Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Marius Oscar Moe

System for gamification of lab work in TDT4100

Master's thesis in Informatics
Supervisor: Hallvard Trætteberg
June 2019

NTNU
Kunnskap for en bedre verden

Marius Oscar Moe

System for gamification of lab work in TDT4100

Master's thesis in Informatics
Supervisor: Hallvard Trætteberg
June 2019

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Summary

Learning new programming languages can be difficult and giving the right amount of supervision can be a challenge, especially in large courses like TDT4100 with up to 835 registered students for lab work. Additional sources of feedback are therefore welcome and gamification in educational contexts has shown promising results in the past, but has to be carefully implemented and does not work in every situation.

This study consist of creating a system for gamifying TDT4100 and testing the usability of that system. The created system focused on the technical challenges of creating an extensible and configurable system for easier testing of effective metrics and rewards. Usability testing was achieved with automatic monitoring of user interaction and a voluntary survey. The study utilized a larger sample size than earlier research on gamifying TDT4100, as the previous research focused on qualitative data on feedback of design and specific metrics.

The experiment testing the system had a large number of entrants, however, fewer than expected completed the usage objectives within the system. The results from the survey and the collected usage data revealed a low degree of usability and individual analysis of the questions from the SUS questionnaire showed a low perception of the system's usefulness.

The system achieved a high level of extensibility by enabling system administrators to choose and combine measures for any exercise in addition to choose criteria for individual achievements. New measures were also easy to add and hard-coded measures could be reused when combined with each other to create composite measures.

Sammendrag

Å lære nye programmerisngsspråk kan være vanskelig, og å gi riktig mengde tilbakemelding kan være utfordrende, spesielt i et stort kurs som TDT4100 med 835 registrerte studenter for øvingsopplegget i 2019. Flere kilder for tilbakemelding er derfor velkomen og spillifisering har vist gode resultater tidligere, men et slikt system er nødt til å bli implementert skikkelig og virker ikke i alle situasjoner.

Denne studien består av å skape et system for å spillifisere øvingsopplegget i TDT4100 og teste brukervennligheten til dette systemet. Systemet som er lagd fokuserte på å løse tekniske utfordringer knyttet til utvidbarhet og konfigurerbarhet for å kunne lettere teste effekten av forskjellige kodemetrikker og belønninger. Brukbarhetstestingen ble gjennomført ved hjelp av automatisk overvåkning av brukerenes interaksjon med systemt og ved hejelp av en spørreundersøkelse. Denne studien benyttet seg av en større prøvegruppe enn tidligere forskningin på området, da denne i stor grad har fokusert på undersøkelser av kvalitativ data fra tilbakemeldinger om design og spesifikke metrikker.

Eksperiementet med testen av systemet hadde mange deltagere, men færre enn forventet fullførte buksmålene til systemet. Resultatene fra undersøkelsen og den oppsamlede brukerdataen avdekket en lav grad av brukbarhet. Individuell analyse av spørsmålene fra SUS spørreskjemaet viste også at den opplevde nytten til systemet var lav.

Systemet oppnådde en høy grad av utvidbarhet ved å la systemadministratorer velge og kombinere metrikker for hvilken som helst øving i tillegg til å velge kriterier for individuelle merker. Nye metrikker var også lett å legge til og hardkodede metrikker kunne bli gjenbrukt og kombinert med hverandre for å lage sammensatte metrikker.

Preface

This master thesis is the work of Marius Oscar Moe as part of the course *IT3901 - Informatics Postgraduate Thesis: Software* at NTNU, Norwegian Univarsity of Science and Technology, Department of Computer and Information Science. The work has been supervised by Hallvard Trætteberg.

We would like to thank all the people making this report a reality and a special tanks to Hallvard for all the help and support.

Table of Contents

List of Tables

List of Figures

Abbreviations

Java AST	=	Java Abstract Syntax Tree
URI	=	Universal Resource Identifier
TDT4100	=	Introductory course at NTNU in object oriented programming
GDPR	=	The EU General Data Protection Regulation
TAM	=	Technology Acceptance Model
SUS	=	System Usability Scale
EMF	=	Eclipse Modeling Framework
DSL	=	Domain Specific Language
POJO	=	Plain Old Java Object
BPMN	=	Business Process Model and Notation

Chapter 1

Introduction

This chapter presents the motivation behind this thesis and earlier work on gamification of TDT4100 at NTNU. The problem description and objectives are also presented together with a note on the ethics behind the data gathered in relation to this master thesis.

1.1 Motivation

TDT4100 serves as an introductory course in object-oriented programming at NTNU [1]. The course utilizes both lectures and exercises to develop the student's skill where the approval of the exercises is done semi-automatically. This means that a student works with an exercise until a series of unit tests pass, these are tests that verify specific parts of the code behaves as intended. The exercise then has to be shown to a teaching assistant, hereinafter referred to as TA, for feedback and approval. The TA's responsibility is to verify that tests pass and give feedback on the exercise in general, the approval of the exercise is not reliant on the TA's remarks on quality. This is the only feedback students get on code qualities beyond confirmation of fulfillment of functional requirements and with more than 700 students registered for the course in 2018, giving all students adequate feedback can be difficult. Code quality is not a binary trait and any additional feedback could be valuable to the student.

The curriculum is also challenging as the failure rate on the ordinary exam for 2018 was 23 percent[1]. Other introductory courses also have high failure rates, but there is clearly room for improvement. The meta-analysis in [2, Chapter 4] shows that gamification is a promising technique for improving education. On the other hand, gamification in education has also been shown to be challenging to implement in such a way that it yields measurable improvements [3, 4].

[1]Spring 2018, course TDT4100 Objektorientert programmering - `https://sats.itea.ntnu.no/karstat/makeReport.do`

As described below in section 1.2, a similar application has been explored earlier as a prototype, but with some important limitations. There is currently no running system that compliments the exercise program in TDT4100 with code insights.

1.2 Earlier work on gamification of TDT4100

David Åse worked on a concept for gamifying TDT4100 as a pre-project for his master thesis in 2014. Christian Rasmussen joined the actual master project which aimed at creating a web-based code editor with some of the game elements from the pre-project incorporated [5]. The system they created also let students see each other's progress and rewarded students for passing Unit tests. The thesis focuses heavily on the technical implementation of a web-based code editor and will have limited influence on this thesis.

In 2015 Espen Selquist Stenmark wrote about how gamification can stimulate development of practical skills in the Eclipse IDE [6]. The developed prototype and the successive user test were based on the teaching goals of TDT4100. The chapter about gamification is especially relevant as this thesis will utilize some of the same methods for gamification.

In 2018 Syver Bolstad wrote his thesis on how to gamify TDT4100 and created a prototype of an application that gamifies the exercise program [7]. This master thesis focused on how to support the course through gamification by utilizing software metrics to achieve badges. It is this concept that lays the design foundation for the system developed with this thesis. It is important to emphasize that Bolstad's prototype had a limited number of; test participants, metrics, and badges implemented.

1.3 Problem description

The problem description was articulated as the following:

Today's exercise program is based on JUnit tests for testing functional requirements, other qualities are important as well that do not become stimulated and are important to improve upon. The project will use elements from games, like badges and rewards to encourage better learning of other qualities than today's lab work. There has been done some work on the design earlier, therefore this project consists of realizing a system that can be tested on a greater scale and be built upon.

The ”other qualities” that are mentioned in the problem description refer to non-functional requirements that can be indicated by code measures, the foundation for specific qualities and metrics considered are described in section 2.2. The different game elements and mechanics considered are described in section 2.3.

The importance of a system that can be built upon was emphasized further during the development and added as a separate objective for this project. The extensibility of the system was therefore subject to further discussion in section 5.3.

1.4 Objectives

The problem description was turned into the following objectives for the software to be made with this thesis:

- **Objective 1:** Create a web-based platform that gamify the lab work of TDT4100 by using software metrics to show progress and skill level
- **Objective 2:** Create a system with good usability, especially in terms of perceived ease of use, that is the user expect to use little effort in using the system
- **Objective 3:** Enable extensibility of which metrics to: use, display for an exercise, and use for achievements

The discussion on how the objectives have been fulfilled can be found in chapter 6. To achieve the objectives an agile development approach was used. The process method utilized methods from Extreme Programming (XP) as described in [8]. The specific tools used were Kanban boards, user stories, and development cycles with defined goals for implemented features. A more detailed explanation of how agile development methods were implemented can be found in section 3.1.

1.5 Thesis outline

Chapter one presents the motivation, earlier work on gamification of TDT4100, the problem description with subsequent objectives, and ethics in relation to the gathered data. Chapter two presents the background theory, which includes; course details, techniques for extensibility, gamification, and metrics. Chapter three explains the relevant research methods and how the experiments conducted were carried out. Chapter four presents the various requirements. Chapter five explains the different design decisions and solutions implemented as well as the overall system architecture. To describe the architecture the 4 + 1 view model from [9] was used. Chapter six covers the experiment with results, discussion of the findings and threats to validity. Chapter seven provides an overall conclusion together with suggestions for further research and development.

1.6 Ethics

The data gathered and used in this research consist of code snapshots and metadata for exercises in TDT4100. More specifically, a snapshot of the code is taken for every saved instance of the code as well as every attempt at running unit tests, the method that ensures that data only related to coursework get stored is done by monitoring packets matching a regex. Metadata like; time window was activated, test results, and stack traces are also recorded. The recorded data file contains no references to a natural person and the implementation of how the recorded data is gathered can be found in the Jexercise repository [10].

The main objective with processing the data, in the context of this thesis, is to convey to the student indications on quality attributes of the code by using code measures. The second objective is to enable course facilitators with aggregated data to better understand what concepts are understood and which need further explanation in lectures.

The GDPR constitutes important rules for how data-driven software shall be governed. The term ’personal data’ is defined in Art. 4(1)(1) in [11] and include information related to work times, which is recorded by Jexercise, when these can be theoretically connected to a natural person. The recorded data has to be uploaded by a logged in user because only that user should be able to see metrics calculated for their exercise. This means that files with recorded data are connected to individual accounts and the data in question must follow the GDPR rules. Several steps were taken to comply with GDPR. Firstly, all participants were informed of the following: what data was gathered, for which purpose, and that data would not be sold or given to any other parties. Secondly, all participants had the option to ask for all their data to be deleted or provided for them to see. The system created also logged behavior and created individual profiles for each user. This was also subject to the same steps for compliance with GDPR and was necessary for acquiring usage metrics to evaluate the system. Other methods were also considered like a longer questionnaire, but the fear of inaccurate results and a plethora of other interesting metrics tipped the scale in favor of automated user behavior collection.

Chapter 2

Background and theory

This chapter presents the background and theory, which contains an introduction to the lab work, software metrics, gamification in an educational context, and extensible systems. The topics were chosen based on what domain knowledge is needed for gamification of TDT4100 and how to achieve the goals presented in section 1.4.

2.1 Lab work

Understanding the various steps in the lab work of TDT4100 was important for gamifying the course as it was the subject of what was gamified. To pass TDT4100 the student must pass the lab work and the final exam, but only the score achieved on the exam count toward the final grade [1]. The lab work consists of a series of programming exercises managed by the course responsible, in order to pass a certain number of exercises must be approved. Each exercise is split into several programming tasks, in which the student has to choose and solve some of them, the specific number of tasks required for an exercise is stated in the respective exercise. Some exercises also have mandatory tasks and tasks with optional sub-tasks [12]. An example could be the state and behavior exercise from 2018, the student has to complete 4 tasks where 3 of them can be chosen from a list of 7 with varying difficulty. Every task is accompanied by a series of Junit tests. To complete a task every Junit test for that particular task has to be completed. The end result has to be shown to a teaching assistant for approval. The teaching assistant may also help the student if required and gives feedback on the implemented code.

The various exercises in TDT4100 focus on a series of important topics within object-oriented languages and programming in general. Some of the topics from the exercises are; objects and classes, encapsulation and validation, object structures, interfaces, handling files, observer patterns, and inheritance. Each exercise typically targets a specific topic but later exercises sometimes cover several topics. The course is not limited to programming alone; there are learning objectives for tool usage and testing as well.

The course has an initiative for recording the student's path to the exercise solution. This is done through an extension to the Eclipse IDE called Jexercise. The extension gathers data about; when and how the code is executed, stack traces, Junit test results, and changes to the source code in between code executions. All data is stored in a .ex file which is a serialized instance of the exercise model from the Jexercise repository. The .ex file is to be delivered in Blackboard, the course e-learning platform, as proof of completing the exercise.

2.2 Software measures

Software measures were included in chapter 2 as they were chosen to be the benchmark at which rewards could be given in the gamification process of TDT4100.

Software measures can use either internal or external attributes to derive their value. Internal attributes are "attributes that can be measured purely in terms of product, process, or resource itself [13, p. 88]". External attributes on the other hand are attributes that make use of observations over time or how the system works with its environment, for example, failure rate or defect rate.

