Master's thesis

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

Jørgen Helgå Stamnes

Gitpod, the new standard in programming courses?

A qualitative design and creation study of opportunities and pifalls of using Gitpod for mandatory programming assignments.

Master's thesis in Natural Science with Teacher Education

Supervisor: Hallvard Trætteberg

June 2020

NTNU
Norwegian University of Science and Technology

Jørgen Helgå Stamnes

Gitpod, the new standard in programming courses?

A qualitative design and creation study of opportunities and pifalls of using Gitpod for mandatory programming assignments.

Master's thesis in Natural Science with Teacher Education
Supervisor: Hallvard Trætteberg
June 2020

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

NTNU
Norwegian University of
Science and Technology

Preface and Acknowledgements

This thesis have been a rough endeavour for me. There has been a lot of events the recent year. I will say this; without the help and support of my family, friends and supervisors, this thesis would not have been finished. I would like to send a big thank you my supervisors, Hallvard Trætteberg and Madeleine Lorås. They have been there for me the whole way. The trust, the caring, the tough love. Without you, this would not have been possible.

I would like to thank my father for giving me the extra push, the free food, free shelter and my own office. My brother for giving me some epic social surroundings, and a shout-out to my boys at Discord, for providing me with tremendous amounts of laughs, mental breaks and social surroundings. All this have been a wonderful antidote to the self-imposed isolation during the corona-virus. It has truly been a wild ride.

Abstract

Gitpod is a new web-based integrated development environment that approaches integrated development environments from a new angle. Giptod focuses on short-lived programming environments that automate builds, setups and installation.

With the recognition for Gitpods potential, this study aimed at answering the following question "What are the opportunities and pitfalls when introducing a cloud IDE (Gitpod) in TDT4100 for mandatory assignments?", where TDT4100 is a course on object-oriented programming at the Norwegian University of Science and Technology. Furthermore, to answer the question, a set of subquestions was proposed to learn more about the opportunities and pitfalls when comparing Gitpod to Eclipse (IDE used within TDT4100), and Gitpods pedagogical implications.

The study was conducted using a qualitative design and research approach. The research method consisted of semi-structured interviews paired with observations of students using Gitpod with mandatory assignments. This study found that Eclipses package system and autogenerating mechanisms might interfere with students understanding of folder structures and Java structures on a deeper level. Accordingly, through the analysis of data, there are three components to a programming course: the technological, the structural and students relationship between them. Subsequently, Gitpod might better adhere to socio-cultural and collaborative learning environments. The study found that students have positive attitudes, recognises use-cases for Gitpod and quickly adapt to the environment. However, individual exploration may be challenging.

Keywords: Gitpod, Eclipse, pedagogy, integrated development environment, cloud, programming

Sammendrag

Gitpod er et nytt utviklingsmiljø i skyen som som legger vekt på korte og kastbare miljø som automatiserer bygging, oppsett og installasjon av program og kode.

Med oppdagelsen av mulighetene med Gitpod, prøver denne studien å svare på følgende spørsmål: "Hva er mulighetene og fallgruvene ved å introdusere en skybasert IDE (Gitpod) for obligatoriske øvinger i faget TDT410?", hvor TDT4100 er et emne om objektorientert programmering ved Norges teknisk-naturvitenskapelige universitet. Studien svarer på dette spørsmålet ved å finne muligheter og fallgruver ved å sammenligne Gitpod med Eclipse (mijøet brukt i faget TDT4100) og mulige pedagogiske implikasjoner.

Studien ble utført med en kvalitativ design and creation metode. Forskningen besto av semistrukturerte intervju med observasjoner av studenter som programmerte i Gitpod med tilhørende øvinger. Studien viser at pakkesystemet og autogenereringsmekanismer i Eclipse forstyrrer, hvor studenter viser tendenser til dårlig forståelse og bruker generelle javastrukturer feil. Ved analyse av data, har det blitt gjenkjent at programmeringskurs gjerne består av tre sammenflettede kompontenter; Det tekonologiske, kursstruktur og undervisning og studentenes forhold til disse. Det viser seg at Gitpod mulig følger en god sosiokulturell og samhandlingsnær tilnærming. Studien viser at studentene har gode holdninger, gjenkjenner bruksområder og tilpasser seg raskt Gitpods utviklingsmiljø. Derimot kan individuelle ferdigheter være en begrensning ved bruk av Gitpod utenfor kurset.

Nøkkelord: Gitpod, Eclipse, pedagogikk, utviklingsmiljø, skyteknologi, programmering

List of Tables

List of Figures

Glossary

API Application programming interface . 60

CAQDAS Computer-assisted qualitative data analysis software . 39

CPU Central processing unit . 15

CSCL Computer-supported collaborative learning . 22, 24

GPU Graphics processing unit . 15

GUI Graphical user interface . 9, 29, 30, 56

ICT Information and communications technology . 3

IDE Integrated development environment. 3, 8–10, 12, 16, 17, 22, 23, 51, 57, 58, 66

JDK Java Development Kit . 24, 57, 61

MKO More knowledgeable other . 20, 60

RAM Random-access memory . 15

URL Uniform resource locator . 8, 13

ZPD Zone of proximal development . 20

Contents

1 Introduction

In 2012, the European Commission adopted a strategy for *"Unleashing the Potential of Cloud Computing in Europe"*. The strategy, which had its aims to create 2.5 million new jobs, and increase the total GDP by 160 billion by 2020 had not included the education strategy needed. Bosse et al. (2016) says that there is a need to focus on the didactic processes in information and communications technology (ICT). Teaching and learning should follow the paradigm of internet connectivity; always connected, always available. The "always connected" philosophy of material availability is enhancing discovery. Bosse et al. (2016) says that collaborating through cloud systems are omnipresent through the social spaces, and should be equally ubiquitous in learning situations.

