.5	Requirements	39
5.5.1	Backend	39
5.5.2	Frontend	39
5.6	High fidelity prototype	40
5.6.1	History page	40
5.6.2	Badge page	42
6	Evaluation	45
6.1	Research Methodology evaluation	45
6.2	Use of Gamification	45
6.3	Choice of Metrics	46
6.4	Testing	47
6.5	Evaluating the results	47
6.6	Validity	48
6.6.1	Method validity	48
6.6.2	Validity of the simulation	48
6.7	Fulfillment of the project goal	48

7	Conclusion	49
8	Future development	51
8.1	Backend	51
8.1.1	Supplementing the badges	51
8.1.2	Improving the method for configuring badge requirements	51
8.1.3	Student task recommendation	52
8.1.4	New applications of the simulation data	52
8.2	Frontend	52
8.2.1	Imrpovements to the current design	52
8.2.2	Leaderboards	53
8.2.3	Additional views	53
	Bibliography	55
	Appendix	59
8.3	Appendix A - Test: Survey	60
8.4	Appendix B - Backend printed	65
8.4.1	Change local path	65
8.4.2	Code run for the results to be printed	66
8.4.3	Student metrics print	67
8.4.4	Solution metrics print	68
8.4.5	Student badge print	69
8.4.6	Cyclomatic Complexity for the proposed solution	71
8.4.7	Unused Metrics for the proposed solution	72
8.4.8	Relevant Metrics for each task	73
8.5	Appendix C - Backend Implementation	74
8.5.1	Git	74
8.5.2	Technologies	74
8.5.3	Overview	75
8.5.4	Metrics	77
8.6	Appendix D - Class diagrams	80
8.6.1	Main.java Class diagram	80
8.7	Appendix E - Iterations	81
8.7.1	Iteration 1	81
8.7.2	Iteration 2	81
8.7.3	Iteration 3	82
8.8	Appendix F - Results	83
8.8.1	Proposed solution metrics extraction	83
8.8.2	Badges with corresponding sub-metrics	84
8.9	Appendix G - Frontend development tools	85
8.9.1	Knockout	85
8.9.2	SweetAlert	86
8.10	Appendix H - Suggested information for the alternative views	87
8.10.1	Information for the lecturer view	87
8.10.2	Information for the teaching assistant view	88

List of Tables

3.1	Levels of game elements [43]	10
3.2	ISO/IEC 25010 Metrics[15]	13
3.3	CKJM Metrics	15
4.1	Metric Categories	21
4.2	Badge requirement levels	23
4.3	Unused metrics	28
4.4	Submetrics used to filter relevant use	31
4.5	Relevant metrics for the sub-tasks of assignment 5	31
5.1	List of backend requirements	39
5.2	List of frontend requirements	40
8.1	Programming curriculum handled by which metric	77
8.2	GeneralMetrics' Sub-Metrics	78

List of Figures

2.1	Different parts of the Master Thesis	5
3.1	Code snippet and Flow graph	16
4.1	Badge valor's	22
4.2	All badges	22
4.3	Planned reward model	23
4.4	Badges awarded each student	25
4.5	Badges awarded for each metric	26
4.6	Overall metrics diagram	27
4.7	Metric results sorted by difficulty	29
4.8	Average Cyclomatic Complexity	30
4.9	Tasks where the iteration metric is relevant	32
5.1	Prototyping tools used	34
5.2	Iterations	34
5.3	Pie chart of the results of questions eight, nine and ten	36
5.4	Cake diagram of the results from question 11	37
5.5	WebView to Web	38
5.6	Button visibility	38
5.7	History page	41
5.8	Detailed metrics page	42
5.9	Button visibility	43
5.10	Detailed badge information and next badge info	43
8.1	Example of a parse tree	74
8.2	Class Overview	75
8.3	JDT-model	76
8.4	Exercise model	76
8.5	Class Overview	78
8.6	View of Overall metrics in the frontend	79

8.7	Wireframe iteration one	81
8.8	Wireframe iteration two	82
8.9	Wireframe iteration three	83
8.10	Proposed Solution metrics extraction	84
8.11	Knockout MVVM[4]	86
8.12	alert() vs SweetAlert	86
8.13	Next alert/notify	87
8.14	Lecturers view	87
8.15	Lecturers detailed view	88
8.16	Teaching assistant view	88
8.17	Teaching assistant detailed view	88

Chapter 1

Introduction

The following chapter describes information relevant before reviewing the rest of the report. It presents the motivation behind the thesis, a detailed problem description as well as the target audience of the product. Lastly it gives an outline of the rest of the thesis.

1.1 Motivation

Computer Science has a huge impact in modern society[40]. Digitalization is changing many of the world industries[18], and is therefore an increasing demand of programmers[27]. To keep up with the demand, and make the digitalization process as efficient as possible. It is important to educate even more computer science students.

In order to educate a larger number of students and teach them good code practices without bursting the workload of the teachers, new advancements in education may be used. Gamification is the use of game elements in non-gaming scenarios[5], and in the learning scenario it introduces many advantages. When using the correct elements from games, Gamification increases motivation[50], and can also increase the learning outcome by introducing active learning[49].

Good code practices are also important. Some developers spend 90% of their time reading code other people have written[32]. Therefore important to write straight forward code that's easier for others to understand. To help the student write good and understandable code, the student assistants must read the student code and inform the student of mistakes in the code, or bad code practices. This is time consuming both for the student and student assistant, and is normally done only once for each assignment.

In this thesis, a combination of both gamification and code information retrieval is used to aid students in the course TDT4100. The solution aims to help motivate students using a reward model, giving rewards based on information about code quality retrieved from the

code. The solution will be tested on previously delivered student code, and the results will hopefully help determine whether the solution is a sufficient aid for students and teachers.

1.2 Problem Description

The problem description was suggested by the supervisor Hallvard Trøttestad as part of the Master program in *Interaction Design, Game and Learning Technology* at Norway's Technical and Natural Science University (NTNU).

Learning analytics are techniques for collecting and analyzing data about the learning process, so that one can provide better learning support. We are working on taking this into TDT4100 and other programming subjects. The project is about studying, and how different code-quality goals can be used to identify the maturity level of students and what type of learning support they need.

The problem description was later redefined to a project goal, explained in the following section.

1.3 Goal

This section presents the main goal of the thesis. The goal was formulated with help from Hallvard Trætteberg.

Create a prototype of a gamified learning system supplementing the coursework in the course Object Oriented Programming (TDT4100). Focusing on the wide span of programming skills significant for the course curriculum.

The prototype will in this setting be evaluated as a more of a conceptual design. It should however, be comprehensive enough to validate the design.

1.3.1 Objectives

Because of the extensive and relatively open scope of the goal, the goal has been split into a group of objectives to try and clarify the different elements needed to complete the goal.

Obj1: Investigate what elements of gamification are relevant for the coursework

Obj2: Use these element to create conceptual prototype

Obj3: Evaluate the prototype

Obj4: Display the gamified coursework for the end user in a understandable manner

Obj5: Help the lecturer maintain and adjust the system over time

Research methods

To complete the objectives of the prototype, several methods were used. Figure 2.1 shows an overview of the main parts of the project.

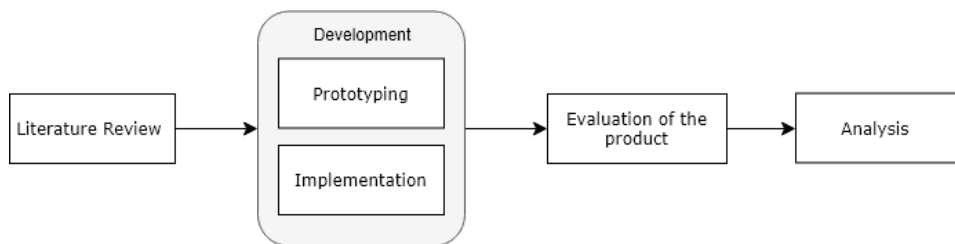


Figure 2.1: Different parts of the Master Thesis

Figure 2.1 above shows the different stages of the process of developing the prototype. Starting with the literature review and development of the product, followed by the prototype evaluation and analysis. This chapter will explain the different methods used throughout the project.

2.1 Literature Review

This stage is comprised of an examination of research literature about Gamification, Motivation and Rewards, Code quality and coding Metrics. In order to find relevant readings, both Google scholar and online articles found in the NTNU University Library was used. To get a overview of the general consensus on the topic, a number of articles were skimmed before deciding on which to read in more depth. When deciding on which articles to read, the articles with a high number of citations and low number of discrepancies between articles were favored. On some of the topics, the recently published articles were chosen over older ones because of the recent progress made in some of the topics. The review

started in general topics, and gradually became more and more specific. For instance, the articles about gamification were read first followed by motivation and rewards within gamification, and eventually gamification in education and case studies of Khan Academy. These readings were the basis when deciding on both the reward model chosen, and which software metrics to use in the product.

2.2 Prototype

The prototype was developed iteratively where Hallvard Trættemberg and several students were used as test subjects to represent the end user. This was useful to get feedback on what gamification elements were relevant (Obj1) and how to display them in a understandable manner (Obj4). *Throwaway prototyping*[44] was used in the development. A low level prototype was created and iteratively changed based on feedback from Trættemberg and the students. After the main design was decided upon, a more formal qualitative test was conducted where the students completed a survey and completed a usability test.

The qualitative method in empirical studies described by Carolyn B. Seaman[42] was used when conducting a qualitative test of the product. The test participants were observed while taking the test while they completed tasks on the prototype. After the test a survey comprised of a mix of open and closed questions, was completed by the test participants as an alternative to the interview phase described by Seaman[42]. After taking the feedback from the survey into account, the view from the prototype was rebuilt in HTML. The prototype development and testing is further described in Chapter chapter 5.

2.3 Implementation

The implementation was split into two parts. The frontend development in HTML and the backend development in Java, which together constitutes the prototype(Obj2). Both parts used an agile approach that included frequent meetings with Trættemberg. The method used in the frontend development part was similar to the MoSCoW method developed by Dai Clegg[20], where the requirements in the backlog were sorted in groups of must have, should have, and could have. The method was chosen to make sure only the most important requirements were completed so there would be enough time for the Java development.

In the backend development phase, planning requirements in detail was difficult because of the uncertainty in what to make and lack of knowledge of the tools used. The open requirements were split into smaller tasks, and new tasks that were continuously discovered as the work was done.

2.4 Validating prototype using student data

In the data gathering phase, a simple version of the *discrete event simulation model* described by Averill M. Law and David Kelton[29] was used. In short, the model uses

information about the system state and acquires statistical data as the state evolves over time. A code base consisting of assignments from last year's students was used in the simulation.

To retrieve useful information about the prototype, the statistical data from the simulation model was evaluated to explore how the prototype operated. This data was later used to evaluate the prototype(Obj3). A more detailed description of the data simulation process is found in Chapter 4.

Literature Review

This chapter addresses some literature that is needed to understand the choices made in developing the solution. The first part gives an introduction to gamification, motivation, and rewards, followed by a description of code quality measures, metrics, and existing tools. Finally, the learning platform Khan Academy and some associated studies are presented.

3.1 Gamification

This section gives a detailed description of the term Gamification. Gamification, defined by Alessandra Antonaci and Specht as,

"The application of game design elements in non-gaming scenarios to solve problems or to influence a user's behaviour change"[6]

and by Sebastian Deterding and Nacke as,

"The use of design elements characteristic for games in non-game contexts" [43]

can take many forms and is used in many different areas. In order to grasp a clearer understanding of the definitions, clarification of what is meant by "games" and "game elements" is necessary. James Juul defines games as *"a rule-based formal system with a variable and quantifiable outcome, the player exerts effort in order to influence the outcome"*[22]. In order to define the game elements to encourage the definition by Juules, Sebastian Deterding and Nacke[43] treats the game elements as a set of building blocks based on semi-abstracted elements found in most games. The table below shows the different elements and examples of these elements as described by Sebastian Deterding and Nacke[43].

Level blocks	Example
Game interface design patterns	Badges, leaderboards, or levels
Game design patterns and mechanics	Time constraint, limited resources, or turns
Game design principles and heuristics	Enduring play, clear goals, variety of game styles
Game model	Challenge, fantasy, curiosity, game design
Game design methods	Playtesting, playcentric design, value conscious game design

Table 3.1: Levels of game elements [43]

These game elements may be used in full or partly in order to apply gamification on non-game contexts. A study by TODD[47] found that implementing gamification on a micro-scale may result in the game being perceived as superficial, which would render the use of gamification ineffective.

3.2 Motivation and Rewards

Player motivation is important when attempting to successfully build a gamified system. Motivation is divided into two different categories, intrinsic and extrinsic motivation described by Harakiewicz. Intrinsic motivation describes the motivation of performing an activity for its own sake instead of as a means to an end. Therefore intrinsically motivated behavior occurs independently of reinforcements or rewards. Extrinsically motivated behaviour describes the opposite, where the source of the motivation is external to the person[19].

Reinforcement studies has revealed how expected reward motivates player action. It shows that the use of rewards can increase player motivation[50][28]. However, the increase in rewards may come at a cost. A study by E. L. Deci and Ryan reveals that extrinsic rewards undermine the intrinsic motivation in an educational context[14].

The article *Exploring Engaging Gamification Mechanics in Massive Online Open Courses*[11] investigates which game mechanics are the most engaging when adapted to MOOCS. According to the study, *Virtual goods* was the most engaging mechanic. Virtual goods are digital rewards that are rewarded to players who have accomplished certain feats in the game. In the book *Gamification by Design*[50], virtual goods is defined as badges, trophies, points, coupons, discounts, or early access to premium content and features. There are many different approaches when handing out digital rewards. the article by Chang and Wei[11] introduces several possibilities introduced in the following sections.

3.2.1 Redeemable points

Redeemable points introduces a point system where points are awarded to students that play the game. These points can then be redeemed in either real world rewards or to unlock

new content in the game[11]. In the study by Chang and Wei[11] redeemable points were found to be the second most motivating reward.

3.2.2 Where's Wally

Another reward introduced by Zichermann and Cunningham[50] is "Where's Wally". Where's Wally is a reward that takes advantage of the natural problem solving ability of people. Wordless figures are used in the learning process, feedback is given based on the student's level, and different cues are used if the student is stuck[11].

3.2.3 Badges

Badges are powerful rewards that are used to motivate people in many different areas. The Google translate community, Khan Academy, and the Army, are some of the many examples where badges are used. Badges can give users enjoyment when they unexpectedly get a badge, and drive users who enjoy collecting them[50]. Badges are also a good way to reveal the completion of a goal and the overall progress in the game[50].

Research suggests that boring and pointless badges may render the other badges vapid, described as *Badgenfreude* in Gamification by Design[50]. Therefore it is important not to exaggerate the use of badges and carefully consider which badges to use. This also concurs with some of the feedback from a educational badge study by Lassi Haaranen and Korhonen[28]. To successfully implement badges, Jen-Wei Chang and Hung-Yu Wei[11] suggest that developers implement badge ladders, where one badge can have several levels (Bronze, Silver, Gold etc) as well as milestone badges that can be achieved when a certain set of badges have been collected. They also mention that the badges should contain a description and be shown in the users personal space.

In the article by Hamari and Eranti[17], badge rewards are divided into three different categories: in-game, meta-game and out-game. The in-game rewards stays within the game, like a profile page in a game containing badges. Meta-game rewards abstract the awards to the game platform. For instance, if multiple courses are all in one gamified system, the meta-game reward would be awarded there. The out-game awards are rewards you get in the real world, such as a free coffee or a contribution to the grades of a course[17].

In a study conducted by Lassi Haaranen and Korhonen[28], a large group of students were tested in a gamified version of a course. The gamified course awarded badges to students who submitted the exercises early, and additional badges to the students who submit a correct assignment on their first attempt. The study revealed that students were motivated to achieve more badges than their class mates, and recommended the option of sharing badges. Another recommendation by the article was to introduce funny badges that didn't necessarily matter specifically to the course[28].

3.2.4 Team leaderboards

Competitiveness is a good motivator that drives gamers, athletes, services, and brands. Dr. John Houston found that competitive people compete even if nothing can be gained[50]. Team leaderboards takes advantage of people’s competitiveness. The leaderboards rank the different teams based on some score or grade, and students can compare their results with that of other classmates. A study by Chang and Wei[11] has shown that leaderboards increase the students reliance on team effort rather than their own[11]. The same study also ranked team leaderboards as the third most motivating reward.

Skill differences and motivational loss

When one team advances on the leaderboard, another team must regress. Balancing winning and losing is difficult. Research suggests that if a motivated piano player starts off winning competitions and then subsequently loses, this will cause the piano player to stop playing piano[50]. A game called “Health Month” introduces a possible solution to the issue of loosing motivation after subsequently loosing[50]. The purpose of “Health Month” is to inspire action outside the game. It therefore uses a structure that is more achiever-oriented than most games. To make sure that the over-achievers don’t ruin the motivation of the other players, players are automatically grouped into brackets based on their level. This will decrease the constant low ranking of some users.

3.3 Code quality

For physical products you can measure quality by the probability of failure. This is however difficult when it comes to software[46]. Code quality and other non-functional aspects of the code does not play a big part in the course TDT4100, but it is useful to learn at an early stage for a programmer. Especially when you consider the 10 to 1 ratio of time programmers spend reading versus writing code[32]. The possibility to automatically retrieve information about code quality means that the prototype could some day make use of these aspects. This would make it possible to lightly introduce non-functional aspects the coursework of TDT4100, and even adapt the prototype to followup courses where code quality and non-functional aspects play a bigger part. This section addresses two different approaches to determining code quality: the ISO/IEC 25010’s model for data quality and SOLID, Robert C. Martin’s five principles of software design.