Since the exercises measured never will be set in production or maintained for an extended period of time, only metrics that utilize internal attributes were considered. The measures were: lines of code, cyclomatic complexity, cohesion, coupling, tree impurity, and Java structures. The measures are explained below together with the reason for why they were interesting for this project.

Several of the proposed measures utilize *modules* as an important concept. This term has been defined multiple times in literature; [14] propose that it can be described as a Java interface, [15] describes it as "Modules are assigned specific computational responsibilities, and are the basis of work assignments for programming teams [15]", and the spring documentation use it for separating different functionality [16]. For the purpose of this thesis, the definition from [15] will be considered a module.

While the goal with static software measures is clear, indicating software qualities, the actuality is that it is rather hard to quantify. Unlike laws in physics, software qualities do not originate from measures and observations, but rather a categorization of what is generally conceived as a quality. One proposal for what software quality is as a whole can be found in ISO 25010:2011 [17]. This document standardizes what is important to consider when reviewing a system's quality in terms of qualities that do not originate from measures. Therefore, it is difficult to create measures that in right quantify these properties.

2.2.1 Lines of code

Lines of code and kilo lines of code often abbreviated to LOC and KLOC respectively are often considered to measure the size of a project [13]. A challenge with LOC is its easily inflatable nature. While verbosity can be a good attribute, making little progress with large amounts of code might indicate that the program could have been written more efficiently. This problem has led to the proposal of numerous other measures that try to describe the project size more transparently. An example is NCLOC, noncommented lines and CLOC, comment lines of program text. These amount to LOC when combined, see formula from [13]

$$LOC = NCLOC + CLOC \tag{2.1}$$

This relation gives other useful derived measures, like density of comments

$$\text{density of comments} = \frac{CLOC}{LOC} \tag{2.2}$$

The separation of program code and comments only partly solves the problem with inflatable numbers. Further fragmentation could be done by creating a scoring system where code lines that do more get a higher score, measuring the amount of functionality instead of purely project size, real-life examples are the COCOMO II model [13, p. 358-359] and Function points from Albrecht [13, p. 352]. The downside with this approach is that it introduces a whole new range of challenges related to the weighting of meaningfulness.

2.2.2 Cyclomatic complexity

While the name 'cyclomatic complexity' contains the word 'complexity', this number does not directly indicate a program's complexity [13, p. 92] it rather measures the number of linearly independent paths through a program. The cyclomatic number can be calculated with the formula in [18]

$$v(G) = e - n + p \tag{2.3}$$

where the flowgraph G has e edges, n is the number of nodes, and p is the number of connected components. Alternatively, one can count the forking nodes of the Java AST generated from the source code.

In the IEEE Standard Dictionary of Measures of the Software Aspects of Dependability from 2005, cyclomatic complexity was removed due to the following reason, "There is no scientific evidence that threshold values that must be used with this measure, such as 10, have general applicability [19]". This measure could still, however, prove useful when only considering the measurement on an ordinal scale against a solution manual as this would give the student a reference point in regard to the number of necessary paths through the program.

2.2.3 Coupling

Stevens, W. P. et al. paper on how structured design can help the development of software from 1974 explain coupling as, "the degree to which each connection couples (associates) two modules, making them interdependent rather than independent [20]", where connections are described as a reference to some external entity. The assumption is that fewer connections between modules lead to code that is easier to read and reduce the chance of errors affecting unforeseen parts of the system upon changes in the system.

One proposal for calculating coupling is *coupling between object classes* (CBO) and is defined for a class as how many other classes it is coupled with [14]. In practice, this translates to summing up the number of external classes a single class references.

2.2.4 Cohesion

Cohesion refers to the degree components inside a module belong together, a high cohesion is to maximize relations among elements in the same module. Several ways to calculate this number have been proposed, one of them being *lack of cohesion in methods* often abbreviated to LCOM, which can be calculated by the formula provided in [14]

$$\begin{aligned} LCOM &= |P| - |Q|,\ if|P| > |Q| \\ &= 0 \text{ otherwise} \end{aligned}$$

where P is $(I_i, I_j)|I_i \cap I_j = \emptyset$, Q is $(I_i, I_j)|I_i \cap I_j \neq \emptyset$, and I_i is the set of instance variables used by method M_i.

2.2.5 Tree impurity

Tree impurity aims to measure the deviation from a tree-like structure in a program's information flow between modules. One proposal for calculating tree impurity $m(G)$ is provided in [14]

$$m(G) = \frac{2(e - n + 1)}{(n - 1)(n - 2)} \tag{2.4}$$

Where e is the number of edges and n the number of nodes in the graph constructed from the information flow between the modules. This measurement is meant to indicate design quality of the software system, where the more a system deviates from a tree-like structure as opposed to a graph like one, the worse it is.

2.2.6 Java constructs

Java constructs are the components a Java program consists of, and not all are commonly used in the literature as measures. Java can be a difficult language to learn and to keep track of which concepts and Java constructs one has used can, therefore, be valuable. One way of deriving these constructs is to look at the abstract syntax tree of the Java code, which can be achieved with Eclipse's syntax parser. This will generate a tree with AST nodes, "An AST node represents a Java source code construct, such as a name, type, expression, statement, or declaration [21]". When these measures are accumulated over several exercises one can tell whether some concept or technique were encountered in the past. This can be used to help the decision process of what one should work with to learn the whole curriculum.

2.3 Gamification strategies

Gamification strategies were included as they address the various ways gamification of a subject can take. Sebastian Deterding et al. propose the following definition of 'gamification', ”the use of game design elements in non-game contexts [22]”. This entails that several components make up a gamified system, but it provides little information about what these elements might be. Another definition made by Karl M Kapp for gamification in an educational context is ”Gamification is using game-based mechanics, aesthetics and game thinking to engage people, motivate action, promote learning, and solve problems. [2]”. This definition highlights an important point that gamification is not only about adding badges, points, or rewards, but encouraging the user to participate and solve problems. In the subsections below a presentation of relevant game mechanics and game thinking will be given.

2.3.1 Goals

Goals are an integral part of every game and are important as they ”adds purpose, focus, and measurable outcomes [2]”. An example of such a goal in an educational context could be to beat your friends in the number of correct answers at a test. The goals can also encourage the player to explore a larger area of the game or achieve a higher level of completeness in the game.

2.3.2 Rules

Rules define what means are allowed to use to reach the goals of the game.

2.3.3 Reward structures

Reward structures are different ways to communicate progress or excess. Some common reward structures for digital systems are:

- Points - can be given based on a number of measurable attributes, but the general idea is that the more you have the better.
- Achievements and Badges - are graphical icons that often carry some meaning, like mastering certain fields or demonstrates a certain proficient in a skill.
- Cosmetic items - this kind of reward requires some sort of avatar system as the reward comes through improving or differentiating this character from other players.
- High score lists - High score lists can be seen as an extension of reward structures as being at the top can give ”bragging rights and social capital to the individuals who achieved the high scores [2, Chapter 2]”.

2.3.4 Motivation

To successfully gamify some domain or task one is also reliant on that the player want to achieve the rewards available. This motivation can come from extrinsically motivating factors, like getting compliments from other students or intrinsically, the student itself.

2.4 Success of gamification in education

A quick overview of the success of gamification in education was included to explore lessons learned from earlier experiements with gamification in education.

In [2, Chapter 4] a meta-analysis study of game-based learning was conducted utilizing six literature reviews that in total examined 341 studies from 1963 to 2007, several studies on specific game elements were also conducted resulting in a series of key takeaways. The findings indicated among others that the benefit of gamification is most apparent when the content is not too broad and learning objectives are clearly defined. It also claims that extrinsic motivation is likely to influence the intrinsic motivation negatively if the reward does not tell the student anything meaningful about their skill or knowledge on any related subject. Common to several of the key takeaways are that they reiterate that gamification does not work in all use cases and the successfulness depends upon implementation and learning context.

The field experiment in [23] gamified the process of committing code during four weeks in a computer programming course, where students competed in having the highest score in code quality. Scores were computed by analyzing the correctness of Javadoc and the experiment found no improvement in quality. The study points to a series of important aspects that can guide future implementations; think about the evaluation of how successful the experiment was, metrics that are understandable and possible to change through hard work for participants, and scores that do not feel unfair or is counterproductive for team performance.

In a recent study, [3] found that gamification of a math-based game yielded minimal improvement with the introduction of points for the 1911 participants. The study contained two experiments; both found no performance gain in terms of accuracy but minor improvements in speed. Bolstad also concluded that whether his prototype provides a benefit for learning could not be determined [7], further confirming that the results of gamification are not always positive.

2.5 Extensible systems and tactics

There are many software measures and metrics that can be useful for indicating software quality, however, there will only be time to implement a handful in this project. Configurability was also seen as important to achieve extensibility since it enables a faster implementation of expanding features. The extensibility that is thought of is the ability to choose measures displayed for any specific exercise, add measures at a later stage in development, and combine implemented measures to new measures. The extensibility of the system was considered a highly preferable trait and different methods for achieving it are explored in the subsections below.

Modeling and the use of domain specific languages (DSL) can be used as a tactic for extensibility. Since extensibility, a system's ability to expand, is a form of change, many of the architectural tactics are the same as for modifiability. The tactics presented in [15] for modifiability are therefore included below in section 2.5.2.

2.5.1 Modelling and domain specific languages

Marco Brambilla et al. argues that one always make some kind of model of the problem domain regardless of the modeling effort, so whether it is needed or not is not an appropriate question [24], despite this the rigorousness, explicitness and how the model is structured matters and deserve attention as creation of models can be both expensive and their usefulness vary. Then why use modeling to achieve an extensible system? Models can be used for both descriptive and prescriptive purposes, e.g. describing the system and figuring out the scope of the problem domain. Models can also contain extension points enabling system designers to choose how and where unknown or new components can enter the system. An extension point enable custom implementations where needed at places where they can interoperate with the rest of the model [25, p. 178]. Object-oriented models can incorporate this feature through inheritance and interfaces. A framework for facilitating the creation of such models is the Eclipse Modeling Framework (EMF) that will be explained further below. One ”version” of modeling where rules for extension points are clearly defined is through domain specific languages.

Configuration with domain specific languages

A domain specific language (DSL) is defined by Brambilla et al. as ”languages that are designed on purpose for a specific domain, context, or company, to support people who need to describe things in that domain [26]”. They can also help alleviate the gap between the problem and the platform as the entities in the DSL more accurately capture the specifications of the problem, rendering general purpose languages more verbose and error prone in some situations.

DSLs can according to [25] take on two opposite directions, towards routine configuration and creative construction. Routine configuration is configuration through static fields or a variation thereof. Creative construction, on the other hand, provides more freedom and enable the designer to create a more graph like language, capable of defining new and connected modeling elements. In between these two, there are DSLs that can provide some guidance to the developer in terms of rules and order of modeling objects. By declaring rules or defaults on the model instance to be configured, simple operations can be standardized and easily repeated.

In some situations where the configuration instance changes rapidly or by a non-technical person a separate syntax for the creation of the configuration instance could be beneficial. One benefit with a separate syntax is that the configuration can be validated by an IDE before it is used , decreasing the likelihood of system instability. Another benefit is the guidance a good editor in combination with the syntax can give. This includes; auto-completion, syntax highlighting, automatic imports when referencing external attributes, and the possibility of adding a graph based editor. Both Eclipse and Jetbrains have their own idea of how this can be achieved. Eclipse has Xtext which builds upon the Eclipse

Modeling Framework (EMF). It enables the creation of textual languages through a grammar language [27]. Jetbrains, on the other hand, proposes the Meta Programming System (MPS). The tool was officially launched in 2009 [28] and is aimed at creating domain specific languages [29]. Class structures can also be modeled with this tool, but that is not its primary objective.

Eclipse modeling framework

EMF is a modeling framework and code generation facility for building various Java applications from models [30]. It combines XML, Java, and UML to facilitate development and XMI, a standard for serialization of metadata with XML, for persistence.

Ecore is used to express models in EMF and is an EMF model, making it a meta-metamodel. Ecore can also provide runtime support for models and provide a well-defined API for manipulation of EMF objects at runtime [31]. This makes Ecore suitable for modeling various problem domains and together with other tools in the EMF enable model to text transformations.