With the rapid change and emergence of new computing paradigms and technologies, educational institutions are continuously challenged. This rate of change is also applicable for cloud technology; the realization that hardware suddenly can be provided as software services is key (Campbell, 2016). Cloud computing let us increase our flexibility, speed, cost-effectiveness and efficiency. Teachers and educators should introduce and teach these concepts to create students that effectively use and explore the capabilities of cloud computing (Campbell, 2016). Sommerville (2013) argues that every single student is in fact, cloud users, and therefore the topic of cloud computing should be incorporated in every course, even if students do not explicitly understand it.

Through discussions with associate professor Hallvard Trætteberg during May 2019, I was introduced to the relatively new paradigm of integrated development environments in internet-browsers. Accordingly, through further supervision, Hallvard proposed an inquiry of potential educational purposes and possibilities with a new web-based IDE called Gitpod.

1.1 Background and motivation

The course TDT4100 - object-oriented programming at the Norwegian University of Science and Technology have a past history of being looked upon as a grandiose course, that is both difficult and frustrating for students. For the time being, a significant role in TDT4100 lifespan

has been the Eclipse IDE, a development tool for programming. The difficulty of the curricula, in combination with a tool aimed at professional developers, have made TDT4100 a rather complex learning environment. Complexity has created some overhead with Eclipse; time is used on non-related curricular activities. Accordingly, the way that TDT4100 is structured, a closed and highly connected Eclipse IDE, have created a situation where learning activities include unwanted non-curricular attention and flexibility limitations.

From my preliminary study of Gitpod with regards to TDT4100 and Eclipse, I found that Gitpod may serve as a good technological choice. Gitpods uniqueness creates an interesting dynamic, where we can think of other ways to structure and teach programming. There is a distinction between these two environments: Gitpod is an IDE that strictly runs in the browser, while Eclipse is a software tool installed on a personal computer. To search for possible answers regarding the potential for Gitpod as an educational component in TDT4100, I wanted to ask the following questions

- RQ.1: What are the opportunities and pitfalls when introducing a cloud IDE (Gitpod) in TDT4100 for mandatory assignments?
  - RQ.1.1: What are the opportunities we gain from Gitpod compared to Eclipse?
  - RQ.1.2: What are the pedagogical implications and opportunities of Gitpods functionality, by introducing it in mandatory assignments?

1.2 Research approach

The mandatory assignments in TDT4100 are tightly configured with Eclipse. To be able to transfer assignments from the Eclipse environment to a strict cloud environment, some time must be devoted to development. These assignments that are introduced in a cloud environment of Gitpod need to be tested for evaluation and gain in knowledge. How such a test should be performed, is a question of time, resources and objectives. Pedagogical implications induce an exploration phase, where literature and previous work are interesting — combining literature with data form a possibility to find cues that can

be analysed with theory. To tackle these broad assumptions, I have based my research on the model of *Design and Creation* (Oates, 2006).

The most fundamental grounding for choosing Design and Creation lies in the description. The design and creation strategy revolves around the development and research of new IT products called *artefacts*. A single artefact that is to be researched can have a plethora of different goals tied to it. These might be an *instantiation*: a process proving that a working system demonstrates constructs, ideas, methods, genres or theories. Other types of examples might be that an application is used in a completely new domain. The approach that applies to this project is the following:

> *An IT application that is a vehicle for a possible gain in knowledge where that application is used in a real-life context* (Oates, 2006).

Our vehicle for gain in knowledge elsewhere respectively becomes introducing Gitpod, and the real-life context is using Gitpod for mandatory assignments in TDT4100 that substantiates its learning outcomes. Furthermore, the context in which Gitpod will be used imposes a development process; we need to transfer mandatory assignments. An artefact has to be developed, in order for that artefact to be tested. Pedagogical implications often have qualitative designs. Design and creation have an array of combinations for data gathering. Allowance for a wide array of data gathering methods creates an opportunity to use qualitative methods.

From the standpoint of design and creation, Oates (2006) suggest that one follow a set of procedures or strategies that will serve as a guideline for how one should research by using an artefact in a real-life setting. The method of conducting research with the strategy of *Design and Creation* resolves around a five-step approach. These steps are as follows: *Awareness*, *Suggestion*, *Development*, *Evaluation* and *Conclusion*.

In Section 2, *awareness* and *suggestion* are presented and discussed to clarify problems and solutions for TDT4100 and the Gitpod environment. In section 3, theoretical frameworks and related work are presented for further discussions on the topic of pedagogical implications as part of *awareness*. The development process of assignments

for the Gitpod environment are clarified in Section 4, and a refined description of research methodology follow in Section 5, both part of a *development* focus. Results and *evaluation* are presented in Section 6, while the discussion follows an overlap between *evaluation* and *conclusion*. Lastly, in Section 8, conclusion and a summary of the inquiry is made.

2 Awareness and Suggestion

The following Section will describe the awareness and suggestion pattern aforementioned in the introduction. Awareness relates to recognition and exploration of new technology, literature, problems or practitioners that express needs (Oates, 2006). *Suggestion* resides in creating a tentative idea formed from the awareness phase.

This Section will begin with an introduction to the course and its corresponding learning objectives and how learning objectives are realized through Eclipse. Subsequently, a brief and concise discussion around the problem areas for the current Eclipse setup will be presented. Lastly follows a description of the Gitpod environment, with a discussion for how Gitpod might improve these problems.

2.1 TDT4100; Object-oriented programming

TDT4100 is a course that is held at the Norwegian University of Science and Technology. The aim of the course is to give students an introduction to the paradigms and practical skill in object-oriented programming with Java. The course is the predecessor from the introductory course TDT4110 - introduction to Information Technology; procedural programming in either Python or Matlab. Students enrolling for TDT4100 are primarily first year students on their second semester. Subsequently, students that enroll for TDT4100 should end up with the following set of knowledge, skills and expertise.