3.3.1 ISO/IEC 25010

The ISO/IEC 25010 is a product quality model that, through different characteristics, tries to measure and evaluate systems and software product quality [15]. One of these characteristics is maintainability. Table 3.2 describes the five sub-characteristics of maintainability that is used to evaluate code quality.

Name	Description
Modularity	Modularity is based on how well the system is divided into discrete components, where change to one part will have minimal impact on the others.
Reusability	Based on which degree code is reused in more than one system or used to build other assets.
Analyzability	Analyzability is based on the efficiency of determining what's wrong with the system, or what a impact a change will have on the system. The goal is to effectively find the deficiencies or causes of failure, and to identify which parts should be modified.
Modifiability	Modifiability is determined by how effectively the system can be modified without breaking existing functionality.
Testability	Testability measures the effectiveness of establishing testing criteria, and how well these tests can be performed.

Table 3.2: ISO/IEC 25010 Metrics[15]

3.3.2 SOLID

SOLID is an abbreviation for the five principles of software design. The five principles aim to make the code reusable, maintainable, and robust. All five principles are connected, therefore all of them must be maintained to achieve these advantages.

Single responsibility principle

Classes with more than one responsibility should be broken down into smaller classes using delegation and abstraction. This way, the big problem is divided into sub-problems and each of the sub-problems will be dealt with by a separate class[21] This reduces responsibility confusion, and increases order and clarity[33].

Open/closed principle

The Open/Closed principle (OCP), is a guideline in class and interface design that helps developers make code that allows for change over time. The OCP states that software should be “open for extensions but closed for modification”. To uphold OCP requirements, classes that are likely to change should be factored out behind extension points [33]. This is done because changing an existing class may cause an otherwise working system to break.[21]

Liskov substitution principle

Liskov substitution principle (LSP) is a group of guidelines used when creating inheritance hierarchies. It states that “derived classes should be substitutable for their base classes”[21]. States of classes/objects must be replaceable with their sub-classes without breaking the program.[33] For instance, if you have a class called ”Rectangle” with ”getters” and ”setters” for height and width, you would think that another class ”Square” could inherit from Rectangle. A square also has a width and a height, however a square cannot have the width and height set independently. This polymorphic behaviour through inheritance breaks the LSP and may cause problems to the system.[21]

Interface segregation principle

The Interface segregation principle, or ISP, states that clients of interfaces or classes should not be dependant on methods they do not use[21]. Many members of big interfaces only need a fraction of the properties.

Dependency inversion principle

The DIP is a simple principle. It states that high-level concrete classes should not depend on low-level concrete classes. Instead, they should both depend on either an abstract class or an interface [21]. That way, the high level concrete class will not be broken if the low level class is altered, as long as it continues to use the same interface.

3.4 Code Metrics

Code Metrics is a number of software measures extracted from code. These measures can give developers greater insight on the ”quality” of the code, and help them decide what parts of the code should be improved. This section describes some existing software metrics tools, and a more detailed description of some of the metrics used.

3.4.1 Existing tools

There are currently many different tools used to inspect source code JCSC, CKJM, Understand, and NDepend are some examples. They all gather information from the code, and use a subset of this information to determine the score for a code aspect. Then together the different aspects of the code try to give some quality score of the code.

Java Coding Standard Checker

JCSC is a basic tool used to inspect source code and determine if it complies with coding standards, and the probability of it being bad code. JCSC looks at naming conventions for fields, parameters, interfaces, and classes as well as the structural layout. When looking for signs of bad code, the code is searched for switches without defaults, classes throwing Exception-objects and empty catch/finally blocks.[38]

CKJM

CKJM is a more advanced tool which calculates the metrics suggested by Chidamber and Kemerer. The table below shows a list of some of the metrics used in CKJM.

Name	Description
Depth of inheritance	Returns the inheritance level starting from the object class.
Class children	Returns a class's number of descendants.
Response for a class	The total number of class methods called, including methods that are contained within the class methods.
Lack of cohesion methods	Measures the level of information cohesion in the classes, by counting the number of methods that does not share any class fields[39].

Table 3.3: CKJM Metrics

The metrics also include *Weighted Method per class* which will be explained in chapter 3.4.2.

Understand

The tool Understand differs from CKJM by grouping the different metrics together into three categories, Complexity, Volume, and Object Oriented that are used to analyze different groups of metrics namely Complexity, Volume, and Object oriented. In addition to most of the metrics from CKJM, Understand introduces two new metrics.[38]

NDepend

NDepend uses 82 metrics that retrieve information to explore mainly three aspects of the code. *Code organization* uses metrics to investigate the how the project is organized, using metrics to find the number of classes and how many methods are declared in each class. The second aspect is *Code quality*, which is determined using metrics like complexity, cohesion of classes, and number of parameters. The final aspect is the *Structure of code*, which explores how the code is connected to itself using forinstance the depth of inheritance metric.

3.4.2 Specific metrics

Lines of code

Lines of code is a simple metric. The metric consists of how many lines of code are written in a program. A high number of lines in a method or class may indicate that too much work is done in a single class, which can result in the code being difficult to maintain[34][10].

A study made by Khoshgoftaar and Munson, however, revealed that other metrics that focused on complexity were of higher value[26].

Cyclomatic complexity

Cyclomatic complexity developed by Thomas J. McCabe[31] is a complexity focused metric that counts the number of independent paths through the code. The equation below is used to find the Cyclomatic Complexity:

$$\text{CyclomaticComplexity} = E - N + 2P \quad (3.1)$$

Equation (3.1) can be used on a flow graph where "E" is the number of edges, "N" is number of nodes, and "P" is the number of connected components which for most simple applications is one. Figure 3.1 shows a code snippet and the appurtenant flow graph of the code.

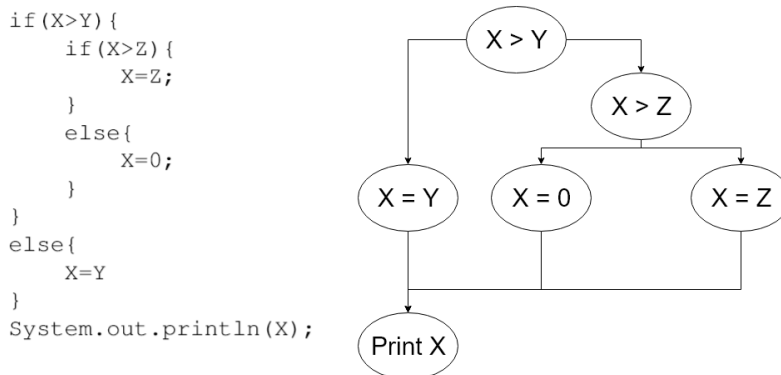


Figure 3.1: Code snippet and Flow graph

The flow graph in Figure 3.1 has six nodes and seven edges. Using formula 3.1, we find the Cyclomatic complexity to be:

$$7 - 6 + 2 * 1 = 3 \quad (3.2)$$

The result can give an approximation of the software maintenance cost of the code[16]. If the Cyclomatic Complexity is high, the code will be more difficult to read and more expensive to maintain. In larger projects, *Cyclomatic Complexity Density* is found to be more accurate[16]. Cyclomatic Complexity Density takes the system size into account and divides the Cyclomatic Complexity with the size of the project.

Weighted method per class

Weighted methods per class is a metric that builds on the principles of Cyclomatic Complexity. It calculates the sum of the complexity of each method in a class. The equation

below is used to find the method weight of a class[35].

$$\text{WeightedClassComplexity} = N + \sum_{p=1}^s MC_p \quad (3.3)$$

Where MC is the MethodComplexity that is calculated for all the methods in the class, and N is a constant added to handle the complexity introduced by variables used in more than one method. The equation can also be extended to get the complexity of all classes in the code, with the equation below[35]:

$$\text{TotalWeightedClassComplexity} = \sum_{x=1}^y WCC_x \quad (3.4)$$

Where y is the total number of classes, and WCC is the Weighted Class Complexity. The advantage of Weighted Class Complexity is its ability to consider internal architecture when calculating complexity. Weighted methods per class is regarded as a metric with good qualities for an evaluation framework[35], whereas a high complexity number is negative because it might lead to higher maintenance costs[10].

Class Coupling

Class coupling is a metric specific to object' oriented languages that measure coupling between classes. The metric measures coupling of parameters, local variables, return types, and method calls, among other things. Parameter coupling is defined as when the method of one class invokes methods or passes parameters to methods of another class[9].

According to the Microsoft Developer Network, low class coupling is an indication of high quality code[34]. High coupling on the other hand, makes code difficult to maintain due to the high number of interdependencies. Therefore, class coupling thresholds are needed, however, where to set these thresholds however is not fully defined[9]. The *AvoidExcessiveClassCopl*ing point in Visual studios analysis policy sets the threshold for parameter coupling at 30 and class coupling at 80 [30]. This roughly concurs with the article by Bidve* and Sarasu[9] where the parameter coupling threshold was set at 24 to 34 [9], Therefore, it is a metric that works best in large projects with multiple classes.

3.5 Case study - Khan Academy

Khan Academy is an learning organization founded by Salman Khan. The organization has created a web platform with different tools to help teach kids and grown ups topics including math, physics, biology, chemistry, economics, and computer science [36]. When a participant chooses a topic, a step by step path is presented where the participants work their way from the basics to the advanced concepts via videos, articles, and quizzes. In the math topic, the order of these steps are determined by a knowledge map[24]. The knowledge map is built like a talent tree, where a mapping between each of the sub-topics are presented.

Another learning aid used by Khan Academy is gamification. To help increase motivation, Khan Academy uses multiple virtual rewards, one of which is Badges. There are six different categories of badges: *Meteorite*, *Moon*, *Earth* and *Sun* badges. These badges are all rewarded based on the level effort put in to learning different topics. Meteorite badges are described as "easy to earn" and sun badges "require impressive dedication"[23]. The final two badge categories are the *Black Hole* badges which are unknown as well as the rarest Khan Academy award. *Challenge Patches* that are awarded for completing topic challenges[23]. Another virtual reward used is *Energy points*, which are awarded when participants complete the quizzes involved in the step by step path of their chosen topic.

The article by Morrison and DiSalvo[36] from 2014 criticizes some of the gamification elements of the Khan Academy. Today, however, most of these elements have been changed to match the suggestions by the article. The *Challenge Patches* works as milestone badges, where a lot of work is required compared to some of the other categories. The platform also gives the students the option to share badges they have been awarded in their profile page. The sharing of badges was also recommended by [28]. The final motivational tools used by Khan Academy is to keep track of the number of days in a row the user has been active and show the highest number of days as a streak.[25].

Gamifying TDT4100

This chapter will describe the process of making a gamified prototype. First, a quick introduction will be given of TDT4100 and the coursework. Followed by a description of the metrics found relevant for the course. Then the results of conducting a simulation on the prototype will be presented. The results are divided into two parts, first the simulated use of gamification and metrics will be presented, followed by an explanation of the method used to tailor these metrics specifically to each task. Appendix B shows how the results are printed by the backend, and appendix C gives a description of how the backend was implemented. The summarized results can be found in the git-repository of the project. The link to the repository is provided in C.

4.1 TDT4100 - Object oriented programming

This section will give a basic description of the course Object oriented programming (TDT4100), and which parts of the code that are relevant to include in the solution. On NTNU's webpage the course content is described as the following:

"Basic algorithms and data structures, constructions and control flows in object-oriented languages. Modularization and reuse. Standard software library. Device testing, malfunction and tools for this. Object-oriented design. Use of class, sequence and interaction charts in UML. Use of design patterns. Java is used as the implementation language."[2]

The course description includes more than just programming. Most of the coursework however, consists of programming exercises where the students are given some system requirements, and a set of java tests they must pass. One could argue that these tests in itself are a way of gamifying the coursework, where the goal is passing the test. Because the goal of the thesis is to supplement the coursework. The solution will try to introduce additional gamification features to the programming exercises with a focus on improving

the learning of the "*standard software library*" and "*construction and control flow*".

4.1.1 Coursework

The students has to gather 750 points from the assignments to be allowed to attend the exam, where 100 points are awarded for each completed assignment. To attain 100 points from an assignment, the student have to pass a number of JUnit tests, and present the assignment to a teaching assistant. There are a total of 10 assignments in the coursework, each of which contains multiple tasks in three difficulty levels: easy, intermediate and hard. The student normally has to complete either one or two of the tasks, in order to complete the assignment.

The assignments are meant to give training in usage of the Java programming language, and achieve the learning objectives of the course. The JUnit tests, handles the writing of functional code. The code style is is handled by the teaching assistants.

As mentioned before, both the points system, and JUnit tests work as a kind of gamification. To further introduce gamification, features information about the students code, must first be attained. This can be done using the *Javaparser*, explained in appendix C. The section below gives an overview of which metrics are suitable when determining the students' understanding of the programming curriculum of TDT4100.

4.1.2 Relevant metrics

The metrics relevant to the course was determined reading through the wiki page and slides of the course[48], in addition to dialogue with Trættemberg. The metrics chosen are a mix of everything from the use of a boolean operator "&&", to calculating the *weighted method per class* metric, explained in Chapter 3.4.2. A total of 39 sub-metrics were chosen to collectively enclose most of the programming elements relevant for TDT4100. To avoid an overly complex frontend, they have been grouped into six categories. The table below lists, the six categories, how many sub-metrics each of the metric categories are comprised of, and a short description of the metric category.

Metric Category	Sub-metrics	Description
GeneralMetrics	13	Lines of code, public, private field declarations weighted method per class, and the unary math expressions
ExceptionMetrics	4	Try and catches as well as throws. Inheritance handles abstract classes, interfaces and super expressions
InheritanceMetrics	2	Abstract classes, super expressions and interfaces
IterationMetrics	7	Creating, using and a iterating through lists
ConditionalStatementMetrics	13	If and else sentences in addition to summarizing boolean operators and calculating cyclomatic complexity
OverallMetrics	34	Summarizes through the java-element usage data from the other metrics to give a overview of the coverage of the course curriculum

Table 4.1: Metric Categories

A more detailed explanation of the *General Metrics* and *Overall Metrics*, in addition to an overview of which parts of the curriculum are handled by which metric, can be found in appendix C. The full list of sub-metrics can be found in appendix F. The metric categories listed above are mostly made up of java-element metrics, which gives information about what code is used but little information about the structure of the classes.

Class coupling, described in chapter 3.4. Is an example of a metric which could retrieve this information. It was however decided not to include it, because advanced knowledge of class coupling exceeds the scope of the course. The literature also suggested, that small code sizes could impair the feedback from the metric. The *Java Symbol solver*, explained in appendix C, does however support the extraction of these metrics. Therefore it could be added to the prototype at a later stage.

4.2 Badges

Badges, was one of many virtual rewards introduced as gaming elements in the first game element building block described by Chang and Wei[11]. Badges were chosen over the other virtual rewards, because they can represent specific parts of the code, and give a more detailed feedback of what is awarded.

To give the badges an additional depth, the badge ladders suggested by *Jen-Wei Chang* and *Hung-Yu Wei* [11], were also introduced. Figure 4.2 below shows the three levels of the badge ladder.



Figure 4.1: Badge valor's

The three levels represents gold, silver, and bronze medals, and were added to the badges to give the students something to strive towards. Hopefully the students will be motivated to try and achieve better badge, even if they have completed the tests of the assignment. Figure 4.2 shows the gold valor of the six badges added to the solution.



Figure 4.2: All badges

The badges were designed to illustrate what the badge represents, the following section will describe how these badges are awarded.

4.3 Reward model

This section describes the reward model of the prototype. Figure 4.3 shows the planned reward model for the prototype.

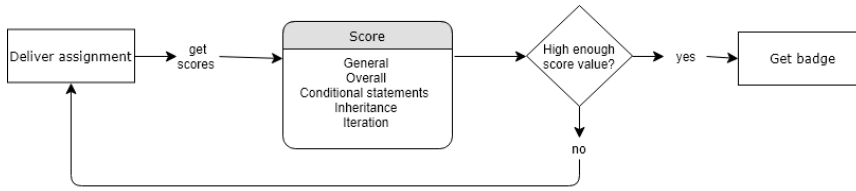


Figure 4.3: Planned reward model

After delivering an assignment, the backend extracts information about each of the Metric Categories in table 8.1. Score ratios are then generated by comparing the metric information of the student, with the metric information of the proposed solution. Shown in formula below(4.1).

Let $P_s(i)$ be the value of sub-metric i , for the proposed solution
 Let $S(i)$ be the value of sub-metric i , for the student

$$scoreRatio = \frac{1}{n} \sum_{i=1}^n \frac{S(i)}{P_s(i)} \quad (4.1)$$

Where n is the total number of sub metrics

The ratio between each of the metrics is summed and averaged. If there for instance is a metric category that only contains two sub-metrics, code lines, and complexity. Given that a student code contains 300 code lines, and has a complexity of 6, and that the proposed solution contains 150 code lines, and has a complexity of 4 the score ratio will be:

$$scoreRatio = \frac{1}{2} \sum \left(\frac{300}{150} + \frac{6}{4} \right) = \frac{1}{2} (3.5) = 1.75 \quad (4.2)$$

After the scores are calculated. They are compared to a list of predefined score ratio requirements, that determine which badge should be awarded. If the score is high enough, the badge is awarded. The table below shows the score ratio requirements used to determine if a badge should be awarded, or not.

Badge name	Bronze	Silver	Gold
General Metrics Badge	3	2	1
Inheritance Badge	4	3	2
Iteration Badge	5	2.5	1
Conditional Statements Badge	3	2	1
Exception Badge	2	1.5	1
Overall Metrics Badge	0.4	0.6	0.8

Table 4.2: Badge requirement levels

Because short and concise code is an indication of high quality code[10], a low ratio

which indicates a short and concise code is preferred. Except for the Overall Metrics Badge, where the numbers represent the percentage of metrics used. This is because a high number of different metrics used, indicates that a student has been through a lot of the programming curriculum, which is rated positively. The score ratio requirements listed

in table 4.2, were decided by qualified guesses, after trying to balance the valor distribution on a few of the students. A possible method for configuring these score values, is discussed in chapter 8.1.