A well-defined Ecore model instance has a highly structured hierarchy, where eObjects have links to other eObjects that it contains and to the eObject where it is contained, if it's not the root node. The Ecore component eReference holds information about the link's direction of relation, a subset of the Ecore model showing how links are stored can be seen in figure 2.1. The structure makes it possible to traverse an instance of the model generi-

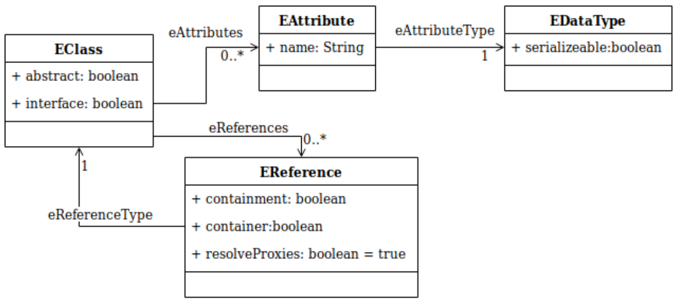


Figure 2.1: Simplified model of Ecore components relations, attributes, and operations

cally, one can then access and edit attributes without foreknowledge of the exact structure of the model instance. In order to serialize a model instance, the instance must be part of a resource which again has to be a part of a ResourceSet. Every resource within the ResourceSet is associated with a URI as can be seen in figure 2.2. If the eObject links to an object in another resource, the reference will be demand loaded at runtime if the *resolveProxies* from figure 2.1 is set to true and the two resources exist in the same ResourceSet.

The concept of demand loading additional resources can be used in conjunction with object traversal to achieve configurability, an example can be seen in section 5.3.1. A more general example of a use case for EMF is to model a problem domain in Ecore, then gen-

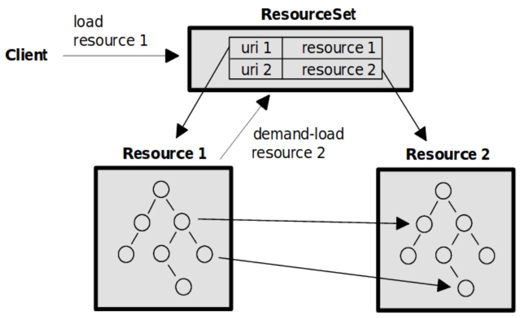


Figure 2.2: Resources as part of a ResourceSet, Ed Merksand and Dave Steinberg - copyright IBM made aviliable under the EPL v1.0, from Eclipsecon 2005

erate the Java code and write the unimplemented methods by hand. The system can then persist its state by storing it in an XMI file.

2.5.2 Other tactics

Reduction of module size

Split modules – By splitting modules with a large responsibility the workload is decreased when changing this module in the future.

Increase cohesion

Increase semantic coherence – this tactic is similar to ’splitting modules’, but with the sole purpose of increasing the cohesion within the module.

Reduce coupling

Encapsulate – encapsulation is a technique where the interactions that can be made on a module is limited to an explicit interface. This can limit the impact of one change affecting other modules.

Use intermediary – By breaking up dependencies with an intermediary, previously depending modules no longer need to know of each other’s implementation.

Restrict dependencies – Limit the number of dependencies, hence the project size.

Refactor – Refactoring or cleaning up code is useful to remove duplicated and poorly written code.

Abstract common services – This tactic suggest moving common code structure to modules for themselves and rather let other modules use these modules to avoid duplicated code.

Defer binding

Defer binding is to let functions or methods be used for a wider set of use cases by enabling the functions to receive more parameters determined at run time.

Chapter 3

Development approach

This chapter presents the chosen approach for answering the objectives in section 1.4 and contain both the development methods and the research process used to evaluate the system.

3.1 Development method

The first and third objective in section 1.4 revolves around creating a functional system that gamifies the lab work of TDT4100. The method used to achieve this was an agile approach based on various tools in extreme programming (XP) and Kanban from Henrik Kniberg [8]. The Kanban board was used in much of the same way as in Scrum where the original objectives for the system are added as user stories to the project backlog and in every sprint, four weeks in this project, a subset is chosen to be worked upon. Every sprint new user stories get selected for the successive sprint, and new user stories that arrive during development are added to the backlog and possibly included in the next sprint. After every two sprints comes a release, which is a working piece of software. User stories were handled by the Kanban board built into GitHub where also the code was committed. The public repository can be found at `https://github.com/mariusmoe/metrics-consumer`.

Scrum as a whole was not used due to its heavy focus on team-based activities like; stand up meetings, daily planning, and sprint demos. Instead, some of the activities were changed to work for a one-man team. Some examples are daily planning in calendar application and retrospective as writing sessions.

3.2 Research question

To be able to discuss the second objective in section 1.4 a research question was articulated as follows:

RQ: Determine how the users' interest and system usability impacts the actual use of the system

It is important to emphasize that the objective was not to defend the inclusion of the built-in metrics.

3.3 Research strategy

To achieve all the objectives in 1.4 and answer the research question the chosen research strategy was set to design and creation. It focused on creating an extensible framework for gamifying the lab work in TDT4100 and investigate the research question. The developed system explored the technical challenges in building an extensible and configurable system that was able to display selected measures, create compound measures, create achievements, and add new measures. A more extensive list of what the system was able to do can be found below in chapter 4. For the investigative part, the developed system was available for four weeks followed by a survey. The timing for the test period was important because it spanned over the delivery window of three exercises and therefore gave students multiple opportunities to test the system. The research strategy fit because it made it possible to create and evaluate the system's usability.

3.3.1 Data generation method

The data used in the evaluation was collected in three ways. The first was with a survey consisting of a questionnaire including the questions from the system usability scale (SUS), described in further detail in section 3.3.3. The second and third were usage patterns recorded by Google Analytics and data aggregated from the system's database. The two last methods for gathering data was implemented in an unintrusive manner to minimize the effect it would have on users. The data generation method aligned well with the research question from section 3.2 as the usage data measured the actual use of the system and the questionnaire aimed at indicating the system's usability.

3.3.2 Participants

The experiment was open for participation for the 835 students taking the course TDT4100 at NTNU in 2019[1]. The participation was voluntary and the experiment was advertised in lectures and linked to from the course e-learning platform. Incentives in the form of food coupons were introduced in the last week of the experiment to encourage users to also answer the survey. TDT4100 serves as an introductory course to object-oriented programming and is typically the second programming course students take. It is therefore expected that many of the participants will be at a low level in regard to programming skills.

[1]Enlisted students for the exercise program in TDT4100, retrieved 1. April 2019 from Black Board (e-learning platform)

3.3.3 Data analysis

The data gathered from the survey was used to compare users' expectations derived from SUS scores with their completed objectives during the test period.

Google analytics was in part used for recording user behavior which collects additional useful information about users behavior. These are statistics on bounce rate, session duration, number of returning users, and many more. The discussion and meaning of the Google analytics statistics can be found in section 6.

Methods for analyzing system acceptance

Several models and techniques have been proposed in the literature for trying to explain why information systems succeed or fail. One of these is the technology acceptance model, hereinafter referred to as TAM. The goal of TAM is to give an explanation of what makes a system accepted by looking into how perception of the system and attitude are influenced by the system's objective characteristics. The model was first presented in [32] and puts forward two determining beliefs for what constitutes the attitude toward using the system, perceived usefulness and perceived ease of use. Perceived usefulness is defined as the probability a user has for the expected increase in his or her work performance. Perceived ease of use, on the other hand, is defined as the users' expected effort in using the system. The model can be seen in figure 3.1.

Measuring perceived usefulness and perceived ease of use has been in frequent use since its introduction and has seen several proposed iterations and variations [33]. The method has also been found fruitful for understanding system usability several times since its introduction in 1986 [34, 35, 36].

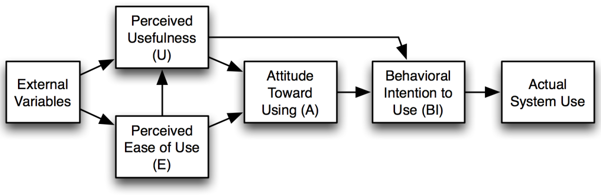

Perceived Usefulness (U)
External Variables
Perceived Ease of Use (E)
Attitude Toward Using (A)
Behavioral Intention to Use (BI)
Actual System Use

Figure 3.1: Technology acceptance model, Nippie CC BY 3.0 [2]

The system usability scale, hereinafter referred to as SUS, was introduced in 1986 and aimed to indicate a users' subjective evaluation of the system usability by using a likert scale with ten questions. The scale outputs one final output calculated by multiplying 2.5 with the points for each question. The points for each question is one subtracted from the score for the odd number questions and 5 minus scale position for even number questions

[2]`https://creativecommons.org/licenses/by/3.0`, from Wikimedia Commons

[37, p. 189-194]. The questions included in SUS can be found in appendix A.

While the output range from 0 to 100 a median score does not mean an average acceptability score, an average acceptability score has been found to be approximately 68 according to [38]. In a recent study on an e-learning system, SUS was utilized in part to infer the usability degree. It was found that SUS on its own is not enough to evaluate usability and suggested that future studies incorporate usage data and interactions recorded by the system as basis for their evaluations [36].

Chapter 4

Requirements

This chapter introduces the system in addition to the functional and non-functional requirements. The requirements were elicited from meetings and the initial objectives described in 1.4.

4.1 System description

To give the reader an idea of how the system looks like, an outline of how it can be used is given in this section. A figure depicting the activities can be found in figure 4.1.

A student can log in to the system and upload a .ex file. When completed the student can navigate to another page where he/she can see computed code measures from the source code contained in the .ex file together with measures from the corresponding solution manual. Based on the measures displayed and the descriptions of the measures, the student can get an impression of what qualities the delivered exercise has. The student can also navigate to the achievements page where progress towards various badges can be seen. These express proficiency in certain areas accumulated across all exercises or on particular exercises. The measures displayed for each exercise and the achievements can

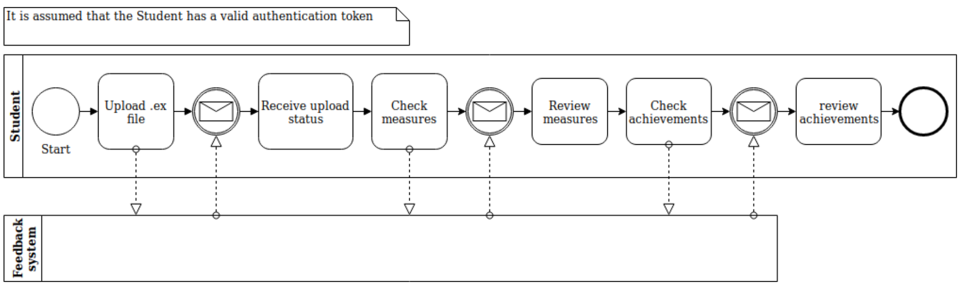


Figure 4.1: Simplified usage example with BPMN

be configured by the course responsible.

4.2 Functional requirements

The functional requirements served as an explicit list of physical milestones that had to be achieved. A more detailed version of this list can be found in the GitHub issue tracker, see reference to repository in section 3.1.

4.2.1 As a student

- FR1 – A student should be able to see measures generated from their progress on exercises compared to measures generated from a solution manual
- FR2 – A student should be able to see achievements, both accumulated and specific to an exercise
- FR3 – A student should be able to submit measures and .ex files as basis for metric calculations
- FR4 – A student should be able to see a set of achievements that can be cumulative or specific to an exercise

4.2.2 As a course responsible

- FR6 – A course responsible should be able to configure for each exercise a set of composite measures or individual measures that shall be calculated and shown to the student
- FR7 – A course responsible should be able to add cumulative achievements and achievements for specific exercises

4.3 Non-functional requirements

The non-functional requirements guided the creation of the detailed issues especially in regard to extensibility.

Security

- Measures submitted by a student should only be accessible for that student
- Login credentials should not be attainable for bad actors
- Login should be handled with Feide authentication provider

Privacy

- Users should be properly informed of the identifying information being stored by the system

- Users should be in control of the data collected about them

Extensibility

- Metrics should be possible to add by including a new IMetricProvider and declaring it in the system's configuration file
- Metrics should be possible to add by deriving new ones from existing metrics through a configuration
- Additional achievements should be easy to add to the system, so that adding achievements would take less than 5 minutes to add.
- Configuring which metrics should be displayed for a particular exercise should be configurable in less than 5 minutes.

4.4 Changed requirements

The focus of FR3 changed to emphasize the upload of .ex files as it was discovered that there would not be time to implement the necessary business logic for automating uploads from the Jexercise extension.

4.5 Outside scope

Some requirements that were envisioned at an early stage like using Graphql and adding community response per metric were not implemented due to their priority and time limitations in the development schedule. An extensive list of not implemented functional requirements can be found among the project GitHub issues with the label ”wontfix” at the GitHub repository, see section 3.1.

Chapter 5

Implementation

This chapter goes through technical details in the application and the implemented; gamification strategies, measures, and architecture.