1. **Knowledge**: Students will have knowledge of the most important concepts and mechanisms of object-oriented languages and how object-oriented programs and simple apps are structured and tested

2. **Skills**: Students will gain skills in object-oriented programming, relevant programming methods (coding, testing and debugging) and modern development tools

3. **Expertise**: Students should be able to use object-oriented programming to solve practical problems and explore the opportunities of modern development tools.

The focus that is most important for the inquiry, is the emphasize on objectives related to the use of modern development tools. TDT4100 realizes modern development tool usage through the professional IDE, Eclipse[1]. To clarify: Students should solve practical problems, explore and use relevant programming methods to gain skill in object-oriented programming with Eclipse.

2.1.1 Course structure

To support the goals for knowledge, skill and expertise facilitated through the Eclipse IDE, a set of structures are made to distribute and give students activities and support mechanisms that mediates learning. Accordingly there are four main learning activities: lectures, assignments, lab-hours and assignment-walk-through-lectures. Furthermore, the course has its main focus on assignments, where all the other activities are support mechanisms to help, teach and prepare them. Completion of assignments are required for becoming eligible for the exam.

2.1.2 Assignments and tasks

For the remainder of the thesis, the following terminology will be used in order to describe the mandatory assignments in TDT4100. Figure 1 outlines the terminology for the assignment task relationship. Each week there is a new *assignment*, each assignment has n tasks (normally in the range of 3-8 tasks per assignment), where a set of m tasks must be finished in order to complete one assignment. More details about the first assignment for TDT4100 can be found in Appendix D.1.

The assignments are given on a weekly basis, with a total of 12 assignments over a 12 week period. Each week the assignments can be on one of the formats given in table 1. The tasks are either sequential, non-sequential, application specific or has a testing or debugging focus.

Each week the students participating in TDT4100 must deliver their asisgnments online by compressing their files that relates to the

[1]Eclipse. A professional development tool developed by the Eclipse foundation. More information can be found here: `https://www.eclipse.org/org/foundation/`

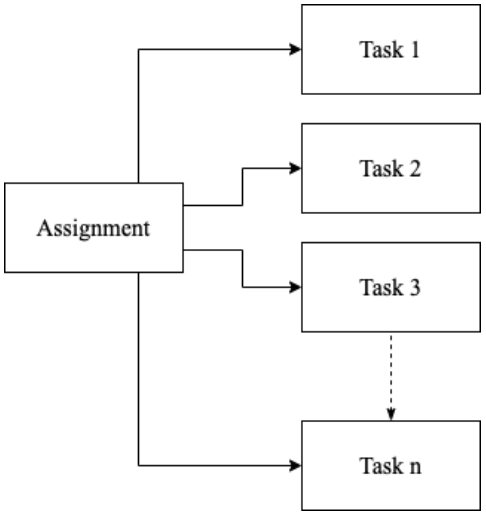


Figure 1: Assignment and task structure

current assignment. With file compression and delivery, they also have to show and explain their solution to a student assistant during lab hours. The assistant will give them a score from 0 to 100. The students will be eligible for the exam if the score exceeds 750 points during the assignment period. However, student assistants are also used for help and tutoring.

2.1.3 Eclipse and problems

As mentioned, there are some practical problems with Eclipse for TDT4100. First, a brief introduction for what an IDE is, and how Eclipse is structured for TDT4100, will be presented. Secondly, the problem of a highly integrated Eclipse will follow given the above.

Eclipse

Eclipse is an integrated development environment developed by the Eclipse Foundation. An IDE stand as a software tool that encompasses common developer functionalities combined into a graphical user interface (GUI). In Figure 2, Eclipse's GUI is shown. An IDE has typically a text-editor for changing and writing files, with the assistance of syntax highlighting capabilities, language specific auto-completion, and

live bug-detection for possible compile or build issues. An IDE usually abbreviate processes for compiling and running programs, and debugging options to graphically display and locate bugs (Redhat, 2009). Accordingly, an IDE is a tool that has features that help with productivity, structure and the creation of programs and applications. Correspondingly, Eclipse has all the features mentioned above.

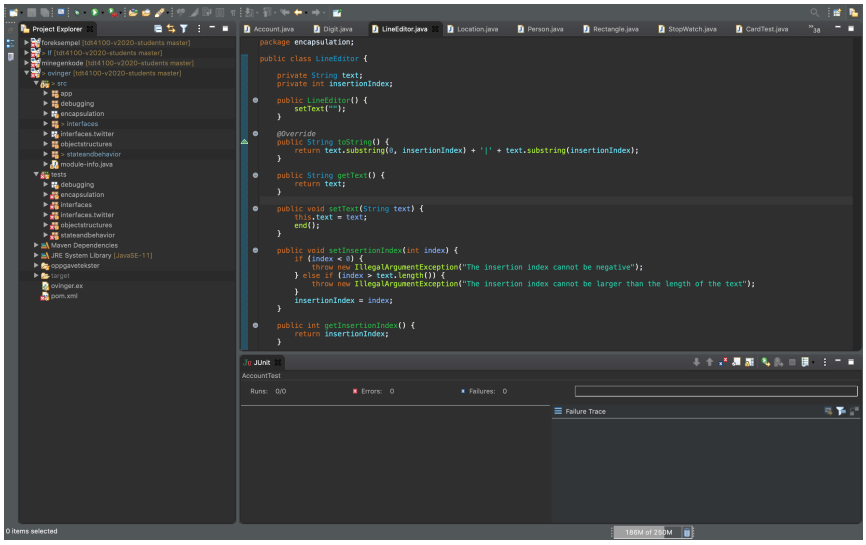


Figure 2: Eclipse integrated development environment. On the left, the file explorer is open, with its correspond tree of packages. The big top-right square is an open text editor, showing code for the java-file LineEditor. At the bottom the JUnit testing framework is shown.