4.4 Simulation as method

Simulation is the process of producing an output, through the use of a model composed of a set of rules. This output may infer what could happen in a real situation if such measures should occur[13]. There are many different uses of simulation: prediction, performance, proof, and theory discovery, are some of examples mentioned by Axelrod[7]. The use of relevant data when performing a simulation gives a higher probability of useful results[13]. Therefore it is important to use actual domain data when possible.

In this setting, the simulation of the system is conducted using the archive of assignments from last years students. This was done using a discrete event simulation model, where the assignment submission works as the time step data, and the each jump is the jump from one assignment to the next.

The goal is to use the results from the simulation to adjust the badge requirements, and configure the metric relevance for each task, which may result in a more just badge distribution. It is expected that the students from this year, deliver similar code to the students attending TDT4100 next year. Therefore the configuration obtained may be applied directly to nest year's students.

4.5 Validating the use of gamification

In the solution, the introduction of badges are used to gamify the coursework of TDT4100. These badges are awarded based on six metric categories, extracted from the code of the students. The following section presents the results after simulating usage of the backend with student code, previously delivered by three students in 2017.

4.5.1 Badge award simulation

To simulate badges being awarded by the system, the code delivered by three students for assignment five, six, eight and nine, was run in the backend. The assignments were chosen for their availability. Student data for the other assignments, was not available at the time the simulation was conducted. The results were collected and summarized in

excel, appendix B shows an example of the results are printed by the backend. Table 4.4 shows the distribution of badge valors, for each of the three students.

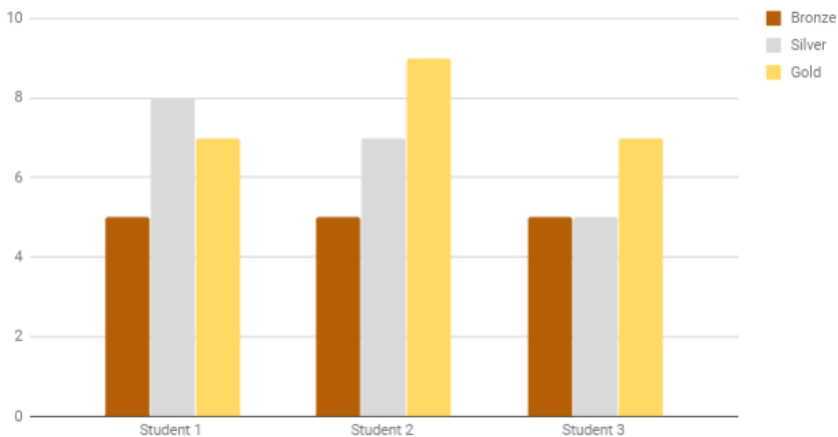


Figure 4.4: Badges awarded each student

Note that when a student badge is upgraded to a new valor, the old badge is changed. Therefore the students will never have more than one badge in each category. The three students all received roughly the same distribution of badge valors, which could be expected because their only goal was passing the test. The high number of gold badges awarded however, was unexpected. To explore this further the results were sorted based on badge types, to see if some badge categories were too kind when awarding the gold badge. The table below shows the average number of badges each student was awarded, organized by badge category.

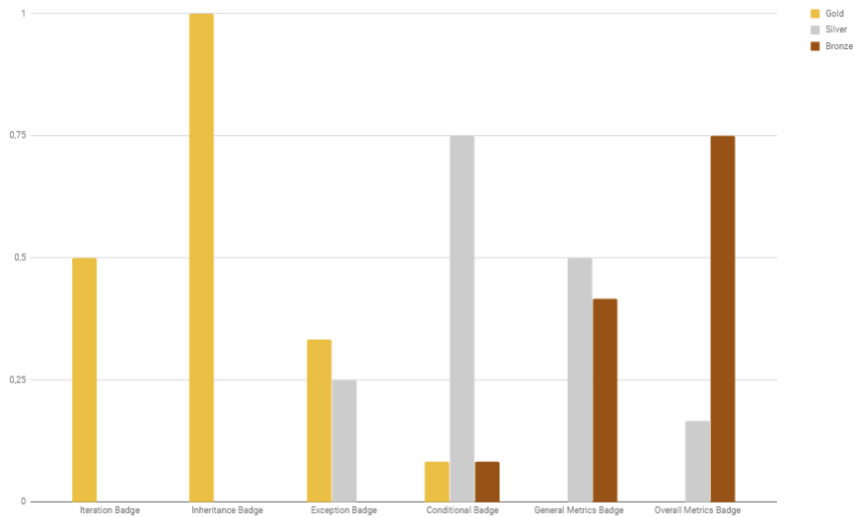


Figure 4.5: Badges awarded for each metric

As seen in figure 4.5, the badge values are not evenly distributed. It also reveals a pattern, seen on the values of the inheritance and iteration badge. All three students were awarded one *Inheritance gold badge*, and half a *Iteration gold badge*, for each of the assignments they did. This awarding is very kind, compared to the *Conditional-* and *General Metrics Badge*. This may be a coincidence due to the small number of students, or that the score levels from chapter 4.3 are unevenly defined. This will be discussed further in chapter 6.5..

4.5.2 Overall Metrics Accumulated

The overall metrics badge, counts the number of java-element metrics retrieved by all the other metric categories. Figure 8.10 shows the accumulated number of metrics for each of the students as well as the proposed solution, after completing assignment five through nine.

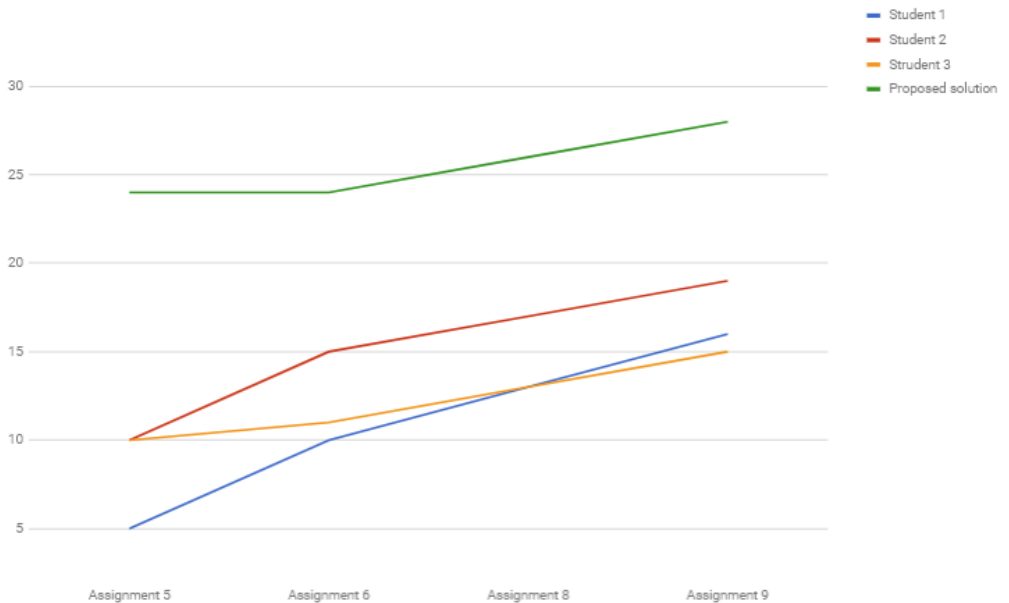


Figure 4.6: Overall metrics diagram

As explained in chapter 4.3, the overall metrics badge uses 34 different java-element metrics as basis for giving the score. The three students ended up between 16 and 19 individual metrics, compared to the 28 metrics used by the proposed solution. The reason for the big difference between the students and the proposed solution, could be that all the tasks in assignment five, six, eight and nine were completed by the proposed solution, while each of the students only completed the minimum number of tasks.

To get some insight to which metrics were unused, the results were aggregated over each of the students, for all the assignments. Table 4.3 shows the list of the metrics unused by the students, as well as the metrics unused by the proposed solution.

Unused Metric	Student	Solution	Example
Lambda			<i>isNegative = (n) → (n<0);</i>
Addition			+
Multiply			*
Divide			/
Remains			%
MinusOne			- -
ArrayCreation			<i>ArrayList<String>list = new Array...</i>
Switches			<i>Switch(day){case 1: day = "Friday";}</i>
PublicFieldDeclaration			<i>public Person person;</i>

Table 4.3: Unused metrics

The green squares illustrate that the metric has been used, and the red squares represents that it hasn't. As seen in table 4.3, neither the students or solution made use of the lambda expressions. This may be because lambda expressions are a relatively new addition to Java[37], or that it was easier to use a loop. The small number of mathematical operators that remained, were most likely unnecessary to use in the four assignments tested.

The remaining java-elements that were used by either the students or the proposed solution, can be worked around using *if sentences* instead of *Switches*, and *List* instead of *ArrayLists*. Even though the assignments can be completed without the use of these java-element metrics, they are still relevant for the exam.

4.5.3 Metrics sorted by task difficulty

Because the assignment tasks have three difficulty levels, a simulation was run to determine if you could see the difference in difficulty, on the code metrics. It was expected that the difficult assignments would have more code lines, and a higher number of code metrics than the easy assignments.

All of the tasks involved in assignment five, six, eight and nine, were sorted by difficulty, and the metrics of the proposed solution was extracted, summarized and averaged for each of the assignments. Figure (a) below, shows the average number of code lines used to solve the assignments, sorted by difficulty. Figure (b), shows the average number of usages of some of the java-element metrics.

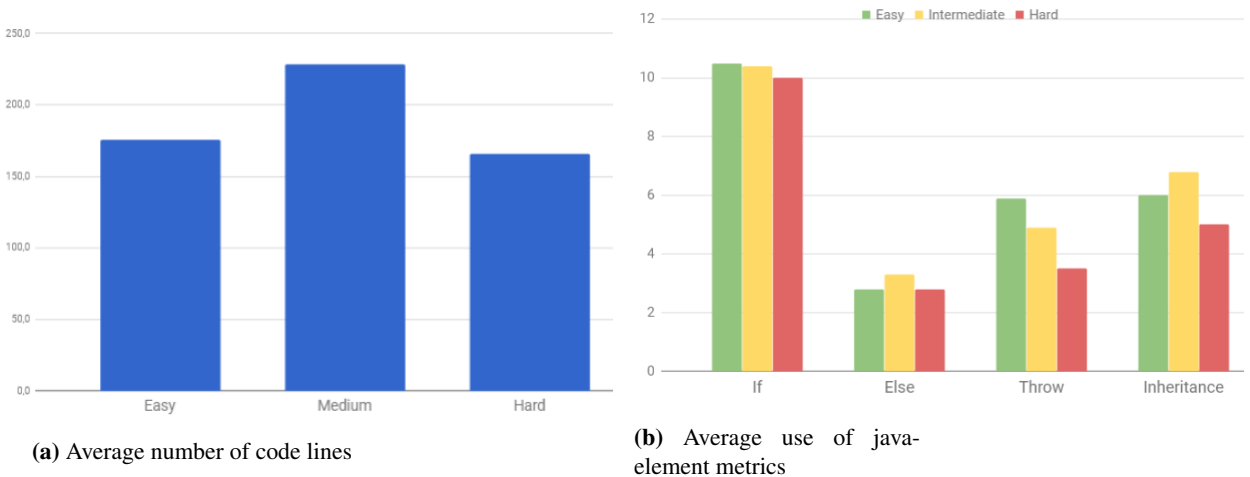


Figure 4.7: Metric results sorted by difficulty

As seen in figure 4.7, there doesn't seem to be a correlation between difficulty and the number of code lines shown in (a), or the use of that selection of java-element metrics shown in (b). This shows that these metrics can't give a good measurement of the assignment difficulty, which is troublesome, because it is desirable to award the students completing the difficult assignments.

These results also made it evident that the number of uses, shouldn't be used as a quality criteria on every metric. The realization also implies that the reward criteria for some of the metrics is unfounded, which means that some of the values in table 4.2, are left vapid. This however doesn't mean that the reward model is rendered useless. Instead of comparing the metric usages with the proposed solution, they can be compared to a static value. Chapter 6.2 will discuss the alternative for the ratio reward model further.

Even though some of the metrics should be awarded using a different method than the ratio calculation. This does not mean that the method is useless for all the metrics. The *number of code lines*, *weighted method per class*, and *cyclomatic complexity* metrics should all have their score calculated based on the number of uses[[16]][10], and can therefore continue to use the ratio score method.

A way to distinguish the students completing the difficult assignments alternatively from the one used in figure 4.7, is to look at code complexity instead of just quantitative use of the java-elements. This was tested by extracting the *Cyclomatic Complexity* for each of the tasks. Figure 4.8 shows the average *Cyclomatic Complexity* sorted by difficulty.

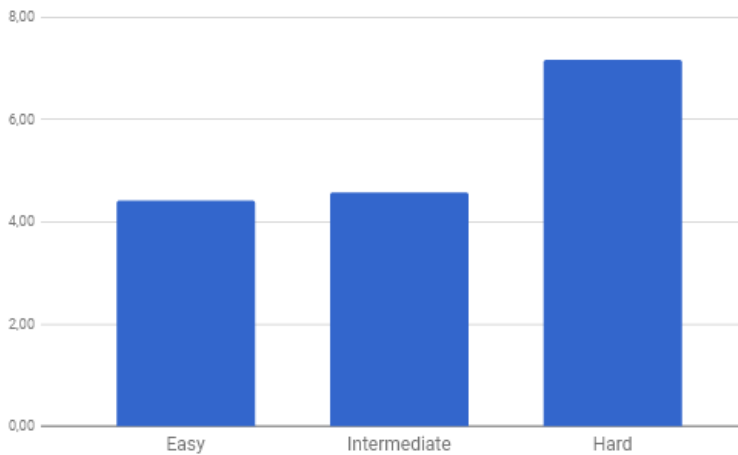


Figure 4.8: Average Cyclomatic Complexity

Figure 4.8, shows that the hard tasks on average has a higher complexity than the easy, and intermediate tasks. This could mean that there is a correlation between code-complexity and task difficulty, which would be useful when attempting to award the students that complete the difficult tasks. If the proposed solution has a high complexity on a specific task, the student solving it, should receive a greater reward than a student solving a task where the proposed solution has a lower complexity.

4.6 Tailoring metrics for assignments

Some of the metric categories are hardly used in some tasks, and because it is undesirable to award a student a gold badge for a single code metric. A config file has been added to the backend. The config file contains a mapping between each of the tasks, and an associated list of relevant metrics. The score values that determine the badge valor, can also be found in the config file. This section describes the method used to configure which badges are relevant for which task, and some of the results.

4.6.1 Method for configuring relevant metrics

In order to configure which metrics are relevant for which task, the proposed solution of the task is run through the backend, and the relevant metrics are chosen based on usage information from each metric. Because the *Overall Metrics Badge*, and *General Metrics Badge*, are relevant independently of task, these two are disregarded in this method. Table 4.4 shows the metrics that are determined relevant, for each of the tasks based on their frequency of use.

Exception	Inheritance	Iteration	ConditionalStatement
Catch Statements	Abstract class	Array Accesses	If Statements
Try Statements	Interface inheritance	Array Creations	Else Statements
Throw Statements	Super expressions	Array Inits	Switches
IllegalArgumentExceptions		Foreach loops	
		For loops	
		While loops	

Table 4.4: Submetrics used to filter relevant use

The sub-metrics listed beneath the main metrics in Table 4.4 are the ones that are counted, when determining if the metrics are relevant or not. This is because the use of those sub-metrics, implies the use of the other sub-metrics as well. If one or more of the sub-metrics are used three times or more, the metric is categorized as relevant. The boundary of three usages was chosen to filter out the metrics that are hardly used, but still include as many metrics as possible. Most of the proposed solutions are concisely written, so increasing the metric usage boundary to four, resulted in the iteration metric hardly being relevant at all. Table 4.5, shows the metrics categorized as relevant for the tasks of assignment five.

Task	Metric
Card part 2	Exception, ConditionalStatements, Inheritance
Partner	ConditionalStatements, Iteration
Twitter	Exception, ConditionalStatements, Inheritance, Iteration
Stop Watch	Exception, ConditionalStatements, Inheritance, Iteration
Person	Exception, Conditional Statements

Table 4.5: Relevant metrics for the sub-tasks of assignment 5

The relevant metrics for each task of assignment five, listed in table 4.5. Can be obtained automatically from the backend, as long as the names of the tasks are supplied. This opens up for the possibility of automating the badge configuration, which can simplify the badge configuration process.

The relevant metrics for the tasks of the three remaining assignments were categorized, and some statistics were drawn from the results. Figure 4.9, shows the distribution of which tasks the *Iteration metric*, was relevant for, sorted by difficulty.