5.1 Gamification

The system incorporated two main game elements, badges and a variation of high score lists. Badges, or achievements, were implemented in a configurable manner and could be added for a single exercise or as a cumulative counter toward a goal. The requirement to achieve a badge could be set with the method described in section 5.3.1. The variation of high score lists used was a view enabling the comparison of the student's metrics to the solution manual. Since every exercise could consist of a variety of tasks with or without additional sub-tasks the metrics for the corresponding solution manual had to be generated for each submission. Screenshots from the system can be seen in appendix D.

5.1.1 Included measures

The uncertainty around which metrics to include was a core issue in the project and spawned much of the need for an extensible system. Some metrics were chosen to be included with the program from the start. The included measures were as follows:

- Cyclomatic complexity – while it was disbanded by the ISO, standard cyclomatic complexity can still indicate if there has been used excessive branching structures to achieve the required functionality
- Lines of code – often used as a measure for size which is useful in conjunction with other metrics
- Method declarations – The number of method declaration can indicate if the student has separated responsibilities enough

- Java constructs – can be used to monitor which parts of the java language has been used in the various exercises. A description of java constructs can be found in section 2.2.6.

5.2 Models from Jexercise

EMF has been used for modeling due to existing models being created with EMF and its useful features within model transformation. While MPS could have been used, this language has a shorter track record and has a more focused set of features making it more likely as an alternative to Xtext, which is built on top of EMF.

The system utilizes two existing Ecore models from the Jexercise repository, these are a feature vector model and an exercise model. The IMetricProvider interface has also been used extensively; while not an EMF model, it is maintained as part of the Jexercise repository.

5.2.1 Feature vector model

A feature vector is simply a list of attributes associated with a numeric value for an object. Throughout the field of informatics feature vectors are commonly used in image processing and artificial intelligence but are not limited to these fields.

The feature vector model from the Jexercise repository, hereinafter referred to as the 'fv-model', is a metamodel for data structures, with the added capabilities of declaring various computational operations on the data it contains. The model can be seen as a DSL that has both routine configuration and creative construction properties, as it enables manipulation of a dynamic tree-like data structure. The fv-model is used for storing and eliciting the metrics that shall be displayed for exercises. One such method is through the ExpressionFeatures class, an example of how lines can be chosen to be displayed can be seen in figure 5.1. The ExpressionFeatures class work by evaluating expressions in the FeatureList contained by the ExpressionFeatures object. In the example this expression is 'lines', a reference to one of the features of the FeatureList pointed to by the other reference, however, this expression can reference multiple features and perform simple arithmetic operations.

The fv-model is also used to determine the threshold for receiving a badge, by using the built in FilteredFeatures class, a simple example of how filters can be used to only show measures with a value above 10 can be seen in figure 5.2.

The feature vector model used in this project has been extended from the original with functionality for; grouping, attaching identifiers, and attaching tags to objects implementing the FeatureValued interface. Identifiers and tags have been added by creating an additional abstract class that delegates all other methods to their respective class. The grouping feature is simply a new class with a list of FeatureValued objects. The fv-model enables many useful features that proved useful, this includes; filtering, arithmetic operations, and grouping measures to their IMetricProvider.

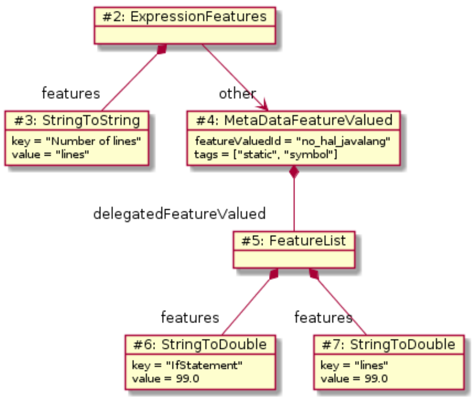


Figure 5.1: Combining measures fv-model instance example

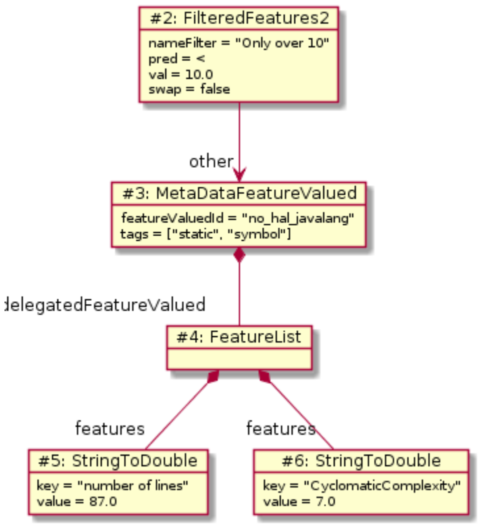


Figure 5.2: Filtered fv-model instance example

5.2.2 Exercise model

The exercise model defines how the exercise data that the student produces is structured. This includes metadata like; exercise name, timestamps for when tests ran, test results, and a snapshot of the code when tests ran. The exercise model also includes a dependency to the fv-model.

This model is needed to parse and extract the code from the student submissions for use in IMetricProviders.

5.2.3 IMetricProvider

The IMetricProvider interface gives the promise of a single method that calculates a metric or metrics as a FeatureValued which is an instance from the fv-model given a string of source code or a Java abstract syntax tree (AST).

5.3 Handling of extensibility

There was great uncertainty connected to both the usage and the inclusion of metrics. Metrics, therefore, became an important point for extensibility.

New metrics were designed to enter the system in two ways. The first was to combine existing metrics from already implemented IMetricProviders to new metrics. This was done in the configuration for each exercise, details for how this was implemented can be seen in section 5.3.1. The second way was to add an IMetricProvider to the classpath and its name to the *application.yml* file. The *application.yml*, encoded in Yaml is one of several formats the global configuration file for a Spring project can have. Yaml was chosen because of its human readable format for structuring information. The new IMetricProvider would be found upon restart and usable for future configurations. The reason for having to add the IMetricProvider name to the *application.yml* file was to easily be able to remove it without removing it from the project classpath, which is a more tedious process than simply commenting it out from the configuration file.

5.3.1 Configuration with EMF

Modeling with EMF was used to enable configuration of existing metrics from implemented IMetricProviders. A configurable array of measures was used to aggregate new measures and therefore relive the need for a lot of custom created measures. More specifically, the feature vector model, described in section 5.2.1, define operations on dummy data that would be replaced at runtime. This method of configuration resembles a combination of routine configuration and creative construction, as the fv-model lays out the operations and connection types the model elements can have while still dictating rules for the tree-like structure. In the simplified example seen in figure 5.3, object #6 and #7 would be swapped out with student data. The addition of metadata was necessary because measure names were only unique within an IMetricProvider. The ExpressionFeatures act as a selector in the example, but other structures like FilteredFeature or more complex use of ExpressionFeatures could also be used.

To traverse and recognize references to dummy data, EMF's support for multiple resources was used. The configuration file consists of two resources, as can be seen in figure 5.4. Resource 1 consists of the various operations made on the data contained in recourse 2. To use this as a configuration the references made from resource 1 to resource 2 had to be identified, as can be seen as the black arrows crossing the dotted line in figure 5.4, and replaced with references to the correct user data, indicated with dotted arrows in 5.4. The replacement depended upon dummy data in the configuration resource set and the student

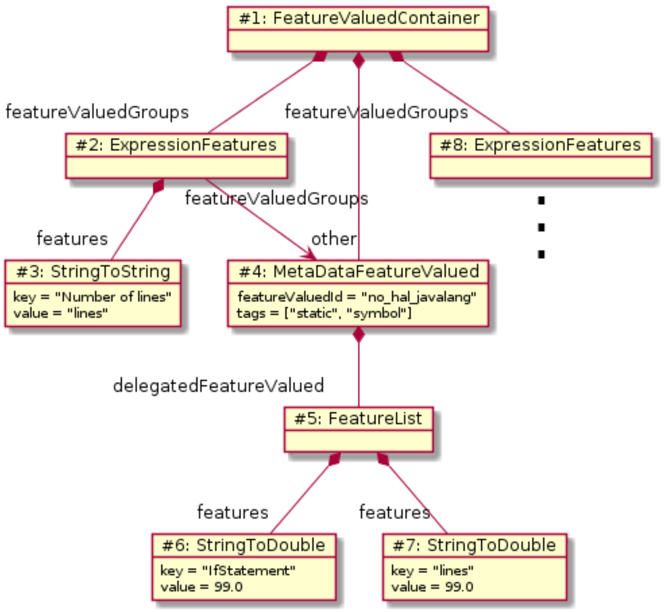


Figure 5.3: Configuration instance

data was contained by a FeatureValuedContainer with the same name to be able to work.

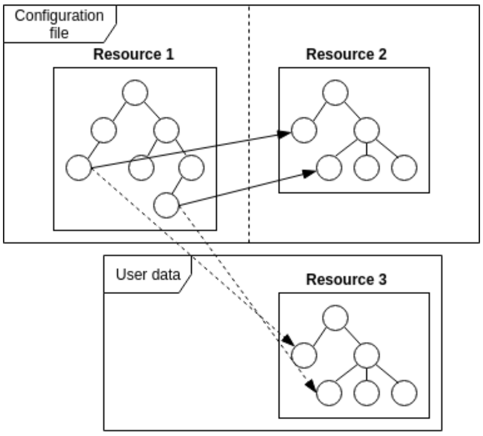


Figure 5.4: Finding references in eObjects

Due to the flexibility of the fv-model, achievements were also awarded based on the existence of a FeatureList after a series of filters had been applied to the student data. This also allowed for achievements spanning multiple metrics or composite metrics.

5.4 Program architecture

This section describes the architecture through different views in the 4+1 architectural view model. To build the code, follow the instructions in appendix E.

5.4.1 Scenarios

Scenario 1 deals with how a student would typically use the application and a BPMN diagram with the omission of checking achivements can be seen in figure 5.5. Scenario 2 on the other hand look at how a course responsible might use the application.

Scenario 1

The student login with Feide in the application. The system will save the user id for this user locally and use this when submitting code (.ex files) and metrics. The student works on the exercise and solves two out of seven unit tests. The student is responsible for uploading the source code for the exercise to the metrics application. Upon logging in to the metrics application in a browser, the student can now see that the exercise is not completed and does not have the same; cyclomatic complexity, number of for loops, and number of exceptions as the solution guide. The student goes back to working on the assignment and pass the remaining unit tests. The student then re-uploads the code. When checking the metrics application again, the student can see that the exercise does not have the same cyclomatic complexity, but has the same number of for loops, and number of exceptions as the solution guide. The student can also see that a silver badge has been awarded for completing the assignment. To improve the badge, the student continues to work on the assignment and remove some unnecessary if and else statements. Upon re-uploading and checking the metrics application again, the student can see that a gold badge has been awarded and that the exercise has the same cyclomatic complexity, number of for loops, and number of exceptions as the solution guide.

Scenario 2

The course responsible logs in to the system and uploads the new configuration for a particular exercise. The file being uploaded is an XMI file. Upon completion of the upload the course responsible visits the page for the exercise he/she uploaded the new config and verify that the new config is in effect for the solution manual.

5.4.2 Logical view

A simplified class diagram can be seen in appendix C where some of the packages have been collapsed for brevity as they contain mostly small and simple objects. The 'Package models' for instance contain the various POJOs used by the controllers and the 'Package repositories' contain interfaces for Spring Data repository abstraction.

For all requests accessing non-static resources, it is assumed that the user has logged in and include a jsessionid with the request. The authentication process utilizes OAuth 2 implicit flow and has been delegated to Uninett Dataporten. This enables the user to log in with their Feide credentials as Feide merged with Dataporten in 2018 [39]. To avoid recording unnecessary identifying information, only required scopes for identifying users were enabled for the authentication procedure. These scopes were; e-mail, username, profile image, and user-ids. The benefit with Uninett Dataporten is that students and TA's do not have to create a new user profile for the application.

5.4.3 Process view

The BPMN diagram in figure 5.5 is included to give some context as to when the included activity diagrams are used. The use case in figure 5.5 is the same as in 4.1, with the omission of checking achievements.