Problems

Interestingly, the root of the problems associated with Eclipse comes down to the fact that all of the learning objectives are facilitated through a highly structured and specific configuration. Furthermore, students are also required to complete assignments for exam eligibility. As a result, there are setup-procedures required for Eclipse to be functioning properly, that relies heavily upon the students themselves. Consecutively, those properties need to be maintained throughout the course, hence there are still non-curricular related activities present. Problems that are recognized are:

1. **Initial setup procedures**: The Eclipse environment for TDT4100 is reliant on repositories containing specific configuration for setup and distribution of assignments. Configuration has created a complex installment procedure that is necessary for getting Eclipse ready for programming. These procedures has created overhead and halts progression from the get-go.

2. **Maintenance**: Students are responsible for maintaining the correct state of the source-code in Eclipse with version control management through git[2]. Students do not get presented with Git in their study program paths before entering TDT4100. As a result, they now becomes the maintainers. Conflicts between source trees occur.

3. **Accountability**: Students are held accountable for their setups and maintenance without the necessary tool-set required to solve particular niche problems that occur.

4. **Resources**: Resources are required to support students during their programming endeavour with obligatory assignments.

With these problem-areas, work has been put in to elevate and potentially find alternative solutions for distributing assignments in a more reliable and accountable manner. Hence, In the next subsection, Gitpod will be presented.

2.2 Gitpod

Browser-based integrated development environments have been around for some years, and Gitpod is one of those environments. In this section I will briefly describe Gitpod and its intrinsic functionality and eco-system. In addition to system functionality, a discussion around opportunities for TDT4100 and its structure will follow. Lastly, a *suggestion* for further directional matters will be presented for *development* purposes.

[2]Git: A version control system for updating, changing and tracking code. Industry standard.

2.2.1 Eclipse Theia

Eclipse Theia is the underlying IDE that Gitpod exposes. It is maintained as an open-source project by the TypeFox foundation (TypeFox, 2019). Theia exposes an extensible developable environment that allow users to create custom browser and desktop IDEs. Theia is presented in Figure 3. In short, Theia contains all the previously mentioned IDE features that was presented by the description given in Subsection 2.1.3.

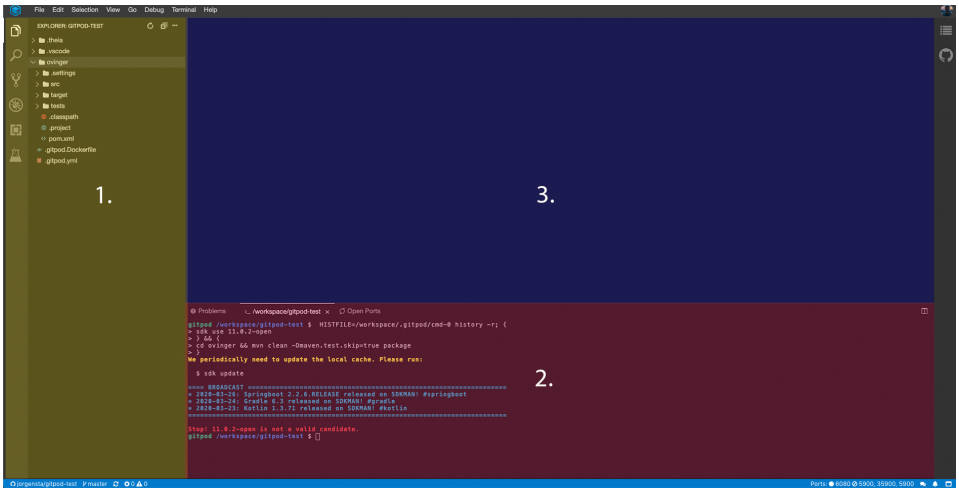


Figure 3: Eclipse Theia IDE. 1. File-explorer 2. Text-editor. 3. Terminal window

2.2.2 Gitpod eco-system

The Gitpod eco-system contains four integrated components: Docker[3], Kubernetes[4], Theia IDE and version control distributors that use Git (Github, Gitlab, BitBucket). Particularly, the Docker (container) component of Gitpod is valuable to understand.

[3]Containerization software. More information about docker can be found here: `www.docker.com`

[4]Kubernetes, container orchestrator tool by Google. More information about k8s can be found here: `www.kubernetes.io`

> *A container is a standard unit of software that packages up code and all its dependencies so the application runs quickly and reliably from one computing environment to another. A Docker container image is a lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries and settings (Docker, n.d.).*

In other words, Theia is an application that is a packaged software container that runs reliably on different environments, and is accessible through web-browsers. Correspondingly, the relationship with version control distributors and Gitpod are important. You can only start your environment from a repository location, copying the files within that repository, and launch them in a Gitpod instance. Figure 4 illustrates this component of working with version control distributors, and Figure 3 show us the result.