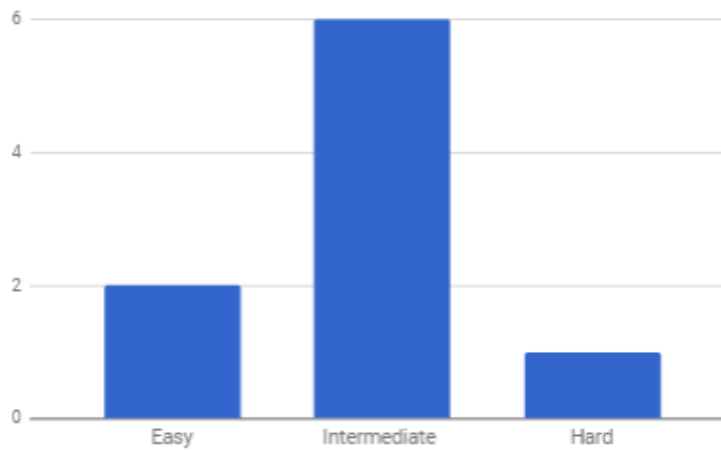


Figure 4.9: Tasks where the iteration metric is relevant

Figure 4.9, shows that six of the intermediate tasks, use three or more of the iteration metrics. This is 85%, compared to the 28% for the easy tasks, and 16% for the hard tasks. The high number of relevant *Iteration metrics* for the tasks of intermediate difficulty, suggest that the intermediate tasks are useful for learning iteration. One possible use of this information, is to recommend the intermediate assignments for the students struggling with achieving the iteration badges.

Chapter 5

Design

This chapter will describe the design of the project. First the prototyping method and tools used will be introduced, before describing the iterations and testing that were conducted to help make out the final design. Then the requirements for both the frontend and backend will be explained before the finally presenting the final design.

5.1 Rapid Prototyping

Rapid prototyping is a very simple type of prototyping. The idea is to spend a minimal amount of time on requirement analysis, and instead build a simple prototype which is used to understand and the actual requirements. After the requirements are decided, the prototype is discarded, and the system is developed[44]. Rapid prototyping was chosen because of the lack of any clear predefined requirements of the frontend. It also made it possible to simply communicate the general idea of the design, and get feedback from the very start of the prototyping process.

5.2 Tools

In order to create the prototypes, mainly two tools was used. Microsoft Paint, and InvisionApp. Paint, is a very simple drawing program which basically offer the same functionality as pen and paper. InvisionApp, is a web-based tool that can use images to build interactive prototypes, and also offers LiveShare where these prototypes can be tested online.



Figure 5.1: Prototyping tools used

Paint and InvisionApp was chosen because of previous experience with the tools, their overall ability to produce simple wireframes, and the simplicity of making modifications to that wireframe. Before deciding to use InvisionApp, Axure, was also considered. Axure, was however found to be a bit excessive for the simple view required in the solution.

5.3 Iterations

The prototype was developed iteratively, this section gives a short introduction to the iterations the prototype went through before landing on the final design.

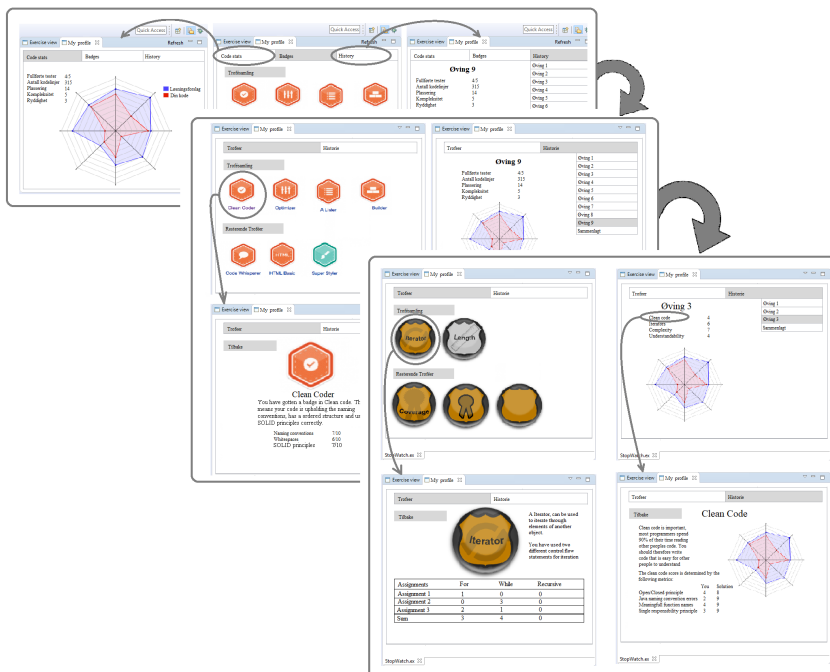


Figure 5.2: Iterations

As seen in figure 5.2, the prototype went through three iterations. Between each of

the iterations the prototype went through a informal tests. The feedback from the test was used, when deciding on which changes to make to the prototype. A more detailed step by step explanation is given in Appendix E. After the third prototype was completed, a more formal testing was completed.

5.4 Testing

In this section a description of the more formal testing will be described, as well as discussing improvements that can be made for the final design. The testing was done to investigate the users understanding and friendliness toward the system. The tests were conducted on three students who has previously completed TDT4100. The reason for conducting the tests on previous students, was partly because the tests were conducted in the autumn when the course isn't taught, and partly because the simplified prototype was considered easier to understand for the previous students, who are more familiar with both the coursework, and the meaning of the badge names. First a usability test was conducted by the test subjects, followed by a survey with questions about their experience with the system.

5.4.1 Usability testing

The usability test comprised of four tasks the test subjects had to try and complete in the wireframe. The tasks they were asked to preform are listed below.

1. *Navigate to the page that shows information about the second assignment.*
2. *Find a more detailed description of the clean code metric.*
3. *Navigate to the page that shows a summary of your achievements so far.*
4. *Locate the specific information about one of the badges you've achieved*

The four tasks were chosen because they collectively include all the different views of the solution. The test was completed using Skype, and *LiveShare* from InvisionApp. *LiveShare* made it possible to see the testers screen while they completed the tasks, and observe their navigation. The reason for conducting such a remote testing, was to follow the principle from [42] which states:

"The participant observer must take measures to ensure that those being observed are not constantly thinking about being observed"[42]

This is due to the *Hawthord Effect*, which addresses the fact that people change their answers if they know they are being observed. All the participants managed to complete the four tasks in decent time without help. Two of the participants did however spend a significant amount of time solving task two, compared to the remaining three. The observation revealed, that they had trouble recognizing the button, due to them hovering the mouse pointer on the right place repeatedly, without clicking.

5.4.2 Survey

As described in Chapter 2, the survey was conducted as an alternative to the interview phase described by Seaman[42]. The survey was made in Google Forms and consisted of twelve questions, six open-ended, and six close-ended questions. The questions disclosed mostly the participants experience with the content of the prototype. They therefore had access to the prototype when completing the survey. This section presents some of the feedback attained from the survey, that was relevant for the changes made to the final solution. First some results from the close-ended questions will be presented, followed by a selection of the answers from the open-ended questions. The full survey can be found in Appendix A.

Close-ended questions

In most of the close-ended questions, the participants all gave the same answer. The figure below shows a pie chart of the answers to question eight, nine, and ten, of the survey.

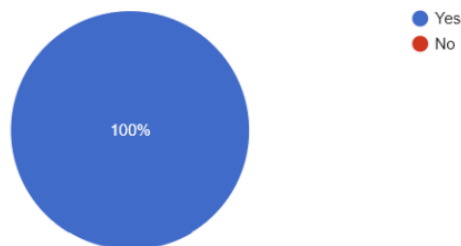


Figure 5.3: Pie chart of the results of questions eight, nine and ten

Question eight, and nine asked whether the participants understood the detailed description of the *Iterator* badge, and the *Length* badge, given in the prototype. Question ten asked whether the participants would like to see the progress towards the next badge level.

Because of the assistance provided by the talent tree in Khan Academy, it was found relevant to investigate whether the students would prefer a talent tree, instead of badges. Figure 5.4 below illustrated the results from question eleven
"Would you prefer a talent tree over badges?"

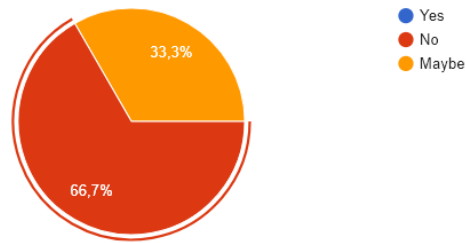


Figure 5.4: Cake diagram of the results from question 11

Figure 5.4, shows that two of the three test subjects would not prefer a talent tree over badges, and the third participant wasn't sure.

Open-ended questions

In the open-ended questions, more detailed answers were given. One of the questions asked for suggestions to other metrics that would be fitting for TDT4100. Here two good suggestions were suggested by the test participants, "*Error handling and robustness*", and "*Meaningful variable names*". Both "error handling", and "meaningful variable names", are considered as good programming practices[41], and are relevant for TDT4100.

The final question, asked whether the participants had any other comments to the solution. Two of the participants answered, that they did not understand what the spider diagram in the *History* view, illustrated. There was also some frustration on the design, one of the test participants wrote, "*Buttons don't look like buttons*". This matches the observation from the usability testing, where the participants had problems locating a more detailed description of the clean code metrics.

5.4.3 Final changes

This section will describe the final changes made, when building the high fidelity prototype in HTML. The changes were made, based mainly on the feedback from the survey, but also through dialogue with Trøetteberg. The biggest change when designing the page in HTML, was the decision to move away from just showing the web page as a web-view in Eclipse, shown in figure 5.5.

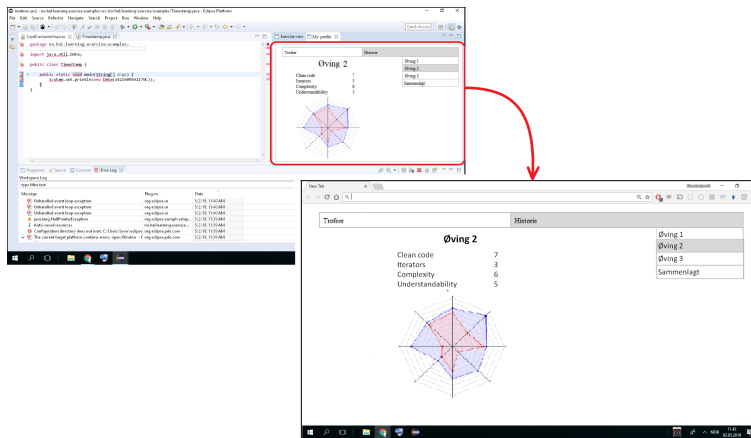


Figure 5.5: WebView to Web

Figure 5.5, shows the transition from the originally planned view in Eclipse. To a responsive web-page, that the student can view both in a browser, and Eclipse. This makes the solution easier to access, which may lead to elevated use[12]. Figure 5.6, shows the changes made to the button visibility.



Figure 5.6: Button visibility

The change was made by adding a hover effect to the text that was clickable, and changing the mouse pointer. A similar alteration was also made on the navigation buttons, and badges.

As seen in Figure 5.6, the names of the metric-categories were also changed. This was done to match the front-end names, with the names in the backend, and to introduce the exception metric which was added as a response to the "error handling", suggestion in the survey. The new metrics are explained in more detail in appendix C.

Two parts of the prototype was removed, the table in the detailed badge view, and the spider-diagram. The spider-diagram was removed due to the test-participants difficulty in understanding what it illustrated. Ideas for a replacement of the diagram are discussed in chapter 8. The table extending the detailed badge view was determined to be too chaotic, due to the increased number of sub-metrics, used to categorize the badges in the backend.

5.5 Requirements

This section will introduce some of the requirements that were made, before the development phase was started. The requirements are divided into two separate parts. The section will first introduce the backend requirements, followed by the requirements for the frontend.

5.5.1 Backend

As mentioned in chapter 2.3, determining accurate requirements for the backend development was difficult, but a few directional requirements was listed. Table 5.1, shows the list of the directional requirements.

Requirement	Description
Parse code	Generate a parse tree of both the student code and proposed solution, in order to extract information
Extract metrics	Extract information about the code
Categorize metrics	Group relevant metrics together
Get score	Compare the student code with the proposed solution to get a score
Make badge	Set score requirements and compare that with the score from the previous requirement, if the score matches the requirement make a badge.

Table 5.1: List of backend requirements

After the implementation phase was started, the list evolved, and many additional sub requirements were discovered. Appendix C, will explain in more detail, some of these sub requirements, and the overall implementation of the backend.

5.5.2 Frontend

The frontend requirements were defined more straightforwardly. To solve sub goal SG4, a very simple prototype was made. The prototype displayed information from the backend, and through iterative testing in the prototyping phase sub goal SG4, was broken down to a set of requirements. The requirements were put in the backlog, and sorted by the categories of the MoSCoW method. This was done to help manage the time, and avoid spending too much time on the frontend, which would hinder the work on the backend. Table 5.2 lists the requirements for the frontend view.

Category	Description	Finished	Comment
MH	Badge view	Yes	The badges attained, should be shown in the view
MH	Assignment list view	Yes	An overview of all the assignments delivered should be displayed somewhere in the view.
SH	Detailed badge view	Yes	It should be possible to press a badge, and get a more detailed view and description of the badge.
SH	Detailed assignment view	Yes	A more detailed view of the assignments, and what metrics are extracted from it.
SH	Badge progress view	Yes	In the detailed badge view, it should be possible to see the progress towards the next badge
SH	Info about metric	Yes	It should be possible to show a more detailed description of a metric, and what submetrics define the score.
CH	Recommended assignment	No	It could be possible to recommend assignments for the student, and show this in the view.

Table 5.2: List of frontend requirements

The category-row in table 5.2 describes the MoSCoW categories, where:

MH = must have
SH = should have
CH = could have.

As seen in the "Finished" row, not all of the requirements were completed. The final "CH" requirement was not finished, due to the backend development taking more time than expected.

5.6 High fidelity prototype

This section will give a short description of two screens, of the high fidelity prototype. The web-development tools used when creating this page, are described in appendix G

5.6.1 History page

The history page, is the first page the student encounters when using the solution. It gives an overview of the assignments the student has delivered so far, as well as a summary of those assignments. As seen in Figure5.7, the metric information from the assignment is also displayed.

HISTORY
BADGES

ASSIGNMENT 5

ASSIGNMENT 6

SUMMARY

Assignment 6

Metric	Code quality	Progress towards next badge
General Metrics	4.14	59%
Conditional Statement Metrics	4.38	72%
Inheritance Metrics	10	100%
Exception Metrics	10	100%
Overall Metrics	17	30%

Figure 5.7: History page

Figure 5.7, shows a table of the metric names, a score for code quality, and a percentage revealing the students progress towards the next badge. If the student achieves a badge, the percentage will change to the progress towards the next badge, except for when the gold badge is achieved. The score is a number between one and ten where ten is best. The score is calculated by the formula below.

$$CodeQuality = 10/Scs \quad (5.1)$$

Where Scs , is the Student complexity score, calculated by dividing the student code complexity with the code complexity of the proposed solution. If for instance the *code line* metric is to be calculated, and the student has 500 lines of code, compared to 100 lines in the proposed solution. The complexity would be, $Scs = 500/100 = 5$, and using equation 5.1 the code quality score would be:

$$10/Scs = 10/5 = 2 \quad (5.2)$$

When clicking on one of the metrics in the metrics table of the History page, a notification window appears showing the metric information seen in Figure 5.8.

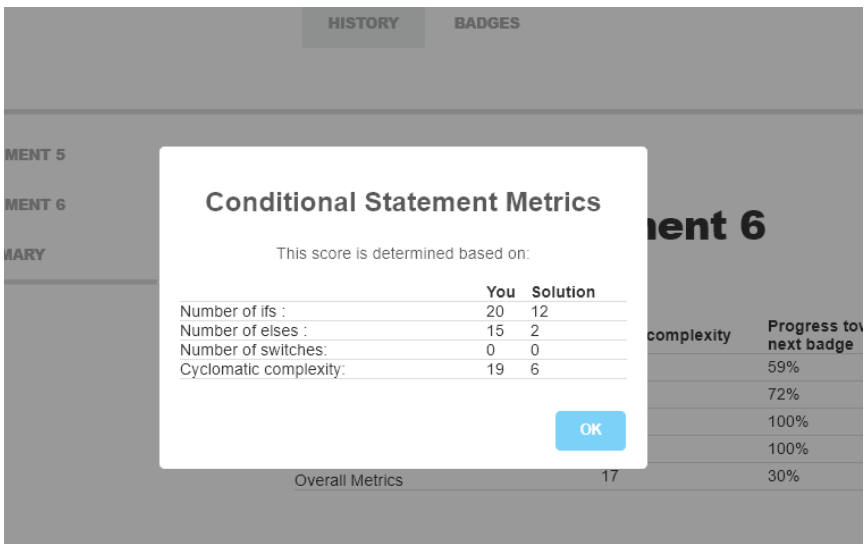


Figure 5.8: Detailed metrics page

The pop up window shows a table of the most important sub metrics composing the metric score. The row marked "You", shows the score of the students code, and the row marked "Solution", shows the score of the proposed solution. This was done in order to give the student a greater insight in what sub-metrics makes up the score.

5.6.2 Badge page

The badge page, is where the student can see obtained badges. As seen in the figure 5.9, the page contains two lists. One of the badges already achieved, and one of the badges still remaining.

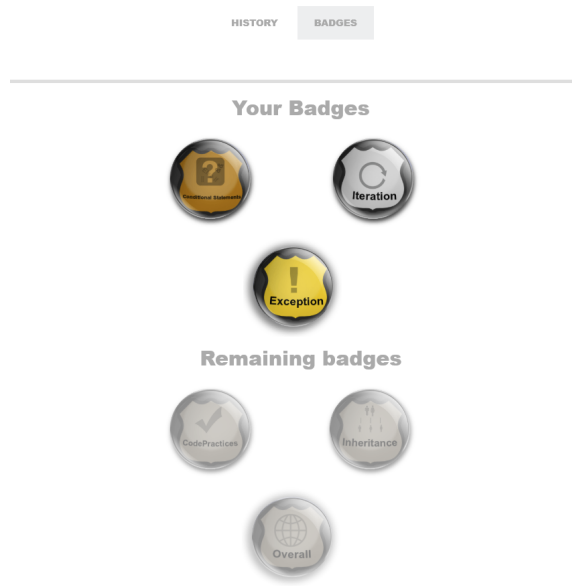


Figure 5.9: Button visibility

Additional badge information can be seen by clicking on one of the badges. The remaining badges are greyed out, but can still be clicked, revealing the progress towards completing the badge. Figure 5.10 shows the detailed badge view, and the "next badge" view.