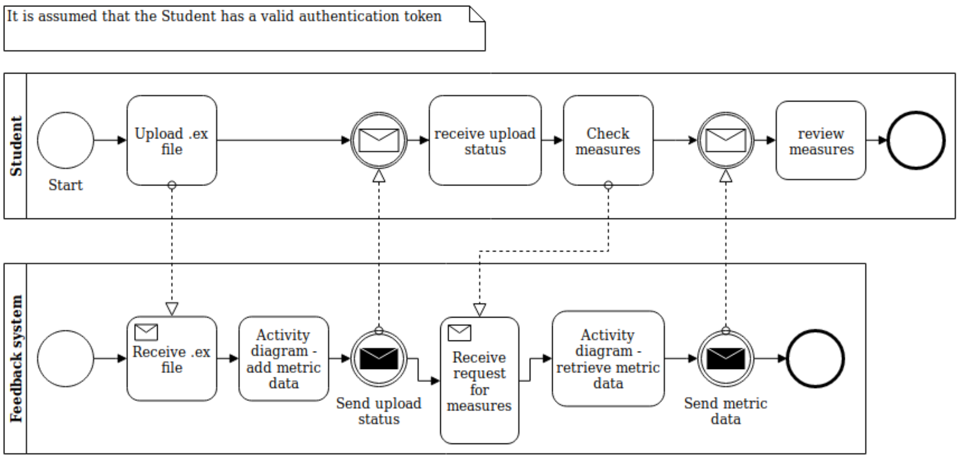

It is assumed that the Student has a valid authentication token
Student
Start
Upload .ex file
receive upload status
Check measures
review measures
Feedback system
Receive .ex file
Activity diagram - add metric data
Send upload status
Receive request for measures
Activity diagram - retrieve metric data
Send metric data

Figure 5.5: Usage example with BPMN

Only the activities with the most steps are included in this section for brevity, as many of the activities do simple operations like store or retrieve data with access control without any further processing.

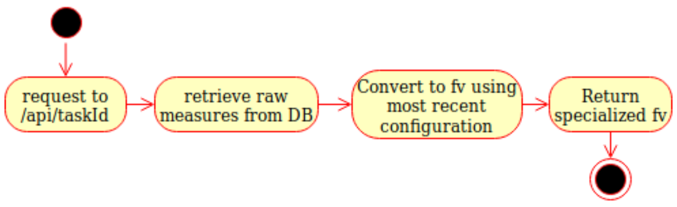

request to /api/taskId
retrieve raw measures from DB
Convert to fv using most recent configuration
Return specialized fv

Figure 5.6: Activity diagram - retrieve metric data

In figure 5.6 the configuration resource set is retrieved and its data resource replaced with student metrics. This method of retrieving calculated metrics ensure that the newest configuration is always used.

The most complex single operation within the program is adding new metric data by uploading a .ex file. In figure 5.7 the ”calculate metrics” segment can be broken down

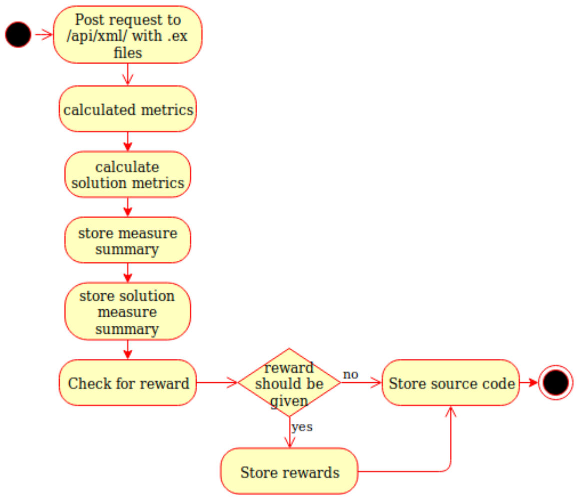


Figure 5.7: Activity diagram - add metric data

into two steps. The first step is to extract the source code by serializing the payload and searching for the final proposal for each task. The second step is looking up available IMetricProviders on the classpath and combine the different metrics to one measure summary for the entire exercise. The same process is repeated with a variation for calculating metrics for the solution. Instead of extracting source code from the provided content, the source code is retrieved from the database. Metrics for solutions must be calculated for each submission as every proposal from the student can contain a different combination of files.

Figure 5.8 show how new metrics enter the system with a sequence diagram. The reason for storing the .ex file is to ensure that future metrics can be calculated without re-uploading anything.

5.4.4 Development view

In figure 5.9 it is important to underline the possibility of multiple IMetricProviders. Another important note is the exclusion of spring-boot components used by the *main* component.

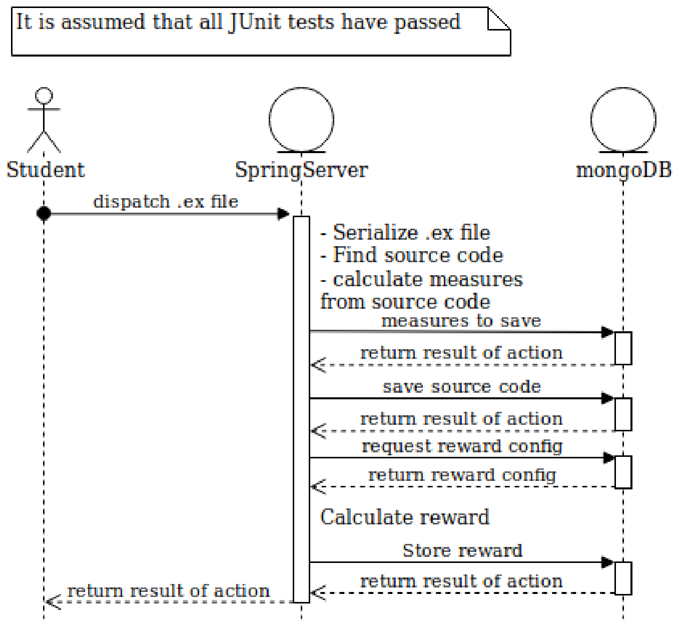


Figure 5.8: Sequence diagram add metric data

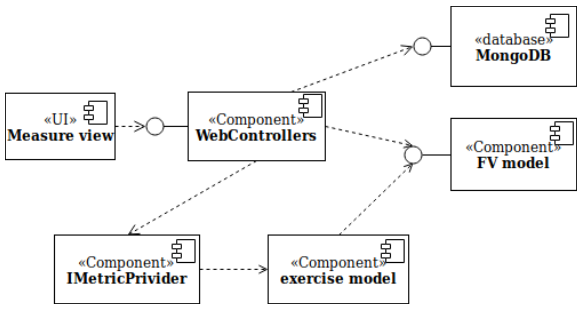

«database» MongoDB
«UI» Measure view
«Component» WebControllers
«Component» FV model
«Component» IMetricPrivider
«Component» exercise model

Figure 5.9: Component diagram

5.4.5 Physical view

The server containing Nginx and the Spring application in figure 5.10 was a cloud-hosted DigitalOcean server with 2GB of RAM and 1 vCPU. Both Uninet Feide and Google analytics provide free access at low usage volumes. The MongoDB server was also hosted on the DigitalOcean server.

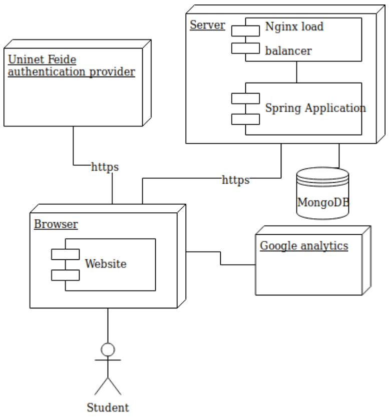


Figure 5.10: Deployment diagram

Chapter 6

Results and discussions

The results from the testing period will be presented and discussed in this chapter together with threats to validity. The necessary definitions from Google Analytics will also be explained. Finally, the chapter presents a discussion of the developed system.

6.1 Procedure

The experiment was first announced on March 17th during a lecture with an additional announcement on the course e-learning platform. Follow-up announcements on the e-learning platform were also issued during the experiment. The participants were asked to login with their Feide user and try out the system and answer a short survey. No further instructions or assistance were given to the students on how to use the system. On April 8th Food coupons were added as an incentive for answering the survey. The experiment ended on April 17th.

6.2 Evaluation of collected usage statistics

The terms; sessions, users, and IDs in DB are needed to present the results and are defined as follows:

A session represents a bundle of actions the user performs while actively engaging with the web page. A break from interacting with the web page for more than 30 minutes is considered the end of one session. Sessions can include; page views, actions, and other events like e-commerce transactions [40].

Users are defined in Google Analytics as someone who has initiated one or more sessions. By default, users are distinguished by a cookie created during their first visit, this is called unassigned user-ID allocation. When the user log in their user-ID gets overwritten with a new one that is unique for the logged in user. This is called assigned user-ID allocation and enables persistent tracking across devices. IDs in DB is a count of how many logged in users the system received. The number of; sessions, users, and IDs in DB can be found in table 6.1. Google Analytics also reviled eight users accessing the site from other countries than Norway, suggesting that some visitors were bots.

Table 6.1: Visitors to https://metricstdt4100.xyz

Sessions	Unassigned users	Assigned Users	IDs in DB
155	65	59	72

The mismatch of the number of users with assigned user-IDs and the number of IDs in DB suggests that 13 users that logged in likely ran some kind of script blocker as this number should be the same if the Google Analytics script loaded correctly. The average session duration distribution can be seen in table 6.2. Only 58 sessions lasted longer than

Table 6.2: Session duration distribution

Session Duration	Sessions	Pageviews
0-10 seconds	68	146
11-30 seconds	29	152
31-60 seconds	24	203
61-180 seconds	20	202
181-600 seconds	4	73
601-1800 seconds	7	150
1801+ seconds	3	446
Total	155	1372

30 seconds and 34 sessions longer than 60 seconds. Google Analytics also reported 11 users accessing the site from a mobile device where none of them logged in.

6.2.1 Usage objectives

The usage objectives and corresponding completion rate can be seen in table 6.3. The method for deriving the various metrics are also included in the third column. Results from

Table 6.3: Usage objectives

Number	**Usage objective**	Value	Derived by
1	User has visited at least one summary page	16	Google Analytics
2	User has visited the achievement page	24	Google Analytics
3	User has uploaded one .ex file	23	MongoDB
4	User has uploaded more than one .ex file	13	MongoDB

Google Analytics were extracted by creating custom views for users and content groups for the respective objective. The results from MongoDB were extracted by executing an aggregation on the collection *measureSummary*. The aggregation used can be seen in apendix B. It is important to point out that the completion criteria for objective 2 was reachable for authenticated users.

6.3 SUS score

The SUS survey received 26 respondents and got an average SUS score of 63.84, which is lower than the average 68 which is considered to be an acceptable score. The standard deviation was 16.94%.

6.4 Discussion of the developed system

The developed system made it easy to add and configure measures for exercises and achievements. Especially the ability to combine an array of measures to a new measure decreased the need for hard-coded measures considerably, facilitating objective 3 from section 1.4. There are however challenges and threats to the system's longevity and potential for extended use. The first challenge is the handover to future administrators. While the course responsible and teaching assistants for TDT4100 are well acquainted with Java programming, the specific technology stack has not been lectured at NTNU. To avoid some of the friction related to the handover the system could have used a packaging module system known to the future system administrators, an example of one such system is OSGi. OSGi would also have made for easier interoperability with the already created IMetricProvider as it already is an OSGi bundle.

One threat to the system is to its potential for extended use. The system relies upon consuming .ex files created with the Jexercise extension for Eclipse. Institutions that wish to give students similar feedback would have to implement a similar lab work routine where exercises are written in Eclipse. This limits the potential for extended use as preferences for integrated development environments are strong.

Another threat is that a more universal process for capturing exercise related data is already in the works. This initiative is led by the Standards, Protocols, and Learning Infrastructure for Computing Education (SPLICE) project. The working group behind ProgSnap2 within the SPLICE project aims to create a standard for capturing student submissions in a CSV format [41]. The format is arguably easier to use for researchers that are not proficient in EMF modeling.

The system is missing some important features to become completely self-sustained. The current implementation for changing solution manuals involves swapping out the affected files in a file structure and run the system with custom command-line arguments to detect the changes. A more graceful solution would allow HTTP routes to change solution manuals. Alternatively, the system could be instructed to watch a specific GitHub repository with solution manuals for changes. The last alternative is feasible as the course already has a maintained repository for solution manuals.

Lastly, the system is missing more extensive testing. While the system had integration tests written in the Postman API development environment and web controller integration tests automatically testing the most critical features upon every build, there were too few. The shortage of integration tests meant that some endpoints were not tested extensively enough. Unit tests could also have been valuable when developing the shared helper classes.

6.5 Discussion of results from the experiment

The lower than average SUS score and lack of people completing usage objective 4 indicate that the system did not achieve objective 2 from section 1.4. There are many potential reasons for this, one of which is related to motivation for use. This is backed by the score of individual questions in the SUS survey. As can be seen in appendix C, question 1, 'I think that I would like to use this system frequently', received by far the lowest score. The SUS survey is not made for analyzing scores on individual questions and this should therefore be a topic for further research. From the number of people actually uploading a .ex file in comparison to the number of authenticated users, it is clear that uploading an exercise is hard.

Table 6.1 revealed 13 users with script blocking extensions. Although this does not impact the results aggregated from the database, it could impact the measurement of the usage objective #1 in both directions. This is also discussed in greater detail below in section 6.5.1.

The actual numbers of users, 111 from Norway, were also relatively low in comparison to the number of unique users logging into the system, 72 from whom only 23 users uploaded .ex files. This highlights one important challenge with the system, getting users to upload files requires considerable effort.