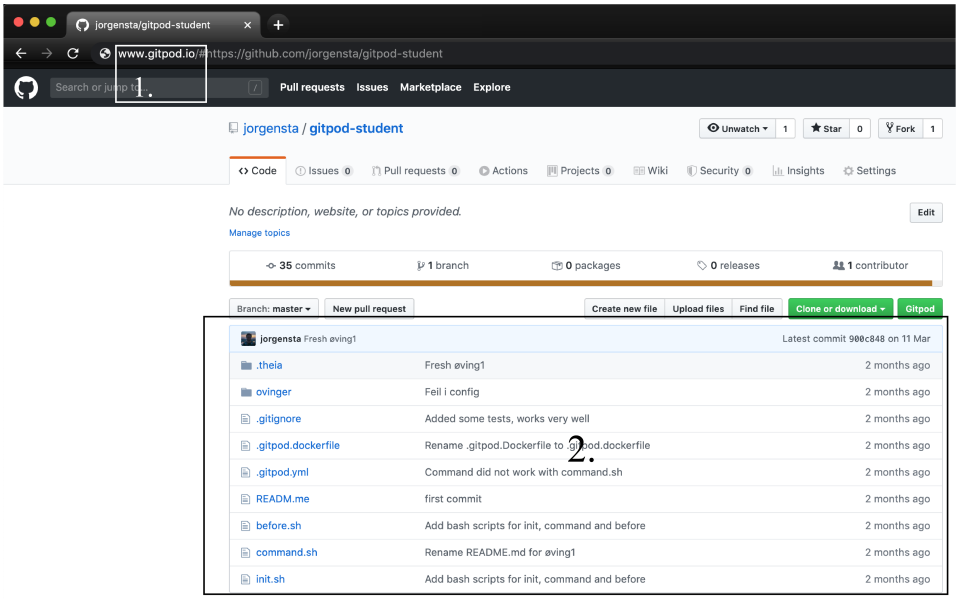


Figure 4: Launching a workspace from a the version control distributor Github. 1. a pre-fix to the Github URL is needed to copy files stored in block 2.

2.2.3 Features

Containerization of applications, and how it relates to Gitpod, creates some interesting features that exceeds Eclipse functionality. Moreover, the features are the characteristics that will furnish a discussion for possible solutions.

Workspaces

A common term in Gitpod is *workspaces*. A workspace is a single container launched in the Gitpod eco-system. We can view this workspace as the terminology suggest, namely a workspace, a place for working (with programming). The previous description of using version control distributors repositories for launching Gitpod is the act of creating a *workspace*. In Figure 3 a Gitpod workspace is running in the browser. The workspace contain all the source code. The files are eligible for execution and are editable.

Declarative control

Gitpod let us describe how the system should behave upon launch. Declarative configuration through YAML[5] definitions and bash[6] scripts ensure that all future workspaces that are launched from a configured repository, are behaving correctly and contains the desired state.

Sharing and collaboration

An Individual *workspace* can be shared. Two sharing features are prominent.

1. **Snapshot** A snapshot is a deep copy of a workspace. It contains all the container-configuration and files of the shared workspace. A complete replication and copy.

2. **Workspace sharing** Shares and opens up the current workspace. Two or more contributors can work on the same file system.

Snapshot and workspace sharing allows users to replicate their current working state. The distinction between the two modes of sharing is illustrated in Figure 5.

[5]A readable serialization language commonly used for configuration files
[6]Shell and command language used for UNIX

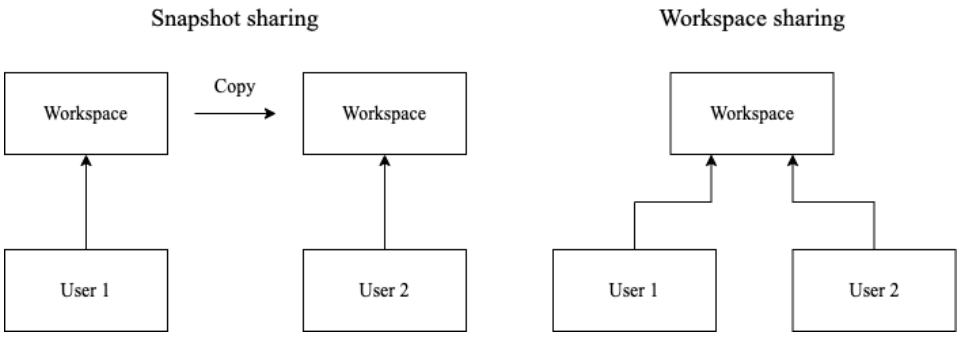


Figure 5: Differentiation between snapshot and workspace sharing

Hardware independence

While Eclipse is a standalone desktop application that requires instalment on a local computer, Gitpod is a software as a service (SaaS) platform, that consecutively requires an Internet connection and a web-browser for software access. Understanding the difference between SaaS and normal desktop applications becomes crucial.

Meanwhile, desktop applications use hardware, such as CPU, GPU, and RAM on a personal computer, Gitpod outsources hardware. A *workspace* do not explicitly execute on a local personal computer, but rather, workspaces are software that runs on hardware provided by cloud-services that are accessible through web-browsers.

2.3 Opportunities

Given the current state of TDT4100, new technology might bring fruition to more interesting qualities within TDT4100. With Gitpod, we gain immediate access to a programming environment with workspaces. No installation required. Furthermore, course administrators can gain declarative control over the distribution, creating more reliable environments for coding, and users (students) are withdrawn from their hardware on laptops. Subsequently, environments are reproduced and replicated through a configured repository. Workspaces are shareable units of pre-configured and ready-to-code environments.

Awareness of Gitpods internal structures gives an overview of potential opportunities. However, the assignments are currently not compatible with Gitpod, hence using Gitpod for mandatory assignments

becomes a process in which some development is needed.

2.4 Summary

TDT4100 is a course that focuses on object-oriented programming through the IDE, Eclipse. Problems with the current distribution of assignments in TDT4100 are initial startup procedures, maintenance of source-code state, students accountability and resource expenditure. Consequently, Gitpod exposes a different alternative to traditional desktop IDEs, with reusable workspaces, hardware Independence and declarative control of behaviour and state. Opportunities with distributing assignments with Gitpod are removal of installation, accessible coding environments and alternatives to resource allocation.

3 Theory and related work

In the world of pedagogy, there are different paradigms that view the world in specific manner, and IDEs are something that programmers do not live without. To find interesting research on student experience and pedagogical implications of IDEs, I will examine methodologies, theories, concepts that are relevant for my research inquiry. First, the pedagogical branch of socio-cultural theory will be presented. After that, mental constructs and the *notional machine* will be described. And lastly, related work from pedagogy and IDEs.

3.1 Socio-cultural learning theory

The socio-cultural learning theory finds its roots from the Russian psychologist Lev Vygotsky. The theory itself has had a cultural development driven by the Europeans and Americans heavily inspired by Vygotsky himself. The main characteristics stem from the early development of children, their culture, which determines how and what a child learns about the world (Skaalvik & Skaalvik, 2013).