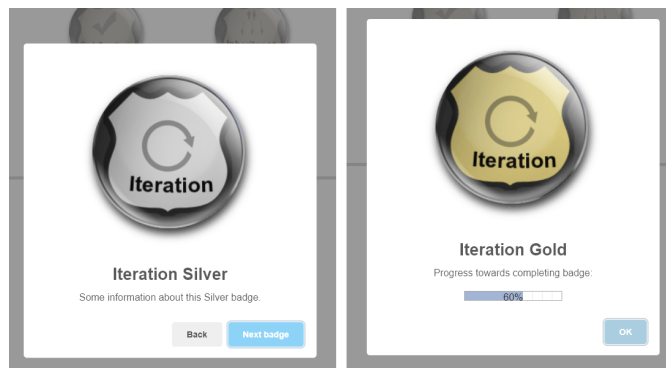


Figure 5.10: Detailed badge information and next badge info

Similar to the detailed description of the remaining badges, the "next badge" view shows the progress towards obtaining the next valor.

Chapter 6

Evaluation

This chapter will present a discussion and evaluation of the research methods, use of gamification, and a brief evaluation of the design phase. Lastly a evaluation of the results will be given, including a discussion of the perceived validity of the results.

6.1 Research Methodology evaluation

The throwaway prototype development with help of the empirical studies, were useful when developing the prototype. It allowed for a exploitative and flexible development, which helped narrow down the uncertainty of the design in the beginning of the project. After deciding which results from the backend to present in the frontend, and how to display these results. The MoSCoW method of prioritization, contributed to making the development phase more straightforward, and made sure the most important requirements were completed first. The *discrete event simulation model*, made it possible to see the development of a student through the semester, and how the overall metrics badge evolved over time. The use of simulation played a significant part in project, because it enabled extraction of results from the prototype, even though the prototype wasn't fully developed.

6.2 Use of Gamification

The literature suggest that the gamification will have a positive impact on the students motivation on the coursework, if they are implemented correctly. The project has a modest approach to the introduction of gamification. Six badge were created, the badges are awarded based on the ratio between code metrics used by the student, and the proposed solution. The choice of using the metric ratio between student code, and the proposed solution, as a quality measurement for awarding badges looked promising. It was however discovered that it was only applicable to some of the metrics.

Chapter 4.5.3, suggests the use of a static variable for the metrics that works poorly

with the ratio reward model. This could be a good replacement, where the static value could be a number, that represents the the total number of recommended uses, of a metric throughout the coursework. Although the students flexibility in solving the assignments can lead to unjust awarding in some cases, the use of code metrics as criteria to award the badges, gives a deeper meaning by representing the students understanding of programming, which may lower the risk of *badgenfreude*.

The badge ladder system was used to introduce gold, silver, and bronze badges, to give a extra depth to the badges. This can help counteracting the problem of micro-scale gamification often being perceived as superficial. It did however also introduce some difficulties, when balancing the badge valors. The gold inheritance and iteration badges were often awarded, even though the student only used one of the badge category's sub-metrics. This was attempted solved, using the method for configuring badge requirements, which removes the metrics that are hardly used. The absence of irrelevant badges, helps balancing the distribution of badges, and prevents students from being awarded undeserved badges.

The badge award system was difficult to configure. The students are free to develop their individual solutions and could chose to avoid using code that is used by the proposed solution, therefore comparing the student code with the proposed solution might not always be the best quality measurement when awarding badges.

6.3 Choice of Metrics

As explained in chapter 8.5.4, the six metric categories are comprised of sub-metrics, that collectively encloses most of the programming elements relevant for the course. These metrics, provides information of which code elements has been used, but little information about the interaction between classes, and the structure of the code. As explained in chapter 8.5.4, this wasn't added due to it exceeding the scope of the course, and how the small code size could impair the feedback of the metrics. It could however be used to include badges inspired by the SOLID principles, that can give the students a light introduction to the principles.

The metrics were also categorized to reduce the complexity of the frontend. There may however be better ways of categorizing the metrics, which would allow for more separation and a higher number of badges, as well as a more even distribution of the sub-metrics. Smaller metric categories increases modularity of the system, that can make it easier for the lecturer to replace, or make changes to metrics as the course evolves. Because the current metrics may award a badge in inheritance as long as the student has made good use of abstract classes, and super expressions, the student could end up forgetting to learn interfaces. A higher modularity, could also give the students a more accurate picture of what they have learned.

6.4 Testing

The testing phase was split into two parts, where the first part was a informal test, used to get a overview of what was expected of the frontend. And the second part handled the user testing, and a survey, to get a more detailed feedback. The informal tests, gave both useful feedback that was used to shape the design of the web page at an early stage, and positive feedback towards the ideas of the prototype. The positivity towards the ideas of the prototype, could indicate that the students would make use of the system if it was available.

In the second part, the user testing provided a more comprehensive feedback from the students, that was useful when making the high fidelity prototype. Even though the feedback was useful, some alterations or additions to the questions, could have increased the surveys' usefulness. The question involving the use of talent trees, could for instance have been replaced with a question asking whether the students would've wanted a talent tree, or a point system, in addition to badges. The use of redundant virtual rewards, with both points, and badges, in Khan Academy. Suggests, that the use of multiple awards is advantageous. Due to the testing being completed at such an early stage, some of the changes made to the metric categories, lead to some of the feedback becoming futile. Therefore, additional tests investigating student understanding of the metrics, should be performed in future.

6.5 Evaluating the results

The results of the simulation on the student-assignments, revealed both a that the ratio between the proposed solution and student code couldn't always be used as reward criteria, and an uneven distribution of gold badges being awarded by some of the metrics. The unevenness of the gold distribution was attempted fixed, using a method for categorizing relevant metrics. The method works by extracting metrics from the proposed solution for each task in the assignment, and disregarding the metrics that are hardly used. Currently the method is configured to allow metric categories that are are used three times or more, because further increase in the number, resulted in the Inheritance metric hardly being relevant for any of the tasks. The number might however need configuring in the future, if the metric categories change, or additional categories are added.

The results also revealed, that extracting metrics from the proposed solution, can give relevant information to the the lecturer. Either by revealing the relationship between metrics, and difficulty, shown in Figure 4.9. Or a by giving a full overview of the metrics used in all the exercises, shown in Figure 8.10. This information can be used, when testing if the exercises cover all the wanted code aspects relevant for the course. Another possible usage, is to use the *cyclomatic complexity* metric, as a measurement when customizing the difficulty level of the exercises, because of the indications that *cyclomatic complexity* is related to task difficulty.

6.6 Validity

6.6.1 Method validity

Because the method uses the proposed solution to extract relevant metrics from each task. The results drawn from the method will continue to be valid, as long as proposed solutions continues being made for new tasks, and the metrics extracted are maintained, to include new additions or changes to the scope of the course.

6.6.2 Validity of the simulation

Due to the simulations being conducted using real student code, and because there is nothing indicating that the code delivered by future students, will result in sudden changes to the metrics. This gives reason to believe that the results obtained from the simulation are valid, as long as the assignments stay roughly the same.

There was however two points of concern, regarding the student code used for the simulation. The first, is that only the code for assignment five, six, eight, and nine was available to use in the simulation, and that the use of all the exercises might lead to different results. The second concern, is that only the code from three of the students were used, due to the time consuming extraction of metrics, and that the small sample size does not represent the average of the class.

6.7 Fulfillment of the project goal

The goal of the thesis was to *create a prototype of a gamified learning system supplementing the coursework in the course Object Oriented Programming. Focusing on the wide span of programming skills significant for the course curriculum.* The prototype created, consists of a frontend web-view, and a backend java platform, that collectively attempt to introduce gamification to the coursework. The java platform introduces badges based on a wide span of code metrics collected from the student code, and the web-view displays these badges to the student. The evaluations above suggests that the all of the Objectives were completed, and that the prototype, once finished, will supplement the coursework, and introduce benefits to both the students, and lecturer. The project will therefore conclude the goal as fulfilled.

Conclusion

In this project, a gamified prototype a learning system was developed to supplement the coursework of Object-oriented programming. Code metrics, use of gamification and different reward models were reviewed and adapted to TDT4100 when developing the prototype. The prototype consists of two parts, a Java backend that use code metrics to award badges, and a frontend web-view to display these badges.

The frontend was designed based on feedback received through iterative low level testing on previous students, and a more through test that explored the usability and content of the system. The results revealed a positive attitude towards the use of badges as well as an overall positive attitude towards the system.

The badges are awarded by the backend based on six metric categories extracted from the code. First the student metrics are compared to the metrics of the proposed solution, then a ratio score is calculated based on the difference between them and finally the score is compared to a list of requirements that determines if a badge should be awarded or not. The list of requirements contains three values for each metric category, where each value represents the requirement for either the bronze, silver or gold badge.

After the backend was created a simulation was conducted where the student code from four assignments was run through the backend. The results revealed a uneven distribution of which valors were awarded by which metric category, and it was discovered that some of the badges were awarded based on a single metric usage. This problem was fixed by introducing a method for configuring relevant metrics, the method ran each task of the proposed solution through the backend and listed which metrics were relevant for that specific task. It was also discovered that running the proposed solution through the backend, could provide information about the assignments that is useful for the educators.

This thesis has shown that the gamified prototype could supplement the coursework by introducing badges that can help motivate students, and by providing metric information to both the students and educators that can be used get a overview of, individual progress or the coursework in general. Whether the prototype will provide a learning benefit for the students cannot be determined without further testing, it does however provide foundation that can be further developed.

Future development

This chapter will discuss the future development of the prototype. First the possible additions to the backend will be discussed, followed by some ideas that could improve the frontend.

8.1 Backend

8.1.1 Supplementing the badges

Currently the awards of the the system are limited to six badges awarded in-game. Because the rewards are limited to the game, they may not exploit the full motivational potential that meta-game and out-game rewards could introduce. One option could be to have the badges count for a small percentage of the grade. The number of badges could also be increased, one option is to introduce milestone badges that are awarded students who for instance have obtained five gold medals in the same category. Another option is to introduce time-badges that are awarded based time consumption instead of code metrics. The *ex-files* that contains the student code explained in appendix C also contains information of how much time the students spend on the assignment, which would be a fun contribution to the badges.

8.1.2 Improving the method for configuring badge requirements

The method explained in chapter 4.6.1 explains how the relevant metrics can be filtered to only include badges that are used multiple times. This whole process could however be completely automated by having the java-class that filters metrics automatically insert the results in the config file.

Currently there is no method for configuring the score value requirements for the badges. But if the score values are summarized for each of the students that completed the coursework last year, the score requirement for gold could be set to where the top 10%

are awarded the badge. As long as there are no major changes to the coursework or the future students starts programming differently, this would make it possible to somewhat distribute the badge valors as wanted.

8.1.3 Student task recommendation

The list of used and unused metrics created by the overall metric badge, could be used to give recommendations of which tasks a student should do. If metrics are extracted from each of the proposed solutions, and the list of used metrics are mapped to the task names. This list could be matched with the used metrics list of the students to maximize the number of metrics exiting in both lists. That way the student would be recommended the task containing the maximum number of unused metrics.

8.1.4 New applications of the simulation data

If the average code metric results for each of the students are organized in a straightforward manner over multiple years, this information could be used to see what impacts changes to teaching, syllabus or assignments have on the students programming. One example could be to introduce lambda expressions early in the course, and see if it results in more lambda expressions being used by the students. If this prototype is ever completed and introduced to the course, the information could even be used to determine if the efficiency of the prototype.

8.2 Frontend

8.2.1 Imprpovements to the current design

The current web view has a lot of numbers and little graphics due to the removal of the metric spider diagrams, therefore a new way of illustrating the metrics should be introduced. One option could be to introduce a pie chart where each piece represents a metric category and the number of pieces could vary depending on the number of relevant metrics. The piece could also work as a loading bar, filling up the piece as the score gets better.

Depending on how frequently the web view is used, a possible improvement would be to add animations or push notifications to inform the students when they receive a new badge. This would increase the students attention towards the badges, which could result in increased interest of the badges. The article by Lassi Haaranen and Korhonen[28] recommends enabling the sharing of badges to make it possible for the students to compete against each other, this could be done by introducing the sharing system used by Khan Academy where the users can choose to share their badges in their profile page.

8.2.2 Leaderboards

Even though chapter 3.2.4 discussed some negative implications of leaderboards, that was mostly due to the motivational loss that could arise for the users with the lowest scores. The chapter also introduced multiple advantages of leaderboards. Therefore a leaderboard could be included that only show the top 10% in the class, that way there is no way for the users with the lowest scores to know that they have the lowest score. Because the award system currently lacks a point system, a simple one could be introduced that gives points for every badge received.

8.2.3 Additional views

The statistics extracted from the backend could be relevant for more parties than just the students, therefore additional views should be added showing information relevant for the student assistants and lecturer. The information shown to the lecturer could be abstracted to only display how many students have used which metrics, while the student assistants views could be more specified and show additional details about each student. Appendix H shows some suggested to tables that could be used to display the information in the different views.

Bibliography

- [1] Javaparserfor processing java code. <https://javaparser.org/about/>. Accessed: 2018-03-20.
- [2] Tdt4100 - objektorientert programmering. <https://www.ntnu.no/studier/emner/TDT4100#tab=omEmnet>. Accessed: 2018-04-20.
- [3] Introduction to knockout. <http://knockoutjs.com/documentation/introduction.html>,. Accessed: 2018-03-22.
- [4] Knockout.js random notes (1). <http://www.programering.com/a/MTN%gTMwATQ.html>,. Accessed: 2018-03-22.
- [5] C.M. Stracke A. Antonaci, R. Klemke and M. Specht. Identifying game elements suitable for moocs. 2017.
- [6] Christian M. Stracke Alessandra Antonaci, Roland Klemke and Marcus Specht. Identifying game elements suitable for moocs, page 355. 2017.
- [7] Robert Axelrod. Advancing the art of simulation in the social sciences. 1997.
- [8] Nick Babich. Prototyping 101: The difference between low-fidelity and high-fidelity prototypes and when to use each. <https://theblog.adobe.com/prototyping-difference-low-fidelity-high-fidelity-prototypes-use/>, 2017. Accessed: 2018-04-26.
- [9] V. S. Bidve* and P. Sarasu. Coupling measures and its impact on object-oriented software quality. 2016.
- [10] Aaron B. Binkley and Stephen R. Schach. Validation of the coupling dependency metric as a predictor of run-time failures and maintenance measures. 1998.
- [11] Jen-Wei Chang and Hung-Yu Wei. Exploring engaging gamification mechanics in massive online open courses. 2015.
- [12] R. Ryan Nelson Dennis A. Adams and Peter A. Todd. Perceived usefulness, ease of use, and usage of information technology: A replication. 1992.

-
- [13] Kevin Dooley. Simulation research methods. 2002.
- [14] R. Koestner E. L. Deci and R. M. Ryan. A meta-analytic review of experiments examining the effects of extrinsic rewards on intrinsic motivation. 1999.
- [15] International Organization for Standardization. Iso/iec 25010:2011. 2017.
- [16] G.K. Gill and C.F. Kemerer. Cyclomatic complexity density and software maintenance productivity. 1991.
- [17] J. Hamari and V. Eranti. Framework for designing and evaluating game achievements. 2011.
- [18] Burak Yahsi Hans-Christian Pfohl and Tamer Kurnaz. The impact of industry 4.0 on the supply chain. 2015.
- [19] Carol Sansone Judith M. Harakiewicz. *Intrinsic and Extrinsic Motivation the search for optimal motivation and performance*. McGraw Hill, 2000.
- [20] Duncan Haughey. Moscow method. 2014.
- [21] Bipin Joshi. Overview of solid principles and design patterns. 2016.
- [22] Jesper Juul. Half-real: video games between real rules and fictional worlds, page 6-7. 2005.
- [23] KhanAcademy. Khanacademy. <https://www.khanacademy.org/>, . Accessed: 2018-05-16.
- [24] KhanAcademy. Knowledgemap. <https://www.khanacademy.org/exercisedashboard>, . Accessed: 2018-05-16.
- [25] KhanAcademy. Profilepage. https://www.khanacademy.org/profile/kaid_1074050811436157338819075/, . Accessed: 2018-05-16.
- [26] T.M. Khoshgoftaar and J.C Munson. The lines of code metric as a predictor of program faults: a critical analysis. 1990.
- [27] Ah-Fur Lai. A study of constructing k-12 programming competence indicators. 2017.
- [28] Lasse Hakulinen Lassi Haaranen, Petri Ihanola and Ari Korhonen. How (not) to introduce badges to online exercises. 2014.
- [29] Averill M. Law and David Kelton. *SIMULATION MODELING AND ANALYSIS*. 1991.
- [30] Code Analysis Team log. Code metrics as check-in policy. <https://blogs.msdn.microsoft.com/codeanalysis/2007/11/16/code-metrics-as-check-in-policy/>, 2007. Accessed: 2018-04-24.
- [31] Thomas J. MacCabe. Structured testing. 1983.