The survey had a turnout of 26 which is higher than the 23 users actually uploading a .ex file. This clearly indicates that 3 people did not test the application to any considerable extent before answering the survey. One possible reason for this is the added incentive of food coupons at the end of the experiment, tempting people that have not tried the system to answer only to be eligible to win coupons.

6.5.1 Threats to validity

One apparent threat to validity is the relatively low participation among students. While the number of responses for the survey was above worst-case scenario, the confidence interval was 18.93 for a confidence level at 95% with 26 respondents and a population of 835. Some of the reasons for the low number of participants and measures to improve them are discussed below in section 7.1.

The incentive program for participation had obvious issues as the survey did not have any validation of respondents having actually tried the system. This could have led to people answering the questionnaire without trying the system beforehand. The usage data does not support this as there were many more visiting the system than answering the survey, but there was no way to confirm this. Additionally, any incentive program can attract responses on the wrong premises.

There is also a risk of Google analytics data being inaccurate in terms of gathering usage across devices and not recording usage patterns at all for some users. The program utilized id's aimed at enabling tracking behavior across devices, but script or content blocking extensions can prevent this kind of data gathering. The practice of blocking certain scripts has become more common and is even included in the Firefox browser, but not turned on by default [42]. By querying the database and looking up how many unique users had logged in and compare that to the number of users recorded by Google Analytics, it was found that 13 people used script blocking scripts which is a considerable number of users. The number of users accomplishing user objective #1 and #3 reveal that some of the most active users probably used script blocker features as it is unlikely that 7 users would choose not to examine the calculated measures, but still upload a .ex file.

Chapter 7

Conclusion

In this study a system for gamifying the lab work of TDT4100 through code measures was created and its usability tested. The system enabled students to compare a configured set of code measures from their own exercises to measures from a solution manual. Students could also complete achievements for single exercises or for all delivered exercises. To avoid the need for additional user accounts, Feide was used as authentication provider. Data about user activities were monitored in an unintrusive manner and used in the analysis of the system's usability.

The system was designed with extensibility and configurability in focus and had the ability to; configure which measures to display for an exercise, add measures later in development, and create composite measures. The method used for configuration relied upon the Ecore API for manipulation of eObjects and demand loading of resources. The same technique was also used to create requirements for achievements in the game.

The experiment investigated the usability of the system by utilizing automatic tracking of user behavior and a questionnaire with questions from SUS. Students could test the system for 40 days and spanned the period of three deliveries. The experiment had a large number of entrants, however, a relatively low number of participants used the system beyond logging in to the system. The results from the SUS score and the number of people completing the usage objectives indicated that the system did not have a high degree of usability. The results also show that script-blocking features in browsers have become more prevalent.

7.1 Further research

The goal is to improve qualities beyond runnable code with correct behavior, extensive feedback could provide much of the same information without the competitiveness of a game. Further research could look into the need for gamification as opposed to just extended feedback through proper A/B testing. Another possibility is to explore other; game elements, reward structures, or game rules.

The implemented solution required the student to manually upload the .ex file which does not provide immediate feedback and demand considerable action from the student. Further research could investigate if automatically updating measures, decreasing time until feedback is given result in improved code quality.

In section 2.4 it is revealed that understandable and useful feedback in terms of performance is important for the success of gamification. The participants were likely to be at an early stage in their programming career and the understanding of code measures might have been low. To explore whether programming experience impacts the usefulness of feedback in the form of gamifying measures, further research can investigate the appreciation of gamification with code measures between novice and skilled programmers.

To improve participation, one could incorporate the feedback system as a part of the lab work. This poses obvious challenges as forcing students to use yet another system might add another layer of complexity to lab work proving already difficult. A different option could be to introduce the feedback system at the start of the semester and remind students to use it at the end of each exercise. There is also the option of adding more incentives.

Tools like Google Analytics provide efficient methods for collecting usage data but are also considered to be privacy intrusive. Google Analytics, in particular, anonymize the collected data and retain the rights to use some of it for internal purposes. Future experiments on usability should, therefore, take into consideration script blocking extensions and the effect this can have on the collected data.

7.2 Further development

As mentioned in section 7.1 automating the delivery of exercises and the elicitation of measures could be an interesting change to the system. To achieve this, deliveries could be handled by the measure system alone and use the Blackboard API to deliver the exercise to Blackboard on behalf of the student. There are however limitations to this approach as the Blackboard version dictates available API calls.

Further work on the system should set strict targets for test coverage as the discussion on testing in section 6.4 revile that there is a need for further testing.

To make it easier to create configurations for which measures to display and aggregate, a graphical editor could be implemented as an eclipse extension. The extension could be created with Sirius, an Eclipse project for creating graphical modeling workbenches [43]. An editor would make it easier to create valid configurations.

To address the concerns related to longevity in section 6.4 the system could be extended with the ability to transform ”ProgSnap2” formatted exercise submissions to exercise model instances.

Bibliography

[1] Norges teknisk-naturvitenskapelige universitet, "TDT4100 - Objektorientert programmering." `https://www.ntnu.no/studier/emner/TDT4100#tab=omEmnet`, 2011. [Online; accessed 26-May-2019].

[2] K. Kapp, *The Gamification of Learning and Instruction: Game-based Methods and Strategies for Training and Education*. Pfeiffer essential resources for training and HR professionals, Wiley, 2012.

[3] N. A. Mokadam, R. Lee, A. A. Vaporciyan, J. D. Walker, R. J. Cerfolio, J. L. Hermsen, C. J. Baker, R. Mark, L. Aloia, D. H. Enter, A. J. Carpenter, M. R. Moon, E. D. Verrier, and J. I. Fann, "Gamification in thoracic surgical education: Using competition to fuel performance.," *The Journal of thoracic and cardiovascular surgery*, vol. 150, pp. 1052–8, 11 2015.

[4] R. van Roy and B. Zaman, "Need-supporting gamification in education: An assessment of motivational effects over time," *Computers and Education*, vol. 127, no. August, pp. 283–297, 2018.

[5] C. Rasmussen and D. Åse, "A Web-Based Code-Editor." `http://hdl.handle.net/11250/2352227`, 2014. Master thesis accepted NTNU IDI.

[6] E. S. Stenmark, "Spillifisering for å stimulere utvikling av praktiske ferdigheter i Eclipse." `http://hdl.handle.net/11250/2352241`, 2015. Master thesis accepted NTNU IDI.

[7] S. Bolstad, "Gamifying TDT4100." `http://hdl.handle.net/11250/2579067`, 2018. Master thesis accepted NTNU IDI.

[8] H. Kniberg, *Scrum and XP from the trenches : how we do scrum*. InfoQ enterprise software development series, S.l.: C4Media, 2007.

[9] P. Kruchten, "Architecture Blueprints," *IEEE Software*, vol. 12, no. November, p. 540555, 1995.

[10] a. V. H. H. N. Z. E. . Hallvard Trætteberg, Vemund Santi, "Jexercise." https://github.com/hallvard/jexercise, 2019.

[11] G. D. P. Regulation, "Regulation (eu) 2016/679 of the european parliament and of the council of 27 april 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing directive 95/46," *Official Journal of the European Union (OJ)*, vol. 59, no. 1-88, p. 294, 2016.

[12] H. Trætteberg, T. T. Iveland, R. Sætre, O.-M. Pedersen, and V. M. Santi, "Faginnhold." https://www.ntnu.no/wiki/display/tdt4100/, 2019. [Online; accessed 26-May-2019].

[13] N. Fenton and J. Bieman, *Software Metrics: A Rigorous and Practical Approach, Third Edition*. Boca Raton, FL, USA: CRC Press, Inc., 3rd ed., 2014.

[14] S. R. Chidamber and C. F. Kemerer, "A metrics suite for object oriented design," *Tse*, vol. 20, no. 6, pp. 476–493, 1994.

[15] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*. Addison-Wesley Professional, 3rd editio ed., 2012.

[16] P. Webb, D. Syer, J. Long, S. Nicoll, R. Winch, A. Wilkinson, M. Overdijk, C. Dupuis, S. Deleuze, M. Simons, V. Pavić, J. Bryant, and M. Bhave, "Spring Boot Reference Guide." https://docs.spring.io/spring-boot/docs/2.0.5.RELEASE/reference/htmlsingle/, 2018. [Online; accessed 26-May-2019].

[17] ISO - International Organization for Standardization, "Systems and software engineering Systems and software Quality Requirements and Evaluation (SQuaRE) System and software quality models," *Iso/Iec Fdis 25010:2011*, vol. 2010, pp. 1–34, 2011.

[18] T. J. McCabe, "A Complexity Measure," *IEEE Transactions on Software Engineering*, vol. SE-2, no. 4, pp. 308–320, 1976.

[19] Ieee, *IEEE Std 982.1 - 2005 IEEE Standard Dictionary of Measures of the Software Aspects of Dependability*, vol. 2005. 2006.

[20] W. P. Stevens, G. J. Myers, and L. L. Constantine, "Structured design," *IBM Systems Journal*, vol. 13, no. 2, pp. 115–139, 1974.

[21] Eclipse Contributors and others, "Class ASTNode." https://help.eclipse.org/neon/index.jsp?topic=%2Forg.eclipse.jdt.doc.isv%2Freference%2Fapi%2Forg%2Feclipse%2Fjdt%2Fcore%2Fdom%2FASTNode.html. [Online; accessed 26-May-2019].

[22] S. Deterding, D. Dixon, R. Khaled, and L. Nacke, "From game design elements to gamefulness," in *Proceedings of the 15th International Academic MindTrek Conference on Envisioning Future Media Environments - MindTrek '11*, (New York, New York, USA), p. 9, ACM Press, 2011.

[23] C. R. Prause, J. Nonnen, and M. Vinkovits, “A field experiment on gamification of code quality in agile development,” in *Psychology of Programming Interest Group Annual Conference (PPIG)*, vol. 2012, Citeseer, 2012.

[24] M. Brambilla, J. Cabot, and M. Wimmer, “Model-Driven Software Engineering in Practice,” *Synthesis Lectures on Software Engineering*, vol. 1, pp. 1–182, 9 2012.

[25] M. Völter, T. Stahl, J. Bettin, A. Haase, S. Helsen, K. Czarnecki, and B. von Stockfleth, *Model-Driven Software Development: Technology, Engineering, Management*. Wiley Software Patterns Series, Wiley, 2013.

[26] M. Brambilla, J. Cabot, M. Wimmer, and L. Baresi, *Model-Driven Software Engineering in Practice: Second Edition*. Synthesis Lectures on Software Engineering, Morgan & Claypool Publishers, 2017.

[27] Miro Sponemann, Christian Dietrich, Tamas Miklossy, and Holger Schill, “The Grammar Language.” `https://www.eclipse.org/Xtext/documentation/301_grammarlanguage.html`. [Online; accessed 26-May-2019].

[28] E. Toporov, “Step Into the Future with MPS 1.0.” `https://blog.jetbrains.com/mps/2009/07/step-into-the-future-with-mps-10/`, 2009. [Online; accessed 26-May-2019].

[29] S.r.o. JetBrains, “How Does MPS Work?.” `https://www.jetbrains.com/mps/concepts/`. [Online; accessed 26-May-2019].

[30] D. Steinberg, *EMF : Eclipse Modeling Framework*. The eclipse series EMF, Place of publication not identified: Addison Wesley, 2nd ed. ed., 2009.

[31] E. Fundation, “Ecore.” `https://wiki.eclipse.org/Ecore`, 2019. [Online; accessed 26-May-2019].

[32] P. R. W. Fred D Davis, Richard P Bagozzi, “User acceptance of computer technology: a comparison of two theoretical models,” vol. 35, no. 8, pp. 982–1003, 1989.

[33] N. Marangunić and A. Granić, “Technology acceptance model: a literature review from 1986 to 2013,” *Universal Access in the Information Society*, vol. 14, no. 1, pp. 81–95, 2015.

[34] A. Aypay, H. C. Çelik, A. Aypay, and M. Sever, “Technology acceptance in education: A study of pre-service teachers in Turkey,” *Turkish Online Journal of Educational Technology*, vol. 11, no. 4, pp. 264–272, 2012.

[35] V. Venkatesh, “Creation of Favorable User Perceptions: Exploring the Role of Intrinsic Motivation,” *MIS Quarterly*, vol. 23, no. 2, p. 239, 2006.

[36] N. Harrati, I. Bouchrika, A. Tari, and A. Ladjailia, “Exploring user satisfaction for e-learning systems via usage-based metrics and system usability scale analysis,” *Computers in Human Behavior*, vol. 61, pp. 463–471, 8 2016.

[37] P. Jordan, B. Thomas, I. McClelland, and B. Weerdmeester, *Usability Evaluation In Industry*. Taylor & Francis, 1996.