The leading cognitive association with Vygotsky is his thoughts on the zone of proximal development (ZPD). The zone of proximal development is categorized as the zone in which the student/pupil with adequate support and guidance from his tutors, may accomplish a goal or a set of tasks, in which the pupil would not accomplish without the guidance.

As for the base for the theory, and presented in Figure 6, Vygotsky proposes three zones from the expected mental and cognitive capabilities for a student.

1. What a student can achieve independently
2. What a student can achieve with guidance and tutoring
3. What a student cannot do on their own with either peer support or tutor guidance

In a world where a child learns about his environment, the key to success and a simplified interpretation of ZPD is the following "The things a student can do with assistance, are the things that are possible tomorrow completely alone" (Skaalvik & Skaalvik, 2013). The key

Figure 6: The zone of proximal development

takeaway is that the curriculum that is taught for a given individual should strive to live in the ZPD.

When it comes to the set of tasks, curriculum and difficulty presented by a given course, ZPD can be an alternative way of describing adaptive education. This style of teaching demands great efforts of differentiation and a highly individualized study of each students' ZPD.

3.1.1 Guidance and support

The idea of an individual having a ZPD accentuates a teaching style that consists of various strategies to give the circular area of development the chance to flourish. Since the instruction is tightly related to the students' proximal development zone, it means that each student needs some guidance and support, hence "Scaffolding" (Skaalvik & Skaalvik, 2013) is used as a metaphorical term.

Guidance and support build the foundation of scaffolding. The takeaway, however, is that scaffolding is expected to be taught in a specific manner; a guide in which you only give hints and tips so that a student can find out the answer on their own, not by mere telling or showing them what they are supposed to do; you are guiding, not showing.

3.1.2 Scaffolding

Instructional scaffolding differs from other types of instruction support in which the goal of what students are intended to get out of it,

the timing of the support and what kind of support. The first thing we must look at is that scaffolding needs to make the student compatible enough to support the current performance, where the main goal is to make the student able to complete a skill independently. A calculator does not qualify as a scaffolding tool in which it aids the student. It may aid their calculation skills in the "moment", but it does not account for independent calculation in the future.

The second thing is when students are engaged with ill-structured / authentic problems in which there are more than just one solution to the problem. Instructional approaches such as lecturing models, strategy and examples do not qualify as scaffolding. It does not build off the current knowledge, it is just pure representational.

Third, scaffolding needs to build upon what the students already know and be tightly relational to the assessment of the given knowledge domain. Show and tell (traditional instruction) do not qualify as scaffolding; it does not support itself on what the students already know (Belland, 2017).

The historical definition of "scaffolding" was initially proposed as a metaphor to describe how parents and teachers provide support for their children as they learned to build pyramids of wooden blocks (Belland, 2017). This scaffolding support was meant to instruct their children in building pyramids, while they were doing the bulk of the work to solve the pyramid problem. Scaffoldings' role was to fill the gaps in students knowledge and abilities to such extent that they could complete the task at hand. The main goal was to support children in their engagement with problems, to lead the development of skills to become and be an integrated part of their problem-solving independence.

Scaffolding was contingent, that means it took on two key events; It was iterative and interconnected. These two key events assess the scaffoldees' current performance characteristics and also the provisioning of just the right amount of support. Contingency means that scaffolding as a whole relies upon a dynamic assessment approach.

Scaffolding also required a shared goal between the scaffolder and the scaffoldee. Shared goals was considered necessary to let the scaffoldee know what a successful task would look like and how to obtain it. This specifies and points out a crucial part of the Independence aspect of scaffolding (Belland, 2017).

3.1.3 Socio-cultural programming

While Belland talks about the building blocks of scaffolding, children and students place in the hierarchy; there are some principles that encourage programming pedagogy. The mediation of teaching with scaffolding ensures that students can complete tasks individually, rely heavily upon the notion of having roles of a "More knowledgeable Other" MKO (Sentance et al., 2019). The idea of a MKO relates to teachers and peers, where concepts and models are mediated to a less knowledgeable student from a MKO. While a construct of pairing knowledge aligns with a socio-cultural stance, the mediation through language is prominent. Teaching should focus on facilitation of collaboration and discussion (Sentance et al., 2019), while programming tasks should be carefully scaffolded over time, gradually more complex, incorporating elements that focus on the ZPD.

3.2 Mental constructs

Why students struggle to understand computing concepts, have been studied by computer science education researchers for a decent amount of time. The concept of a "notional machine"; An abstract model of the computer may have been concluded as the missing building block. (Bower & Falkner, 2015).

A notional machine can be categorized as a characterization of the role of the computer of executing programs in a particular language (Sorva, 2013). Notional machines are the abstractions we create, to connect different parts of hardware and software that execute programs. While there are different kinds of programming languages, a single notional machine mostly exists for each language. Java might have a different notional machine than say, Python. Notional machines that describe object-oriented behaviour are different from procedural, that is different from mathematical machines. A higher-level abstraction of notional machines in regards to Java might be the mental model of a computer keeping track of objects running through the program, passing messages and interacting with each other (Sorva, 2013).

Learning to program is not easy. There are five areas of difficulty connected to the task of programming. The general problem of *orientation* discusses the problem of finding out the question of what programming is for, what problems programming tackles and presumed

advantages of learning such a skill (Du Boulay, 1986). Secondly, there is the difficulty of understanding the general properties, of the machine that is programmed, in regards to the mental model of a *notional machine*. The third difficulty exemplifies the formal language, learning semantics and syntax. Understanding standard structures to achieve goals with sums, loops and conditions are the fourth condition. The last and final difficulty resides in the aspect of programming pragmatics. How does someone test, debug and specify applications with available tools? These five difficulties are not entirely separable; they are often intertwined into a shockingly complex system that students attempt to deal with simultaneously (Du Boulay, 1986).