-
- [32] Robert C. Martin. *Clean code: A handbook of agile software craftsmanship*. 2008.
- [33] Gary McLean. *Adaptive code via c: Agile coding with design patterns and solid principles*. 2014.
- [34] microsoftDeveloperNetwork. Code metrics values. <https://msdn.microsoft.com/en-us/library/bb385914.aspx>. Accessed: 2018-04-21.
- [35] Sanjay Misra and I. Akman. *Weighted class complexity: A measure of complexity for object oriented system*. 2008.
- [36] Briana B. Morrison and Betsy DiSalvo. *Khan academy gamifies computer science*. 2014.
- [37] Brian Goetz Oracle. Updates to the original jsr. <https://www.jcp.org/en/jsr/proposalDetails?id=335>. Accessed: 2018-05-23.
- [38] Satwinder Singh Pavitdeep Singh and Jatinder Kaur. *Tool for generating code metrics for c source code using abstract syntax tree technique*. 2013.
- [39] Jon Pearce. *Measuring cohesion*. 2016.
- [40] Richard S. Rosenberg. *The Social Impact of Computers*. Academic Press, 1986.
- [41] Stephen R. Schach. *Object-oriented and classical software engineering*. 2007.
- [42] Carolyn B. Seaman. *Qualitative methods in empirical studies of software engineering*. 1999.
- [43] Rilla Khaled Sebastian Deterding, Dan Dixon and Lennart Nacke. From game design elements to gamefulness: Defining “gamification”, page 10. 2011.
- [44] Barbara Bichelmeyer Steven D. Tripp. *Rapid prototyping: An alternative instructional design strategy*. 1990.
- [45] C. Synder. *Paper prototyping: The fast and easy way to design and refine user interfaces*. 2003.
- [46] the Federal Aviation Administration. *Faa system safety handbook*. 2000.
- [47] AMY TODD. *Why gamification is bullshit malarkey*. 2017.
- [48] Hallvard Trætteberg. *Objektorientert programmering*. <https://www.ntnu.no/wiki/display/tdt4100/Objektorientert+programmering>. Accessed: 2018-04-21.
- [49] Lincoln C. Wood and Torsten Reiners. *Gamification in logistics and supply chain education: Extending active learning*. 2012.
- [50] Gabe Zichermann and Christopher Cunningham. *Gamification by design*. 2011.

Appendix

8.3 Appendix A - Test: Survey

Below is the full survey with both questions and answers. Used as part of the testing phase.

Brukerundersøkelse 1 Alle endringer er lagret i Disk

SPØRSMÅL SVAR **3**

3 svar + ⋮

SAMMENDRAG INDIVIDUELL Tar imot svar

Do you understand the detailed description of the "Iterators" metric?
3 svar

Yes!

If I understand what iteration is based on the text? Yes.

Yes, but could maybe use a bit more information.

Would you like to have a more detailed view of the control flow statements ? (While/For/Recursion)
3 svar

100%

● Yes
● No

Would you choose a different name than "Iterators"? if yes, which name?

1 svar

I think iterators is a good and concise name, although people who are new to coding may prefer a simpler, less scary name (like "Looping operators")

Do you understand the detailed description of the "Clean code" metric?

3 svar

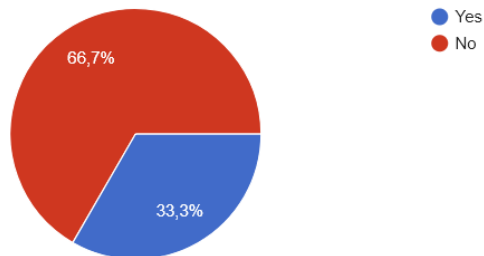
Yes, that's a good description although people may confuse it with the Clean Code development process.

Yes

Does the solution contain 9 naming convention errors? And are errors rewarded? Other than that it was very easy to understand.

Would you like to have a more detailed view of the metrics that determine the Clean code score? (Open/Closed principle, Java naming convention error, etc)

3 svar



Would you choose a different name than “Clean code”? if yes, which name?

1 svar

Maybe something like “Code readability” or something like that.

Can you think of another metric that would be fitting to rate code quality in TDT4100/Object oriented programming?

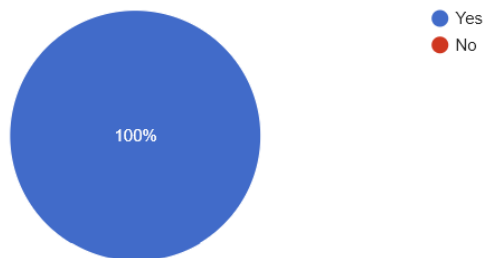
2 svar

I'm assuming correctness is already a metric. Maybe something related to error handling and robustness, if this is already not captured by the correctness tests. Maybe metrics for memory usage and execution time to encourage efficient programming, although this might be overkill for an introductory programming course.

Meaningful variable names.

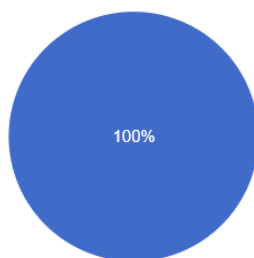
Do you understand the detailed description of the “Length” badge?

3 svar



Do you understand the detailed description of the "Iterator" badge?

3 svar

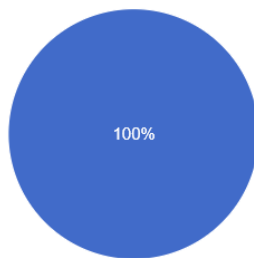


● Yes
● No

Would you like to see the progress towards the next badge level? (for example gold/platinum badge)



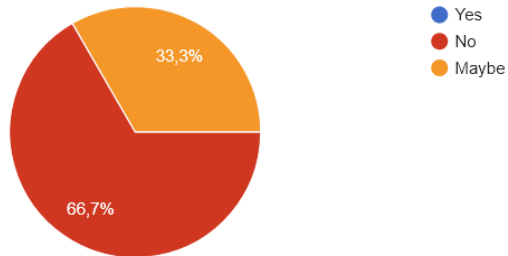
3 svar



● Yes
● No

Would you prefer a talent tree over badges?

3 svar



Do you miss any functionality in the solution? Or do you have any other comments?

2 svar

I'd like some more information about how the score is calculated. For example, for the iterator badge is the score equal to the sum of the number of times I have used various iterator types? The length badge has some explanation, which is good. For the various metrics, I assume the figure is a placeholder. If not, I am unsure what information is conveyed here.

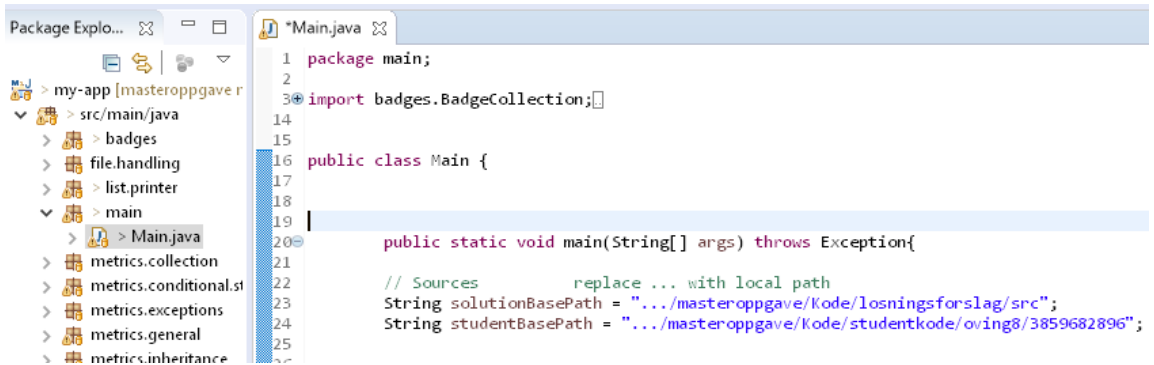
Buttons don't look like buttons, hard to know if it's just text or something you can actually press. And I don't really get what the graph is illustrating.

8.4 Appendix B - Backend printed

This section shows the different prints from the backend, that was used for the simulation in chapter 4.

8.4.1 Change local path

Before the code can be run, the local path must be changed in the top of the Main.java class.



```
1 package main;
2
3 import badges.BadgeCollection;
4
5
6
7
8
9
10
11
12
13
14
15
16 public class Main {
17
18
19
20     public static void main(String[] args) throws Exception{
21
22         // Sources         replace ... with local path
23         String solutionBasePath = ".../masteroppgave/Kode/losningsforslag/src";
24         String studentBasePath = ".../masteroppgave/Kode/studentkode/oving8/3859682896";
25
26     }
```

8.4.2 Code run for the results to be printed

The following code is run by the Main.java class in order to see the prints shown below.

```
// Solution and student name generator - filenameGenerator = new SolutionAndStudentFilenameGenerator(studentBasePath, solution
StudentPathGenerator studentPath = new StudentPathGenerator(studentBasePath);

//Student metrics
GetStudentCodeSource studentCodeList = new GetStudentCodeSource(studentPath.pathList, studentBasePath);
MetricsCollectionList studentMetricsCollectionList = new MetricsCollectionList(studentCodeList.studentCodeList);
MetricsSummary studentMetricsSummary = new MetricsSummary(studentMetricsCollectionList);
System.out.println("STUDENT CODE METRICS: \n" + studentMetricsSummary + "\n\n");
//System.out.println("Student code metrics are calculated from these files: \n" +studentBasePath+"\n"+ studentPath.pathList);

//Solution Metrics
FilteredSolutionPathGenerator filteredSolutionFileList = new FilteredSolutionPathGenerator(studentCodeList.studentCodePackagell
MetricsCollectionList solutionMetricsCollectionList = new MetricsCollectionList(filteredSolutionFileList.solutionFileLamelist,
MetricsSummary solutionMetricsSummary = new MetricsSummary(solutionMetricsCollectionList);
System.out.println("SOLUTION CODE METRICS: \n" + solutionMetricsSummary);
//System.out.println("SOLUTION METRICS ARE CALCULATED FROM \n" +solutionBasePath +"\n and this list: " + filteredSolutionFile

//PropertiesFile
ReadPropertiesXmlFile xmlProperties = new ReadPropertiesXmlFile("C:/Users/Syver/masteroppgave/MavenJavaProject/my-app/propert
ReadXMLFile xmlProperties = new ReadXMLFile("C:/Users/Syver/masteroppgave/MavenJavaProject/my-app/properties.xml");

//Badges
BadgeCollection badgeCollection = new BadgeCollection(studentMetricsSummary, solutionMetricsSummary, xmlProperties, "5-1");
System.out.println(badgeCollection);
```

8.4.3 Student metrics print

Below are the results printed after running the backend solution on student (Hashed ID) 3859682896's delivery of a assignment 8.

STUDENT CODE METRICS:

MetricsSummary:

ConditionalStatementMetrics

 Cyclomatic Complexity: 4

 Number of Ifs: 6

 Operators:

 Number of Equals (==): 1

 Number of Not Equals (!=): 1

 Number of Less than (<): -1

 Number of Less or equal (<=): 2

ExceptionMetrics

 Number of Throw statements: 1

 Exceptions:

 Number of IllegalArgumentExceptions: 1

GeneralMetrics:

 Total number of code lines: 98

 Total number of variables declared: 7

 Total number of method calls: 21

 Number of unaryOperators

 PlusOne (++): 4

 BooleanInverter (!): 1

 Uweighted Methods per Class:

IteratorMetricsSummary

 Total amount of Foreach loops: 5

 Total amount of For loops: 5

 Total amount of While loops: 5

IterableMetricsSummary

InheritanceMetricsSummary

 Total amount of abstract class or interface inheritance: 2

8.4.4 Solution metrics print

Below are the metrics extracted from the proposed solution from the tasks student 3859682896 completed from assignment 9.

```
SOLUTION CODE METRICS:
MetricsSummary:

ConditionalStatementMetrics
  Cyclomatic Complexity: 2
  Number of Ifs: 2
  Operators:
    Number of Not Equals (!=): 1

ExceptionMetrics
  Exceptions:

GeneralMetrics:
  Total number of code lines: 42
  Total number of variables declared: 1
  Total number of method calls: 11
  Number of unaryOperators
    BooleanInverter (!): 1
  Wheighted Methods per Class:

IteratorMetricsSummary
  Total amount of Foreach loops: 1
  Total amount of For loops: 1
  Total amount of While loops: 1

IterableMetricsSummary

InheritanceMetricsSummary
  Total amount of abstract class or interface inheritance: 1
```

8.4.5 Student badge print

The following shows what the BadgeCollection class prints after calculating the scores calculated using the student metrics and proposed solution metrics.

```
Badge collection
ConditionalStatementBadge
    Number of If's ratio: 1,4
    Number of operators used ratio: 1,0
    Student Cyclomatic complexity: 4,0
    Cyclomatic complexity ratio: 1,0

    The total score is: 1,13
    The Valor is: SILVER
        Bronze score limit: 3.0
        Silver score limit: 2.0
        Gold score limit: 1.0

GeneralMetricsBadge
    Number of lines ratio: 1,14
    Number of methods used ratio: 1,67
    Number of variableDeclarations ratio: 1,22

    Weighted methods per class Ratio: 1,0

    The total score is: 1,26
    The Valor is: SILVER
        Bronze score limit: 3.0
        Silver score limit: 2.0
        Gold score limit: 1.0

IterationMetricsBadge
    Number of iterators used ratio: 1,0
    Number of iterables used ratio: ?

    The total score is: 1,0
    The Valor is: GOLD
        Bronze score limit: 5.0
        Silver score limit: 2.5
        Gold score limit: 1.0
```

ExceptionMetricsBadge

Number of try ratio: ?
Number of catch ratio: ?
Number of throw ratio: 1,0

Number of [illegalArgumentException](#) ratio: ?
The total score is: 1,0
The Valor is: GOLD
 Bronze score limit: 3.6
 Silver score limit: 2.4
 Gold score limit: 1.2

OverallMetricsBadge

Unused Metrics:
 For
 Lambda
 ArrayInit
 ArrayCreation
 ArrayAccess
 Switches
 BooleanOperatorFunction
 BooleanOperatorAnd
 BooleanOperatorOr
 BooleanOperatorLessThan
 BooleanOperatorNotEqual
 BooleanOperatorEqual
 BooleanOperatorLessOrEqual
 UnaryOperatorAddition
 UnaryOperatorSubtraction
 UnaryOperatorPlusOne
 UnaryOperatorMinusOne
 UnaryOperatorMultiply
 UnaryOperatorDivide
 UnaryOperatorRemains
 UnaryOperatorBooleanInverter
 PublicFieldDeclaration
 CatchStatements
 TryStatements
 IllegalArgumentExcpetions

Metrics used/Metrics total: 9 / 34
The total score is: ,26
Valor: NONE
 Bronze score limit: 0.5
 Silver score limit: 0.8
 Gold score limit: 0.95

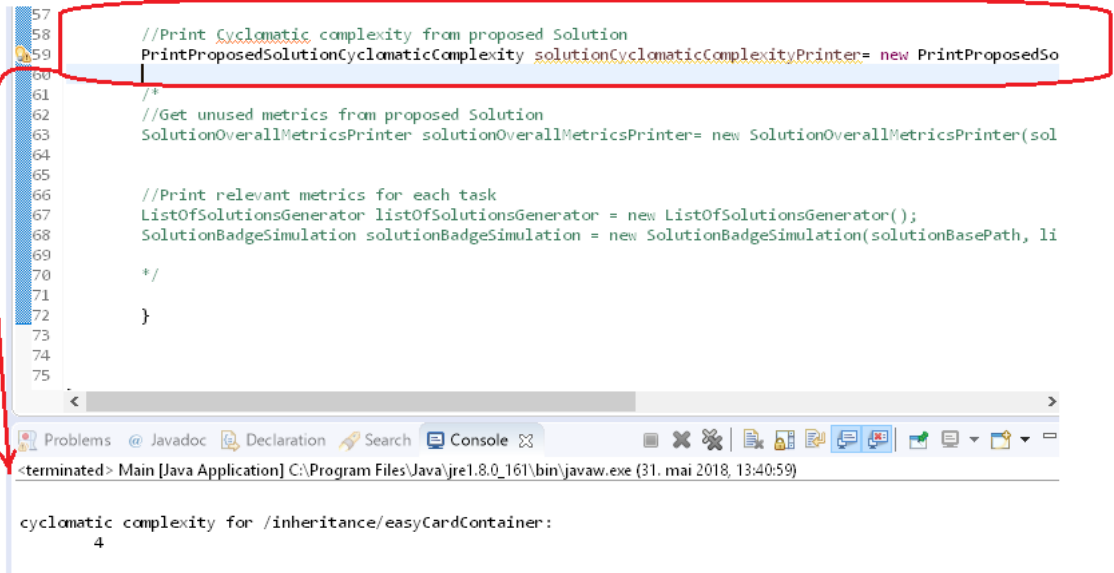
InheritanceMetricsBadge

Number of interface or abstract classes implemented ratio: 1,0
Number of super expressions used ratio: ,0

The total score is: ,5
The Valor is: GOLD
 Bronze score limit: 4.0
 Silver score limit: 3.0
 Gold score limit: 2.0

8.4.6 Cyclomatic Complexity for the proposed solution

The picture below shows what part of the Main.java is run in order to see the Cyclomatic complexity of the proposed solution. The printed results are only for one task, this was therefore completed on each individual task to get the results for figure 4.8.



```
57
58 //Print Cyclomatic complexity from proposed Solution
59 PrintProposedSolutionCyclomaticComplexity solutionCyclomaticComplexityPrinter= new PrintProposedSo
60
61 /*
62 //Get unused metrics from proposed Solution
63 SolutionOverallMetricsPrinter solutionOverallMetricsPrinter= new SolutionOverallMetricsPrinter(sol
64
65
66 //Print relevant metrics for each task
67 ListOfSolutionsGenerator listOfSolutionsGenerator = new ListOfSolutionsGenerator();
68 SolutionBadgeSimulation solutionBadgeSimulation = new SolutionBadgeSimulation(solutionBasePath, li
69
70 */
71
72 }
73
74
75
```