[38] I. S. Venticinque, "Determining What Individual SUS Scores Mean: Adding an Adjective Rating Scale," *Journal of Usability Studies*, vol. 4, no. 3, pp. 114–123, 2009.

[39] Feide, "Feide innlogging." `https://www.uninett.no/feideinnlogging`, 2018. [Online; accessed 26-May-2019].

[40] Google, "How a web session is defined in Analytics." `https://support.google.com/analytics/answer/2731565?hl=en`. [Online; accessed 26-May-2019].

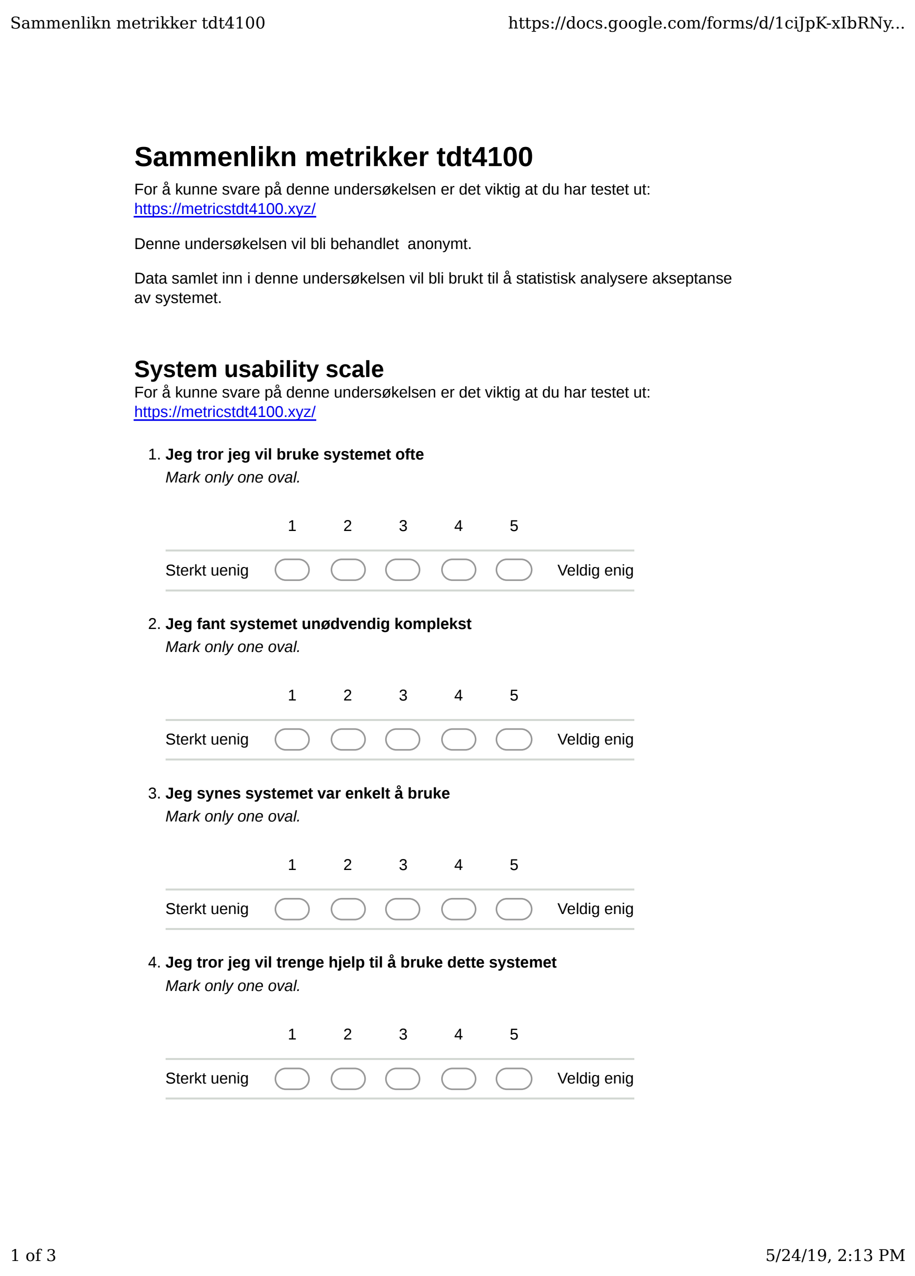
[41] SPLICE, "SPLICE: Standards, Protocols, and Learning Infrastructure for Computing Education." `https://cssplice.github.io/`. [Online; accessed 26-May-2019].

[42] J. F. AliceWyman, Michele Rodaro, "Firefox - Content blocking." `https://support.mozilla.org/en-US/kb/content-blocking`. [Online; accessed 26-May-2019].

[43] Sirius, "Sirius." `https://www.eclipse.org/sirius/doc/`. [Online; accessed 26-May-2019].

Appendices

Appendix A



Sammenlikn metrikker tdt4100 https://docs.google.com/forms/d/1ciJpK-xIbRNy...

Sammenlikn metrikker tdt4100

For å kunne svare på denne undersøkelsen er det viktig at du har testet ut: https://metricstdt4100.xyz/

Denne undersøkelsen vil bli behandlet anonymt.

Data samlet inn i denne undersøkelsen vil bli brukt til å statistisk analysere akseptanse av systemet.

System usability scale

For å kunne svare på denne undersøkelsen er det viktig at du har testet ut: https://metricstdt4100.xyz/

1. **Jeg tror jeg vil bruke systemet ofte**
Mark only one oval.

	1	2	3	4	5	
Sterkt uenig	◯	◯	◯	◯	◯	Veldig enig

2. **Jeg fant systemet unødvendig komplekst**
Mark only one oval.

	1	2	3	4	5	
Sterkt uenig	◯	◯	◯	◯	◯	Veldig enig

3. **Jeg synes systemet var enkelt å bruke**
Mark only one oval.

	1	2	3	4	5	
Sterkt uenig	◯	◯	◯	◯	◯	Veldig enig

4. **Jeg tror jeg vil trenge hjelp til å bruke dette systemet**
Mark only one oval.

	1	2	3	4	5	
Sterkt uenig	◯	◯	◯	◯	◯	Veldig enig

1 of 3 5/24/19, 2:13 PM

5. **Jeg fant funksjonene i dette systemet godt integrert**
Mark only one oval.

	1	2	3	4	5	
Sterkt uenig	○	○	○	○	○	Veldig enig

6. **Jeg synes det var for mye inkonsistens i dette systemet**
Mark only one oval.

	1	2	3	4	5	
Sterkt uenig	○	○	○	○	○	Veldig enig

7. **Jeg ser for meg at de fleste vil kunne lære seg å bruke dette systemet raskt**
Mark only one oval.

	1	2	3	4	5	
Sterkt uenig	○	○	○	○	○	Veldig enig

8. **Jeg fant systemet veldig tungvint å bruke**
Mark only one oval.

	1	2	3	4	5	
Sterkt uenig	○	○	○	○	○	Veldig enig

9. **Jeg følte meg veldig selvsikker med å bruke systemet**
Mark only one oval.

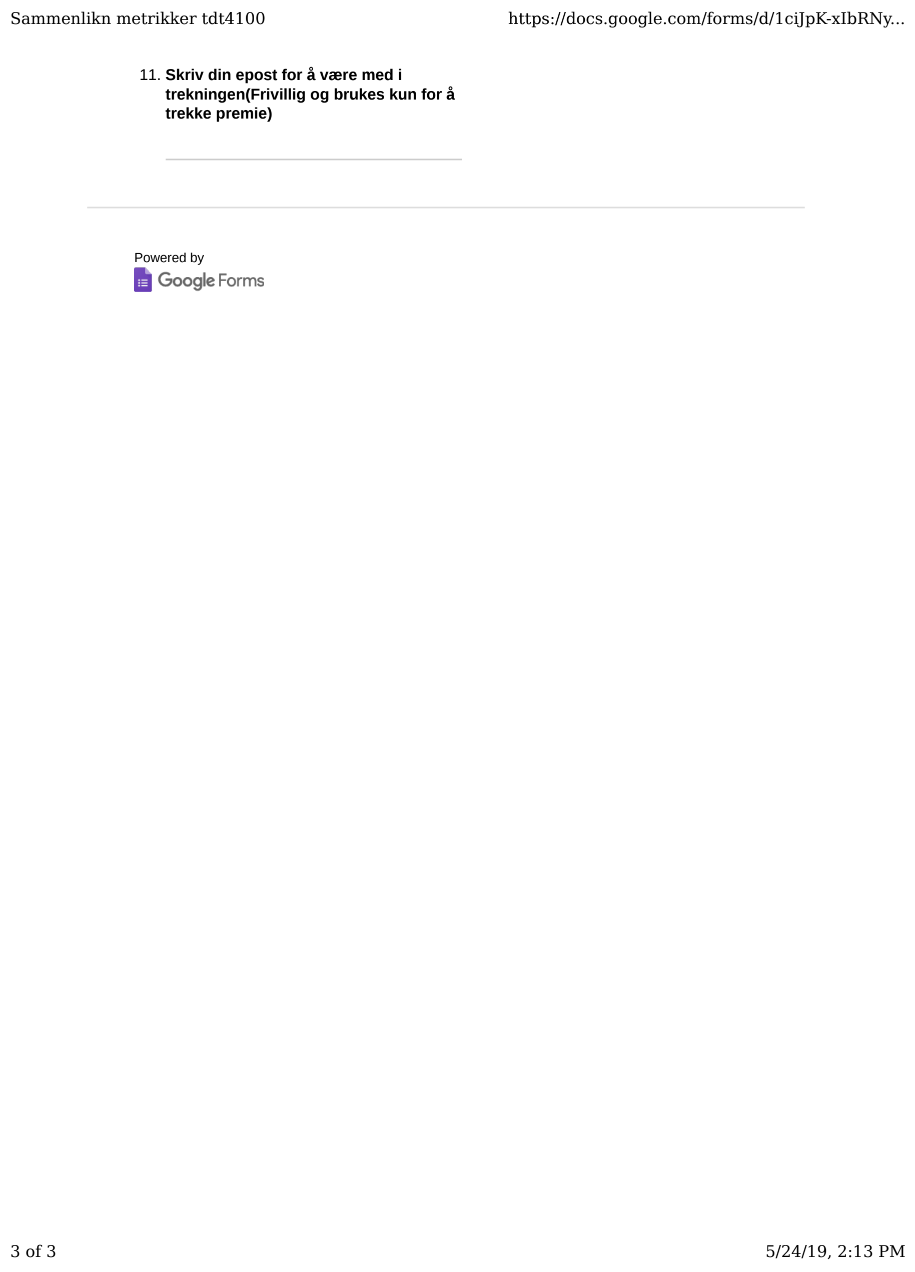
	1	2	3	4	5	
Sterkt uenig	○	○	○	○	○	Veldig enig

10. **Jeg var nødt til å lære mye nytt for å kunne begynne å bruke systemet**
Mark only one oval.

	1	2	3	4	5	
Sterkt uenig	○	○	○	○	○	Veldig enig

Vinn gavekort

Bli med i trekningen av gavekort hos SiT på 75 kr. Det er 7 premier totalt.



Sammenlikn metrikker tdt4100 https://docs.google.com/forms/d/1ciJpK-xIbRNy...