> *"Learning a programming language involves not only learning that language, but also the language for managing programs as well as the language for editing programs"* (Du Boulay, 1986)

A computer program with its programming system is also a mechanism, a mechanism that creates other mechanisms. Even so, learners will often interpret these systems with their own mental models, rather than relying on help. These mental models can be impoverished and insufficient to explain program behaviour (Du Boulay, 1986). A learner needs a rather simple explanation of system ingredients.

3.3 Web-based IDEs - *What is out there?*

As cloud computing has emerged in the last decade, desktop applications migrate to the cloud and web. Applications such as Slack, Google Sheets and Docs, Microsofts office package and normal file storage have become commonplace. The success of application migration to web-based systems is also challenging the status quo for integrated development environments. Often, these integrated development environments are trying to offer functionality beyond the original desktop applications. Tran et al. (2013) have a long-term strategy to develop their IDEOL application for fixing the challenges of collaboration through ICT courses. Collabode (Goldman et al., 2011) tries to support synchronous programming through a web-based IDE, CoRED (Lautamäki et al., 2012), IDE 2.0 (Itahriouan et al.) and Adinda

(van Deursen et al., 2010) resolves around the challenges of collaboration. Gaikwad et al. (2014) tries to resolve a presentation problem by having all the environment installed on their own computer during presentations. Dutta et al. (2014) are discussing how to implement a multilingual web-based IDE.

Almost all the former IDEs have some property of collaborative learning. Collaborative learning is the property of peer interaction and serves as one of the most important factors in learning (Dillenbourg et al., 2009). The evolution of computer-supported collaborative learning (CSCL) can be divided into three eras. 1: CSCL emerges after a neglect for such properties. The neglect resulted in the understanding of co-constructed knowledge and productive social interactions. The second area constitutes of a scientific community growth of social interactions through activities, environments and utilization. The third era concludes a disappearance for CSCL as an individual pedagogical framework, where CSCL are becoming more and more integrated within activities, both physical and virtual (Dillenbourg et al., 2009).

Group learning and activities are becoming the norm, that culture without such interactions would be unthinkable. The scene of CSCL proves an emergence of both learning theories and technological evolution, where an illusion of converging towards a socio-cultural pedagogical perspective and individual constructs are debatable. Some think that a CSCL environment is strictly for the individual, while others argue that knowledge is co-constructed through a pure socio-cultural lens of social interaction and thinking (Dillenbourg et al., 2009).

The broad definition of "collaborative learning" regardless of computers, is the activity where two or more persons actively attempt to learn something "together" (Dillenbourg, 1999). The scales that induce the concept of "collaborative learning" are many. From a single group of 3-5 people collaborating, or the label of CSCL where around 40 people participate in a course. While the context of collaborative learning seems rather broad, there are a variety of meanings for both keywords of "learning" and "collaboration".

3.4 Pedagogical IDEs

Teaching object-oriented programming in universities has become more and more common (Kölling et al., 2003). This is most likely ubiquitous in today's computer age and time. Although there seems to be a lack of clear direction in ICT and pedagogy, especially for object-oriented languages, where Kölling et al. (2003) profoundly suggest that object-oriented programming inherently find itself at a complexity level of procedural programming. Kölling et al. (2003) commonly reveals three common fundamental problems with existing integrated development environments: the environment is not object-oriented, the environment is too complicated, and the development environment focuses on the graphical user interface.

Object-oriented environments should reflect the paradigm that you are programming in, where students should work with abstractions that has a purpose. Object-oriented classes and objects should be worked with directly, instead of files and applications found in most IDEs, that mostly are too complicated (Kölling et al., 2003).

BlueJ, as an educational component in object-oriented teaching, shows us a specific set of pedagogical interventions that supposedly abrupt and change how we teach object-oriented teaching. However, from another perspective, the key to students performance in introductory courses do not depend on the tools used; instead, it is an extended duration for the introductory course. Silva-Maceda et al. (2016) compared pedagogical approaches in introductory programming for the C-language, using a pedagogical tool called RAPTOR in one module, and a standard module without. These modules had two timeframes, a pre-university course paired with regular university courses, and without any pre-course, following a usual university scheme. Both RAPTOR and the standard way of teaching followed the pre-paired and typical scheme. While RAPTOR seemingly outperformed the standard approach, on both lengths of introductory programs (pre and non-pre), the RAPTOR approach did not show any statistical significance when preliminary skills were taken into account (Silva-Maceda et al., 2016). Moreover, preliminary skills are an essential performance indicator; however, more data is required to generalize their findings. Meanwhile, during a study of how novices tackle their first line of code, Vihavainen et al. (2014) argues that their data suggest that beginning with an off-the-shelf IDE do not impose any detrimental effects on stu-

dents. Instead, it is syntactical issues that arise during start phases of programming.

The implications of IDE usage can also be seen as a necessity for students attention and excitement for a particular tool. During a two-year pedagogical experiment, Chen & Marx (2005) developed an approach to teach introductory programming in Java with Eclipse, that encompasses working with command-line JDK first, for several weeks, and then afterwards including IDE usage through Eclipse gradually. Supposedly, this line of an introduction should create a more general sense of the higher aspects of productivity within IDE's and get a general overview of structures within Java. While, given the choice of choosing IDE's, students would pick more comfortable IDE's over Eclipse (Chen & Marx, 2005).

3.5 Summary

The zone of proximal development proposes three zones in which a learner may be located, with a focus on activities that create challenges. Learning is mediated through scaffolding. Guidance and support become essential aspects of realizing the learning potential by creating stimulus. Learning programming is often looked at as a grandiose task. Some believe that a wrong interpretation of a *notional machine* may be the issue, while others argue that it is a complicated endeavour that includes pragmatics, semantics, mental constructs and properties. Numerous IDEs have been developed, with effort in creating collaboration within tools. CSCL gradually has become more integrated into physical and virtual spaces. IDE related pedagogy suggest that there are implications of usage that indicate either using professional IDEs or strict special tooling for pedagogy.