<terminated> Main [Java Application] C:\Program Files\Java\jre1.8.0_161\bin\javaw.exe (31. mai 2018, 13:40:59)

cyclomatic complexity for /inheritance/easyCardContainer:
4

8.4.7 Unused Metrics for the proposed solution

The figure below shows the metrics unused by assignment five, six, eight and nine. In addition to which part of the Main.java produces the results.

```
54
55
56 //Get unused metrics from proposed Solution
57 SolutionOverallMetricsPrinter solutionOverallMetricsPrinter= new SolutionOverallMetricsPrinter(solu
58
59 /*
60 //Print relevant metrics for each task
61 ListOfSolutionsGenerator listOfSolutionsGenerator = new ListOfSolutionsGenerator();
62 SolutionBadgeSimulation solutionBadgeSimulation = new SolutionBadgeSimulation(solutionBasePath, lis
63 */
64 }
65
66 }
67
68
69
```

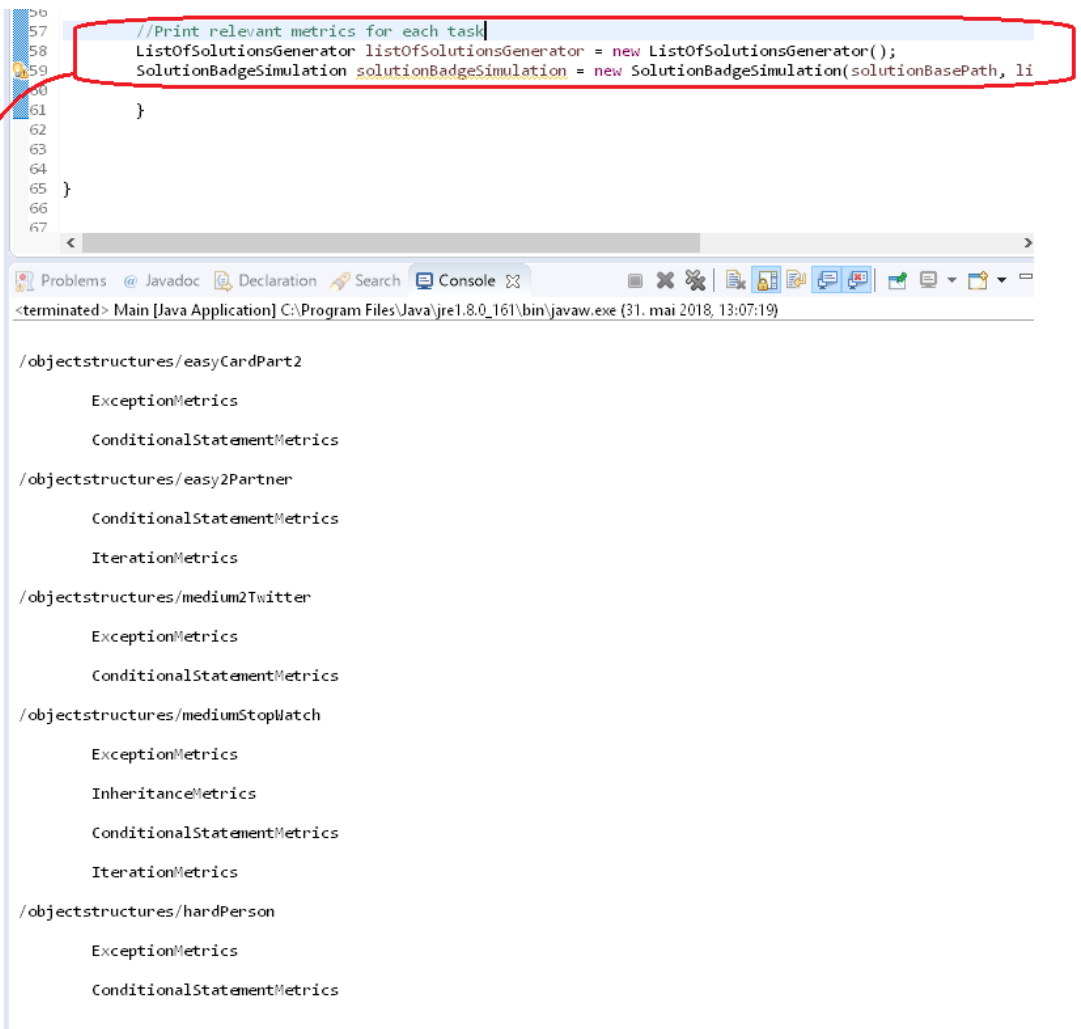
Problems @ Javadoc Declaration Search Console

<terminated> Main [Java Application] C:\Program Files\Java\jre1.8.0_161\bin\javaw.exe (31. mai 2018, 13:15:24)

Unused metrics for proposed solution:
Lambda
BooleanOperatorLessThan
BooleanOperatorMoreThan
UnaryOperatorAddition
UnaryOperatorMinusOne
UnaryOperatorMultiply
UnaryOperatorDivide
UnaryOperatorRemains

8.4.8 Relevant Metrics for each task

Below are the results produced by the SolutionBadgeSimulation.java class, that prints which metrics are relevant for which task. The results printed below are the metrics relevant for the tasks in assignment five.



```
56
57
58 //Print relevant metrics for each task
59 ListOfSolutionsGenerator listOfSolutionsGenerator = new ListOfSolutionsGenerator();
60 SolutionBadgeSimulation solutionBadgeSimulation = new SolutionBadgeSimulation(solutionBasePath, li
61
62 }
63
64 }
65 }
66
67
```

<terminated> Main [Java Application] C:\Program Files\Java\jre1.8.0_161\bin\javaw.exe (31. mai 2018, 13:07:19)

```
/objectstructures/easyCardPart2
    ExceptionMetrics
    ConditionalStatementMetrics

/objectstructures/easy2Partner
    ConditionalStatementMetrics
    IterationMetrics

/objectstructures/medium2Twitter
    ExceptionMetrics
    ConditionalStatementMetrics

/objectstructures/mediumStopWatch
    ExceptionMetrics
    InheritanceMetrics
    ConditionalStatementMetrics
    IterationMetrics

/objectstructures/hardPerson
    ExceptionMetrics
    ConditionalStatementMetrics
```

8.5 Appendix C - Backend Implementation

This section will give a description of how the backend was implemented. First the different technologies will be introduced, followed by an overall description of the solution before some parts of the solution are explained in more detail.

8.5.1 Git

All the code for both the frontend and backend as well as the summarized results in excel, can be found by cloning the git repository:

<https://github.com/ssbolsta/masteroppgave.git>

8.5.2 Technologies

This section gives a short explanation of the most important tools and technologies used in the Java part of the solution.

Javaparser

The Javaparser is a library containing a set of tools. Mainly to parse javacode, but it can also be used to analyze or generate code[1]. In the solution, the Javaparser is used for analyzing and parsing source code. The parser recognizes the different elements of the source code, which is used to generate an Abstract Syntax Tree (AST). The figure below illustrates what a simplified section of this parse tree could look like.

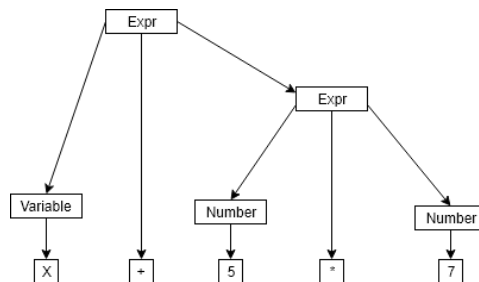


Figure 8.1: Example of a parse tree

To extract information from the code the analyzer uses the parsed code, Each node in the AST is checked, and information about that node is obtained, in the example above you could for instance count the operators "+" and "*" or the number of expressions "2". To extract more detailed information about the code, another tool is needed.

SymbolSolver

The SymbolSolver is an integration to the JavaParser that also uses the AST as an input, it can however return a more elaborate report of the code[1]. In figure 8.1 the Javaparsers analyzer would only be able to see the variable "X" as a name, where as the Symbol Solver could determine if it was a parameter, a local variable or a field.

8.5.3 Overview

The solution consists of a collection of classes used to read student assignment files, calculate different metrics from those files and give a badge based on those scores. The figure below, shows a simplified class diagram of the most important classes in the solution, the full diagram can be found in Appendix C

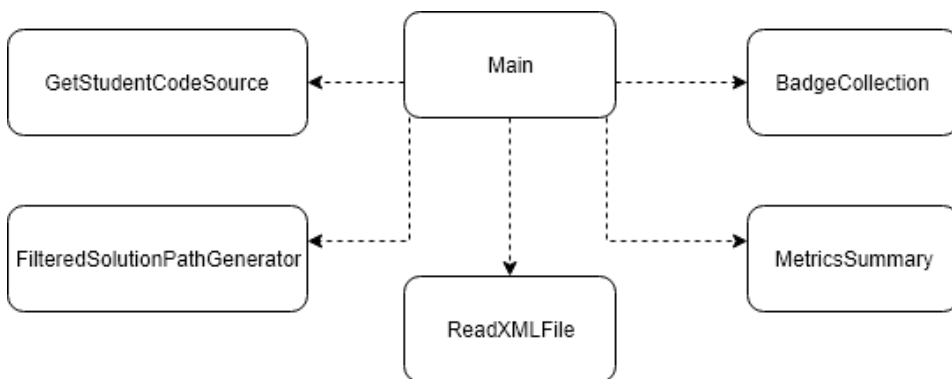


Figure 8.2: Class Overview

As seen in Figure 8.2 the solution is built up by five main parts. First the *GetStudentCodeSource* class retrieve the code from a student assignment *.ex* file, then the metrics from each of the classes are extracted using the *MetricsCollectionList* class.

Before the metrics are extracted from the proposed solution, the solution code is filtered in the *FilteredSolutionPathGenerator* class to only include the tasks completed by the student. Afterwards each of the metricCollections in the *MetricCollectionList* are summarized in the *MetricsSummary* class, this is done both for the proposed solution and the student code.

Finally the *ReadXMLFile* class is used to read the badge valor properties from a config file, before the badges are determined in the *BadgeCollection* class. To determine what badges are awarded, the metrics from the student code are compared to the metrics from the proposed solution and if the ratio is lower than the badge valor the badge is awarded.

File handling

Before the backend can compare the metrics of the student with the proposed solution the student, the proposed solution must first be filtered to only accept the java classes with the same the package as the tasks of the proposed solution. The student code it extracted

from a .ex file, which contains information about how the assignment was completed. This includes information of the time spent solving the assignment, when it was solved and a step by step overview of the work from start to finish. The figure below shows the basic structure of the .ex file.

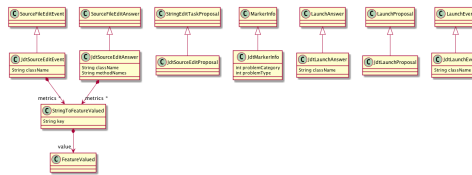


Figure 8.3: JDT-model

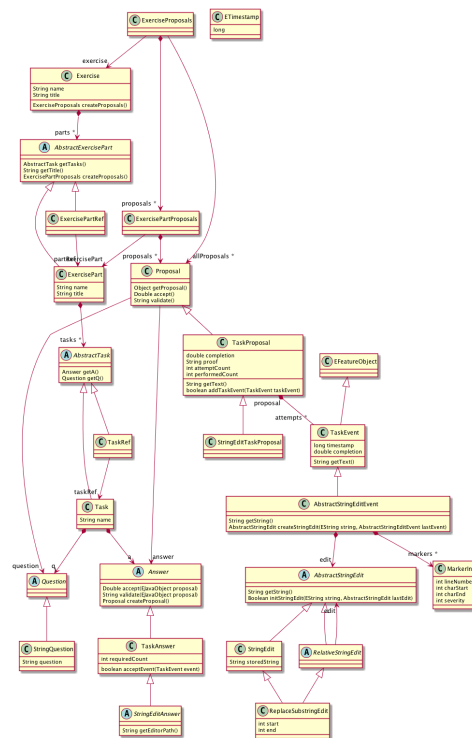


Figure 8.4: Exercise model

The solution currently only retrieves the last element of step by step overview, to obtain all the code delivered.

8.5.4 Metrics

This section will give a more detailed explanation of the General metrics and Overall metrics categories were created. The table below shows how the metric categories are distributed between the three parts of the programming curriculum.

java standard library
General Metrics Overall Metrics Code construction
Inheritance Metrics Exception Metrics Control flow
Iteration Metrics Conditional Statements Metrics

Table 8.1: Programming curriculum handled by which metric

General Metrics

The General Metrics category explores aspects of the code similar to the Code organization in NDepend tool explained in chapter 3.4.1. This section will give an overview of what sub metrics are extracted for *GeneralMetrics*, and how the score is calculated. The *GeneralMetrics* class uses three classes shown in the figure below to extract the relevant metrics.

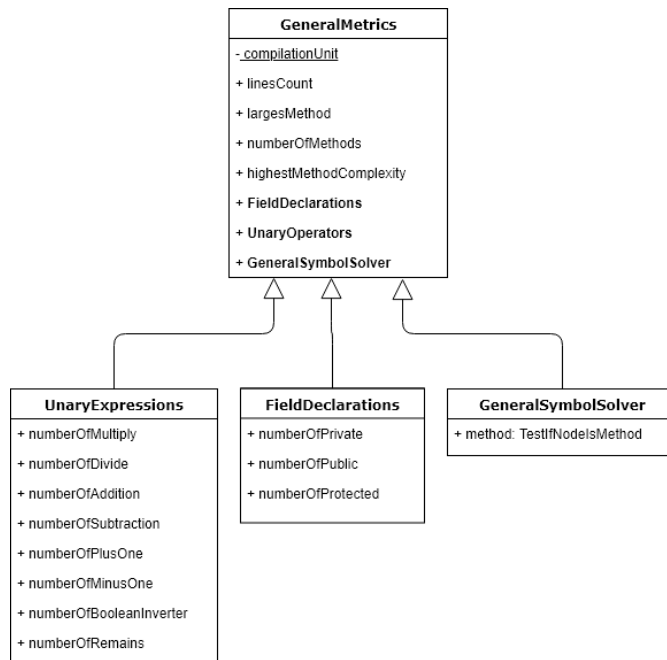


Figure 8.5: Class Overview

The General metrics class uses the UnaryExpression class and FieldDeclaration class, to extract simple information about the code. This information is later used in the OverallMetrics class, explained in the next section. The GeneralSymbolSolver class uses the Java Symbol solver to distinguish which parts of the code are methods. These methods, in addition to the number of code lines, which are counted in the GeneralMetrics class are important when deciding the score of the Metric. The list below shows which sub-metrics are extracted.

Number of code lines
Weighted methods per Class
Number of unary operators
Field declarations

Table 8.2: GeneralMetrics' Sub-Metrics

The *GeneralMetrics* score is determined by the top two metrics in the table, *Number of code lines* and *Weighted methods per class*. The Number of code lines was found to be a relevant sub-metric, because short and concise code gives an indication of lower code maintenance costs[10]. It might also be useful for the students to see how many lines of code they use compared to the proposed solution.

Weighted methods per class is used as a sub-metric because it has good qualities when evaluating code[35]. To calculate the value for *Weighted methods per class* the methods found in the GeneralSymbolSolver class are used. The Cyclomatic complexity is calculated for each method using equation 3.1, before the total weighted class complexity is calculated with equation 3.4 as explained in Chapter 3.4.2. A high complexity score indicates a higher cost in maintaining the code[10], therefore a low complexity score will give a high quality score for this metric.

Overall Metrics

The main idea behind the *Overall Metrics*, is to give the student an overview of which sub-metrics have been used throughout the assignment. The overall metrics score is not based on the number of usages of each sub-metric, but rather the number of sub-metric used at all. The figure below, gives an example of how the detailed description of the *Overall metrics* is shown to the student in the web-view.