11. **Skriv din epost for å være med i trekningen(Frivillig og brukes kun for å trekke premie)**

Powered by
Google Forms

3 of 3 5/24/19, 2:13 PM

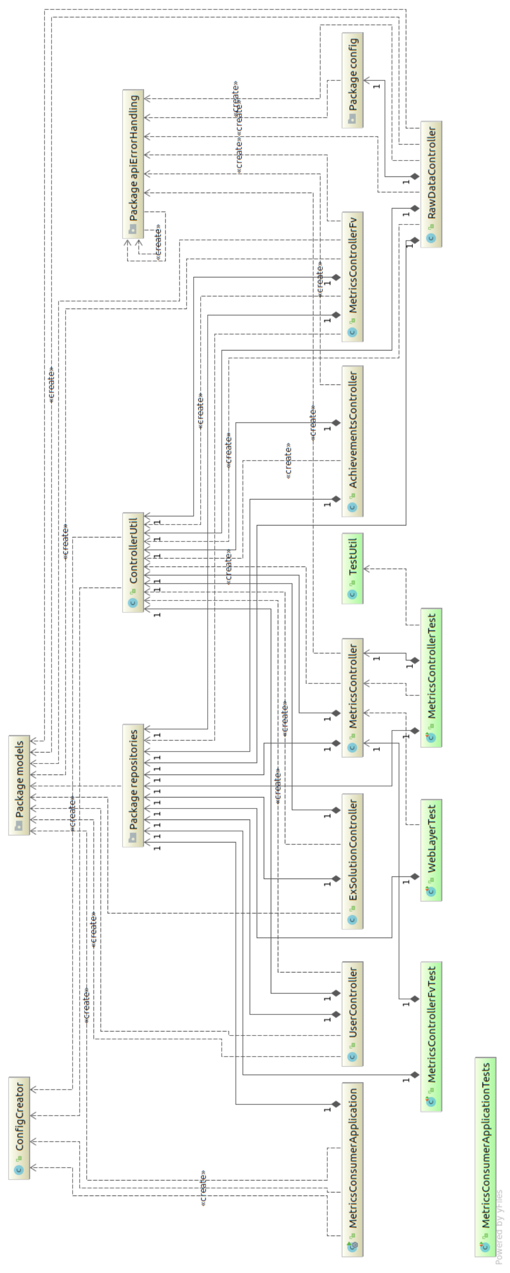
Appendix B

MongoDB aggregations

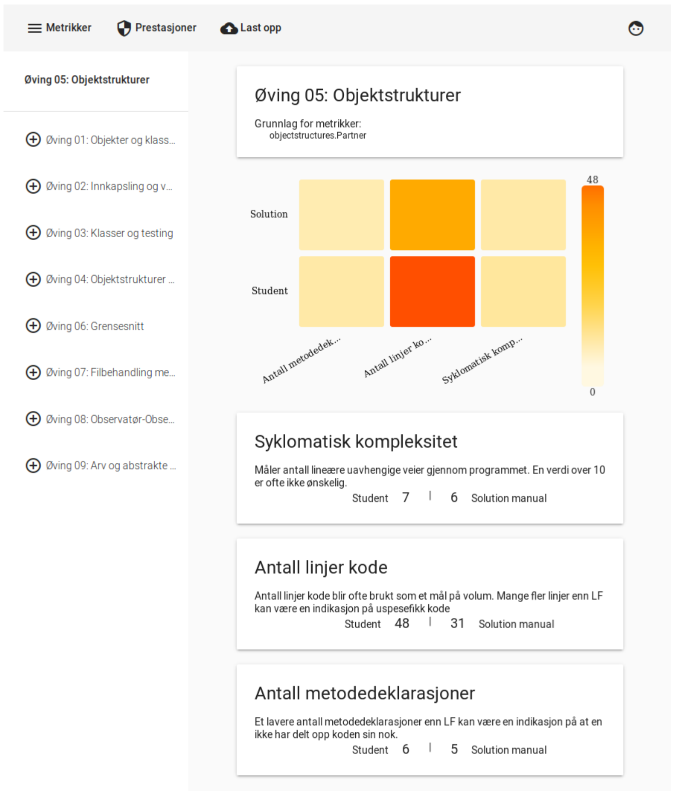
```
[
  {
    $project: {
      userId: 1,
      taskId: 1
    }
  }, {
    $sort: {
      userId: 1
    }
  }, {
    $group: {
      _id: "$userId",
      count: {
        $sum: 1
      },
      numTasks: {
        $addToSet: "$taskId"
      }
    }
  }, {
    $project: {
      _id: "$_id",
      numTasksInt: {
        $size: "$numTasks"
      }
    }
  }, {
    $sort: {
      numTasksInt: -1
    }
  }
]
```

Appendix C

Class diagram



Appendix D



Metrikker
Prestasjoner
Last opp

Øving 05: Objektstrukturer

- Øving 01: Objekter og klass...
- Øving 02: Innkapsling og v...
- Øving 03: Klasser og testing
- Øving 04: Objektstrukturer ...
- Øving 06: Grensesnitt
- Øving 07: Filbehandling me...
- Øving 08: Observatør-Obse...
- Øving 09: Arv og abstrakte ...

Øving 05: Objektstrukturer

Grunnlag for metrikker:
objectstructures.Partner

Syklomatisk kompleksitet

Måler antall lineære uavhengige veier gjennom programmet. En verdi over 10 er ofte ikke ønskelig.

Student 7 | 6 Solution manual

Antall linjer kode

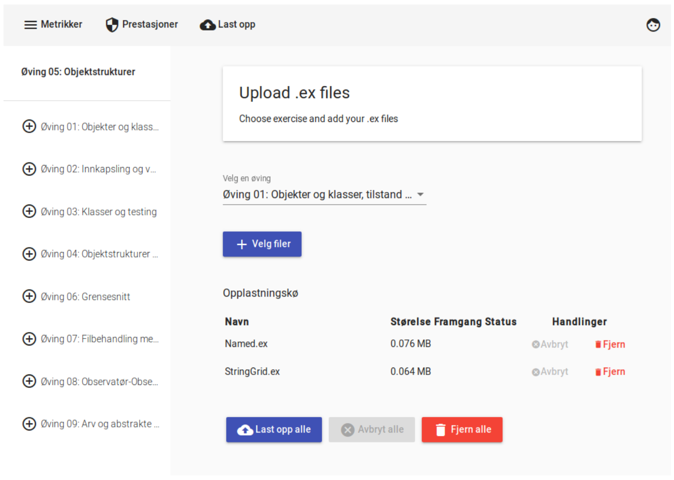
Antall linjer kode blir ofte brukt som et mål på volum. Mange fler linjer enn LF kan være en indikasjon på uspesefikk kode

Student 48 | 31 Solution manual

Antall metodedeklarasjoner

Et lavere antall metodedeklarasjoner enn LF kan være en indikasjon på at en ikke har delt opp koden sin nok.

Student 6 | 5 Solution manual



Metrikker
Prestasjoner
Last opp

Øving 05: Objektstrukturer

- Øving 01: Objekter og klass...
- Øving 02: Innkapsling og v...
- Øving 03: Klasser og testing
- Øving 04: Objektstrukturer ...
- Øving 06: Grensesnitt
- Øving 07: Filbehandling me...
- Øving 08: Observatør-Obse...
- Øving 09: Arv og abstrakte ...

Upload .ex files

Choose exercise and add your .ex files

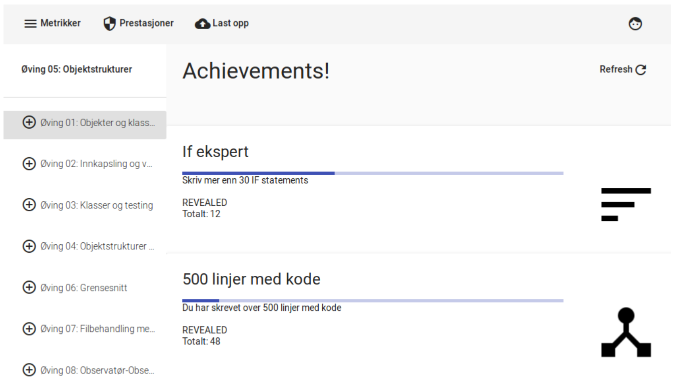
Velg en øving
Øving 01: Objekter og klasser, tilstand ...

+ Velg filer

Opplastningskø

Navn	Størelse	Framgang	Status	Handlinger
Named.ex	0.076 MB			Avbryt Fjern
StringGrid.ex	0.064 MB			Avbryt Fjern

Last opp alle
Avbryt alle
Fjern alle


Metrikker
Prestasjoner
Last opp
Øving 05: Objektstrukturer
Refresh
Øving 01: Objekter og klass...
Øving 02: Innkapsling og v...
Øving 03: Klasser og testing
Øving 04: Objektstrukturer ...
Øving 06: Grensesnitt
Øving 07: Filbehandling me...
Øving 08: Observatør-Obse...

Achievements!

If ekspert

Skriv mer enn 30 IF statements

REVEALED
Totalt: 12

500 linjer med kode

Du har skrevet over 500 linjer med kode

REVEALED
Totalt: 48

Appendix E

System for gamification of lab work in TDT4100

> Using measures to give indications of qualities in student exercises

Project structure

```
e2e         --> Angular end to end testing
libs        --> External jars
postman     --> postman project backup
src
  app         --> Angular
  assets      --> Angular
  environment --> Angular
  main        --> Java Spring backend
  test        --> Java Spring tests
```

Installation

Prerequisites

- node and npm
- angular CLI: 6.2.1
- Java 1.8
- MongoDB 3.6.3

Setup

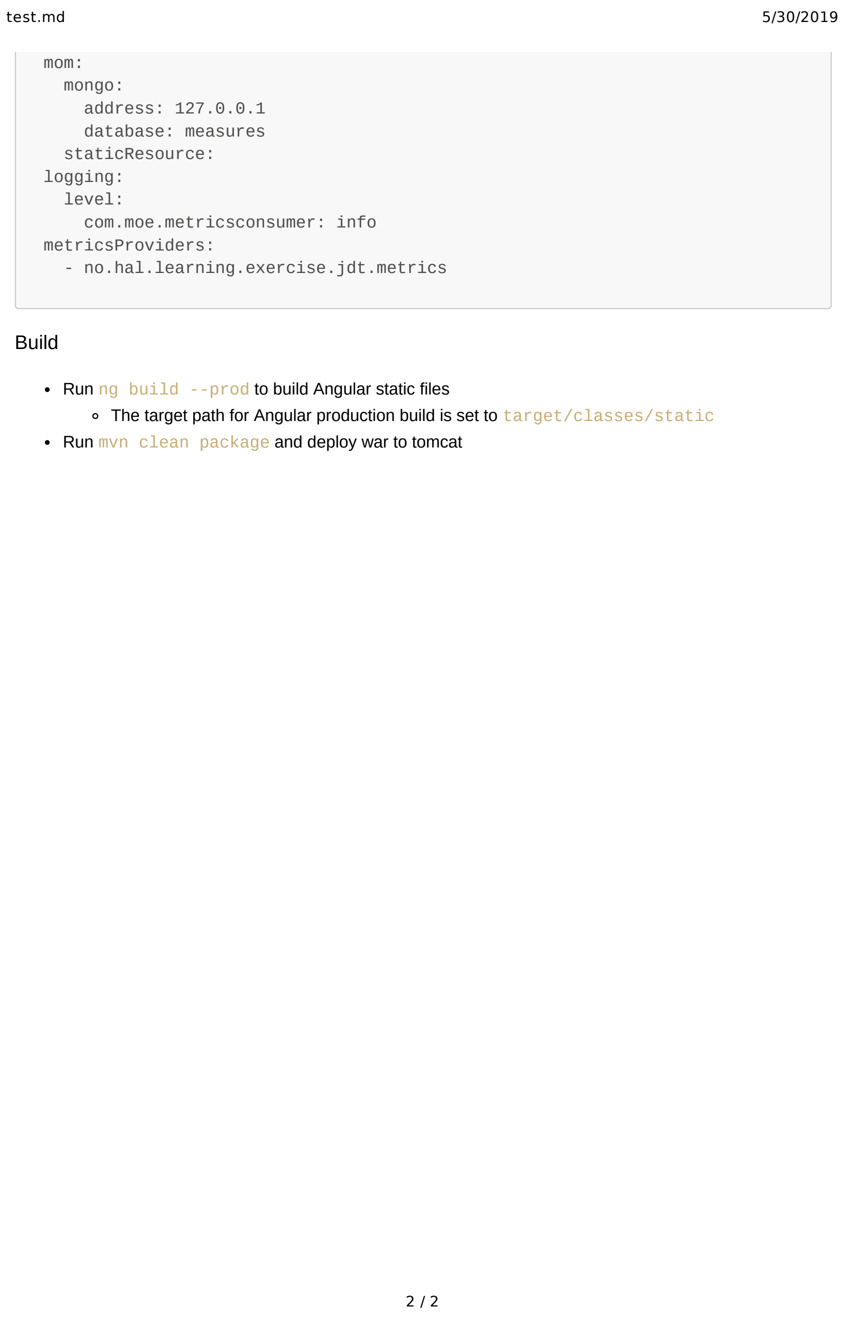
- Clone this repo to your local machine using https://github.com/mariusmoe/metrics-consumer.git
- Go to `libs/` and run commands from `maven-manual-imports`

Authentication provider

> The application uses spring security oauth2 implicit flow.

- Create an account at an authentication provider or set up an Authorization Server.
  - An example is Feide dataporten
- Create the Spring configuration file at `metrics-consumer/src/main/resources/` and fill in the provided fields from the authentication provider:

```
  oauth2:
    client:
      accessTokenUri:
      userAuthorizationUri:
      clientId:
      clientSecret:
    resource:
      userInfoUri:
```



test.md 5/30/2019

```
mom:
  mongo:
    address: 127.0.0.1
    database: measures
  staticResource:
logging:
  level:
    com.moe.metricsconsumer: info
metricsProviders:
  - no.hal.learning.exercise.jdt.metrics
```

Build

- Run `ng build --prod` to build Angular static files
    - The target path for Angular production build is set to `target/classes/static`
- Run `mvn clean package` and deploy war to tomcat

2 / 2