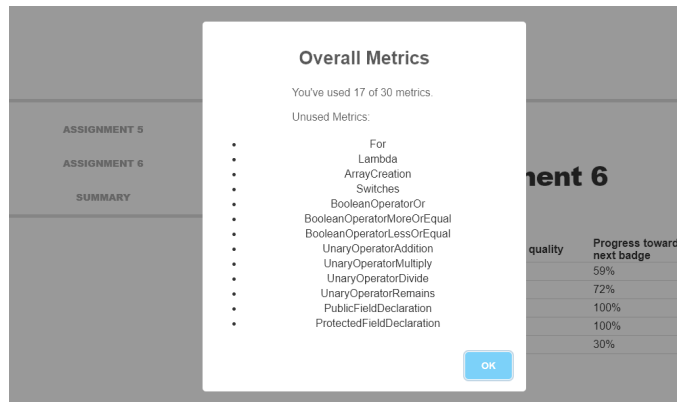


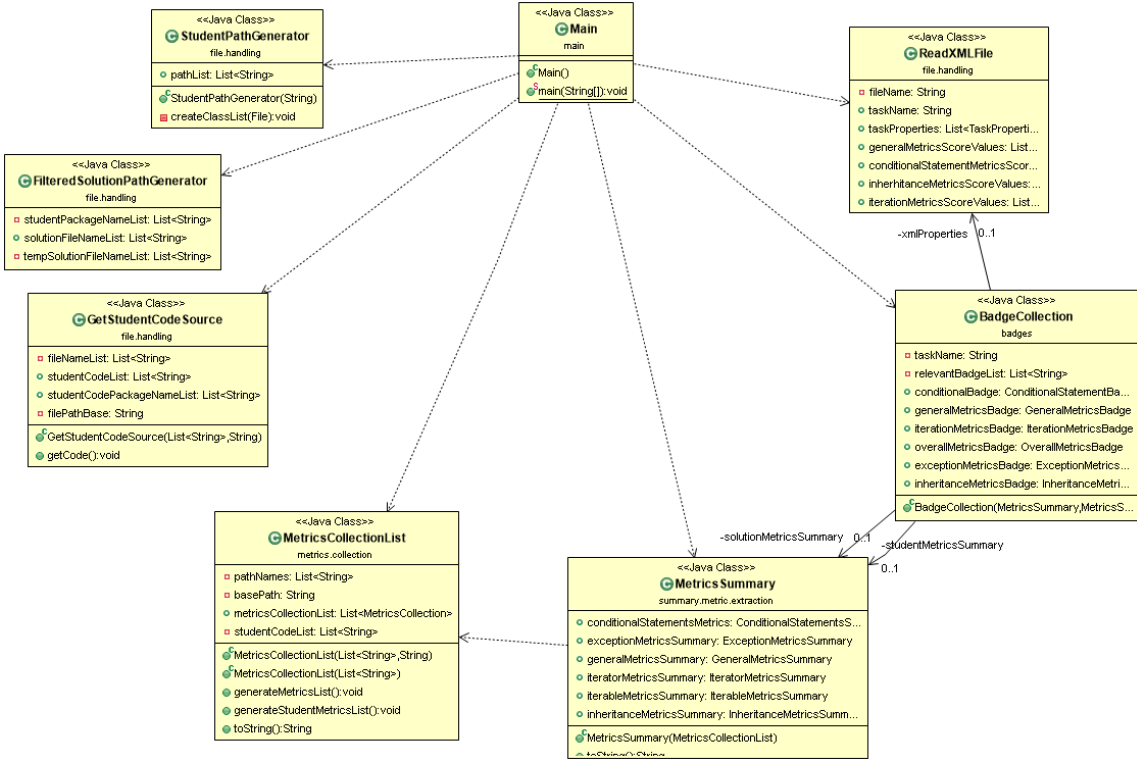
Figure 8.6: View of Overall metrics in the frontend

Figure 8.6 displays an example of a list of unused sub-metrics. The sub-metrics used in *OverallMetrics* are extracted from the other metric categories, such as the *unary operators* and *field declarations* mentioned in the previous section. A total of 34 sub-metrics comprise the Overall Metrics score, these sub-metrics consist only of specific code properties and therefore exclude sub-metrics like *cyclomatic complexity* and *the number of code lines*. Only a boolean check of the sub-metric use is performed, therefore the number of usages above one has no impact on the score.

8.6 Appendix D - Class diagrams

The following diagrams are made using the *ObjectAid UML Diagram* plugin for Eclipse.

8.6.1 Main.java Class diagram



8.7 Appendix E - Iterations

This section contains the different stages the design went through.

8.7.1 Iteration 1

The first iteration started with making a simple mock up of a view showing some of the information retrieved in the backend, and a view of the badges. The different views were created in paint before they were conjoined in InvistionApp. The figure below shows the different views made for the first iteration.

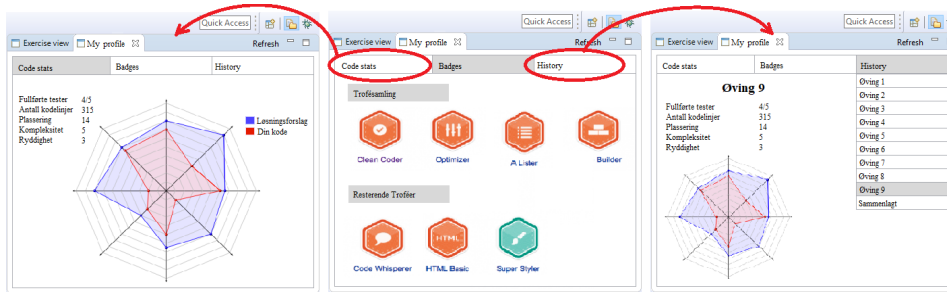


Figure 8.7: Wireframe iteration one

The view on the left shows the *Code stats* view, where the idea is that the view gives an overview of the assignment the student currently works on. The spider-diagram shown in the view was meant to illustrate the students scores in a straightforward manner. The middle view is the *Badge* view, that shows both achieved and remaining badges. On the right is the *History* view, that shows an overview of the assignments currently delivered as well as a summary which gives the average score of all the assignments.

A full wireframe was made instead of just a paper prototype. Because the paper prototype offers a lower design quality, lacks interactivity[45] and requires a higher imagination from the tester, which might limit the outcome of the testing[8]. The view was tested casually on some students in my class by letting them explore the wireframe, and asking them questions about their understanding of the solution.

8.7.2 Iteration 2

The feedback from the students testing iteration 1 was taken into account before making changes to iteration 2. The two main changes from iteration 1 was adding a detailed badge view and removing the *Code stats* view. The figure below shows the alterations made from iteration 1 to iteration 2.

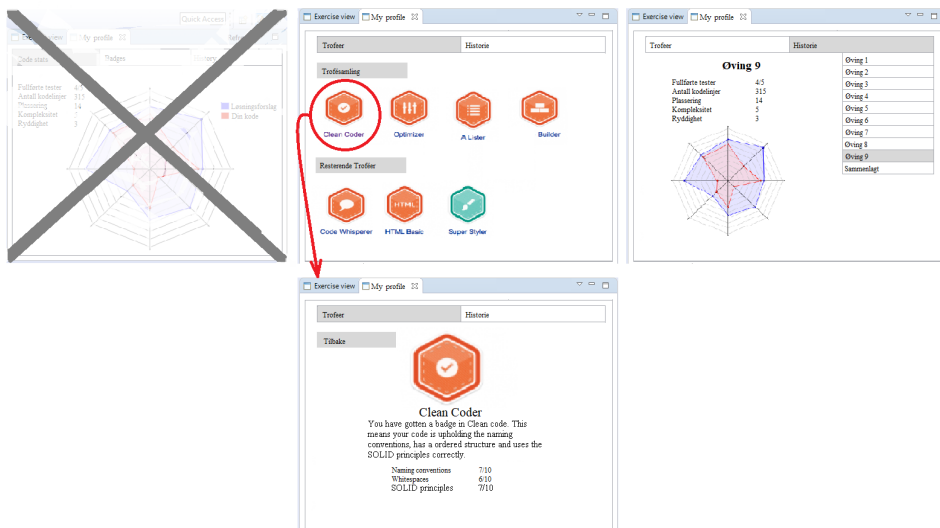


Figure 8.8: Wireframe iteration two

The reason for removing the *Code stats* view, was that the students that tested iteration 1 found it excessive and confusing, due to the same information being shown in the *History* view. The *Badge description* view, was added, this was done because the testers tried clicking on the badges, and were interested in knowing more about the badges. The same casual testing from iteration 1, was also conducted on iteration 2.

8.7.3 Iteration 3

After receiving student feedback on iteration 2, three changes was made to iteration 3. The figure below shows the main views of iteration 3.

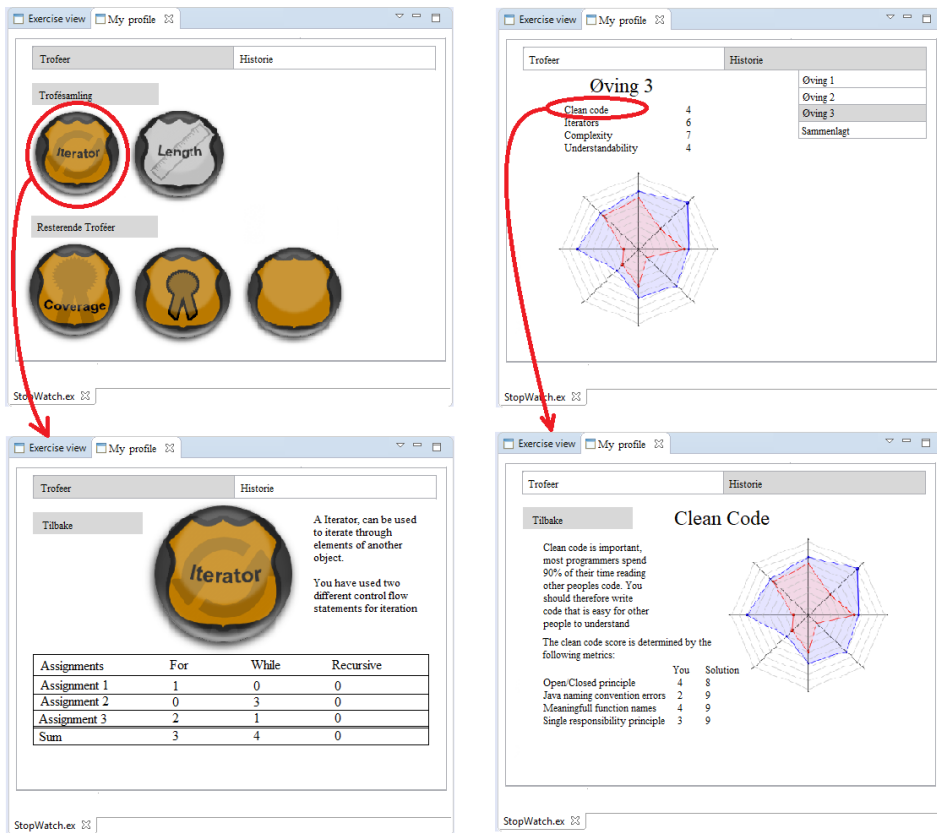


Figure 8.9: Wireframe iteration three

The first change that was made, was changing the badge design, this was due to experienced mismatch between badge name, and design. The second change was adding a more detailed explanation to the metric information listed in the *History* view, after being requested by multiple testers. Finally the *Badge description* view that was added in iteration 2, was extended with a table. Displaying specifically which sub-metrics were used in what assignment.

8.8 Appendix F - Results

This section shows a small selection of the results discussed in chapter 4. The remaining results can be found in excel documents located in the git repository

8.8.1 Proposed solution metrics extraction

The table below shows the full list of the metrics extracted from the proposed solution of assignment 5, 6, 8 and 9, sorted by difficulty. The numbers are the average number of uses

of each of the metrics.

Averages	Easy	Medium	Hard
elses	2,8	3,3	2,8
ifs	10,5	10,4	10
Throw statements	5,9	4,9	3,5
try	0	1	0
catch	0	1	0
while	0,5	2,3	0,5
array init	0,1	0,6	0
array creation	0,6	0,6	0
foreach	1	4,5	1
array accesses	2.8	0,2	0
method calls	36,4	55,7	36,8
code lines	175,9	228,1	165,5
abstract class + inheritance	6	6,8	5
super expressions	0,5	1,8	1,5

Figure 8.10: Proposed Solution metrics extraction

8.8.2 Badges with corresponding sub-metrics

The tables show each of the Badges, with the sub-metrics which are used to constitute the score.

General Metrics Badge
Lines total ratio Method calls total ratio Variable declaration ratio Weighted Method Per Class
Inheritance Badge
Class or Interface inheritance ratio Super expression ratio
Iteration Badge
Iterators used ratio Iterables used ratio
Exception Badge
Number of try ratio Number of catch ratio Number of throw ratio Number of IllegalArgumentExceptions
Conditional Badge
Number of Ifs Ratio Number of Operators used ratio Cyclomatic Complexity Ratio
Overall Metrics
Total number of metrics used

8.9 Appendix G - Frontend development tools

This Section will explain the technologies used to develop the web view.

8.9.1 Knockout

Knockout is a pure javascript library that enables coupling between front end objects and an underlying data model[3]. The image below shows how knockout keeps an automatic coupling between the viewmodel and the view.

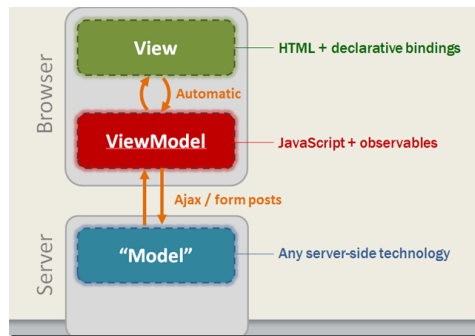


Figure 8.11: Knockout MVVM[4]

This helps keep a responsive design on the website, and is useful when the items on the site may change over time.

8.9.2 SweetAlert

SweetAlert is a replacement of the JavaScript alert. The figure below shows the difference in appearance between the JavaScript "alert()" on the left and a standard SweetAlert "alert()" on the right.

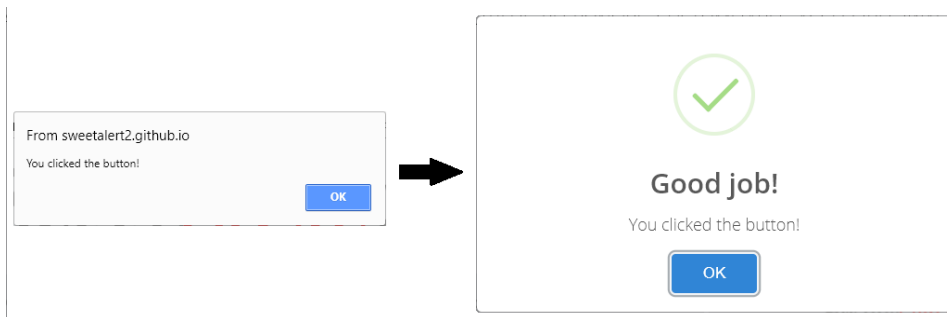


Figure 8.12: alert() vs SweetAlert

In addition to being visually better looking, the SweetAlert library comes with many other advantages. The alerts are responsive and automatically centers itself on both desktop and mobile. They also close when you press outside the message, compared to the javascript alert, where you have to press the "Ok" button. Another advantage is that they support customizable images and recursion, making it possible for an alert to open another alert directly. The figure below illustrates how this has been used in the solution.

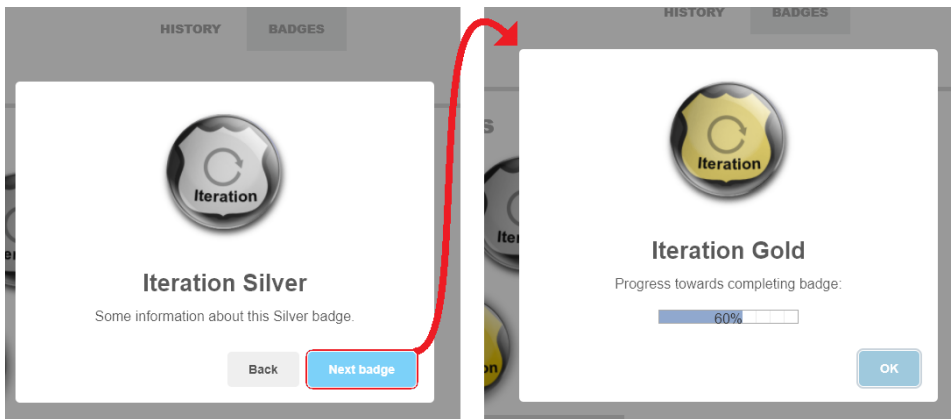


Figure 8.13: Next alert/notify

The alert on the left automatically opens the alert on the right when the button is pressed.

8.10 Appendix H - Suggested information for the alternative views

This section shows tables with suggestions of what information could be displayed in the alternative views for the teaching assistants and lecturer.

8.10.1 Information for the lecturer view

Part of curriculum:	Not started	Struggling	Mastered
Conditional Statements	40	130	470
Iteration	30	220	390
Inheritance	80	180	380
Exceptions	200	280	160
Code practices	20	80	540
Java Standard class methods	90	140	410

Figure 8.14: Lecturers view

Conditional Statements	Unused / Poor mastery	Partly mastered	Mastered
If	30	130	470
Else	45	110	390
Else If	90	90	380
Cyclomatic Complexity	200	300	160
==	20	80	540
!=	90	180	410
Boolean Function	40	400	170
>=	80	180	380
<=	10	280	160
&&	20	70	540
	90	140	410

Figure 8.15: Lecturers detailed view

8.10.2 Information for the teaching assistant view

Assignment 5	Iteration			General Metrics			Inheritance			Conditional Statements		
Anne	X					X		X			X	
Ola		X			X				X		X	
Finn			X			X			X		X	
Hanne		X				X			X	X		
Truls	X			X			X				X	
Dennis			X		X				X		X	
Håvard		X			X			X		X		
Ida			X	X					X	X		

Figure 8.16: Teaching assistant view

Anne Bråten - 193042	Assignment 1	Assignment 2	Assignment 3	Assignment 4	Assignment 5	Assignment 6	Sum / Avrg score
Conditional Statements ▾	3.2	1.1	2	1.9		1	1.84/2.2
If	2	4	3	11		3	23 / 32
Else	4	3	1	8		1	17 / 23
Else If	2	1	3	2		2	10 / 14
Cyclomatic complexity	4	3	3	4		2	16 / 19
==	3	2	2	3		1	11 / 9
!=	0	0	2	4		0	6 / 3
Boolean Function	1	0	1	2		0	4 / 2
>=	0	2	1	0		3	6 / 15
<=	0	0	0	2		0	2 / 2
&&	0	0	0	2		1	3 / 2
	0	1	1	1		0	3 / 1
Iteration	3.1	2.2		1.1	1.3	1.4	1.82/ 2.5
Inheritance			4	2	2.3	1.3	2.4 / 3
Java Standard Methods	2.7	3.2	1.6	2	2.5	1.1	2.2 / 3
Code practices	3.1	2	2	1	1.1	1.8	1.8 / 2.1
Overall	9/30	11/30	14/30	16/30	23/30	28/30	28 / 30

Figure 8.17: Teaching assistant detailed view