4 Artefact development

In order to test Gitpod in a real-life setting, I had to do some initial development. The goal was to have a functional Gitpod setup that had similar design and structure as the mandatory assignments from Eclipse. This section does not resemble any newfound development strategy, as its sole purpose was to create a working system that could be used for testing.

The development process and the final product description was a collaboration between me, and my supervisor, course teacher and associate professor Hallvard Trætteberg. The goal before the tests was to create a minimal viable product. The product was a pre-configured GitHub repository ready for Gitpod workspaces. The Gitpod workspace was to take on a set of functionality that we already had with our Eclipse installation, creating some sort of familiarity between these two IDEs. The primary goal for the development process was to transfer the existing assignments of our Eclipse setup, onto Gitpod.

4.1 Initial discussions

We discussed and found learning activities where we could eventually use Gitpod. Three learning activities became our focus, and are outline in table 2.

Since time and resources had to be taken into consideration, the development of the artefact had to be of reasonable scope. In Table 1 types of different assignments are presented. Thus, we found that an artefact with Gitpod in regards to mandatory assignments for TDT4100 would suffice. With the more problematic area of application development (Table 1) for some assignments and the complexity of doing such application development with Gitpod, we resided in narrowing the scope even further, excluding applications assignments from the artefact.

The artefact was to include the same structure as Eclipse (folders and Maven setup), the same testing regime (each task has a pre-made test-suite) and keep the sequential nature of tasks intact.

Assignments	
Format	Description
Sequental	New assignments are tightly correlated to preceding assignments. Previous classes (objects) and tasks are expanded with new principles and methodologies
Non-sequential	New assignments sometimes do not rely on tasks from previous assingments; Some tasks cannot be broadened in order to emphasize an object-oriented methodology.
Application development	Assignments can have an application focused theme; creating simple desktop applications to exemplify object-oriented programming in an applicable manner.
Testing and debugging	Each task in an assignment may have test-suites. Knowledge about testing and debugging occurs throughout the entire course. Some assignments emphasizes primarily on testing and/or debugging

Table 1: Assignment formats

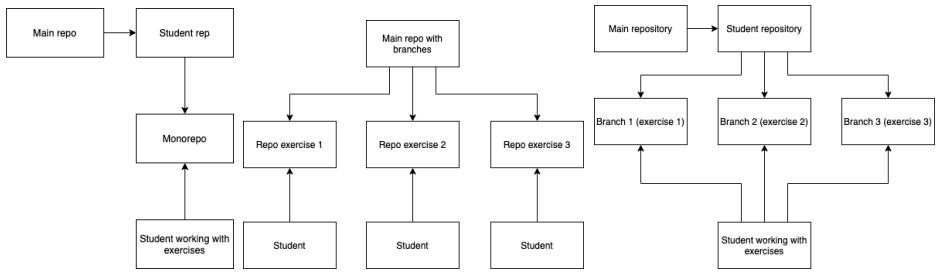
4.2 Repository architecture and sequential exercises

Since Gitpod only initializes from various distributors of repositories, one question arose during development *"How do we structure these repositories?"*. While Eclipse has a direct coupling with the assignment repository, but not explicitly reliant on these repositories, we had to find a solution for how assignments should be distributed.

Different architecture setups were discussed for assignments during the development phase. Three architecture setups where discussed: assignments on branches, one central repository for everything and multiple repositories for each assignment. These structures are illustrated in figure 7.

Learning activities		
Activity	Description	Situation
Assignments	A Gitpod environment with a focus on mandatory assignments should cover sequential, application and debugging/testing assignments.	TDT4100
Lectures	A Gitpod environment with a focus on lectures included more interactive lectures with code-examples and snapshot sharing during lectures.	Lectures in general
Web development	Some courses have web and application development focuses, and thus a new focus on development could be with Gitpod.	Web development courses

Table 2: Main learning activities for a Gitpod environment



(a) Mono repo (b) Multi repo (c) Branch structure

Figure 7: Repository structures

4.3 Challenges

Some challenges where more prominent than others. The central problems are outlined in table 3. The focus for development became ensuring that these issues where solved. The issues became the source

of truth whether the artefact was finished or not. During development daily notes were taken with note keeping in Microsoft OneNote.

Gitpod artefact problems	
Topic	Description
Test suites and feedback	The environment should have easy-to-use tests and good feedback. The Eclipse test dependency did not work on Gitpod, therefore an alternative had to be implemented
Repository architecture	Gitpod has a tight integration with Github and Git. An easy repository architecture must be ensured
runnable code	The code had to be fully built before any code could run. This problem made the Gitpod setup non-congruent with our Eclipse installation
Sequential exercises	We needed a way to ensure that sequential exercises worked properly

Table 3: Backlog items for artefact development

4.4 Artefact design and description

In section 2.2, the Gitpod environment was described. In this Section we continue to expand upon the Gitpod artefact inquiry, and discuss how the obstacles was solved.

4.4.1 Repository architecture

The debate of repository architecture for the mandatory assignments converged on a the mono-repo architecture; keep all the assignments continuously in one single repository. Meanwhile, branch and single-purpose repositories are viable options; they seemed to include

teeming arrays of complexity to the already complex nature of working with Docker container virtualization. Furthermore, keeping track of multiple repositories and branches for each assignment would not impose any significant concerns; rather, GUI tasks had to be written manually in advance to switch between assignments. Additionally, the nature of saving programming progress during a branch and multi-repo structure would potentially create more difficulties for students to store progress. However, one continuous repository would not be free of problems; rather, it stood out as the preferable choice and substantiated the existing assignment scheme.

In appendix Appendix E, I included the configuration for the declarative style of creating Gitpod environments. Here, we can see in E.4 a command bash script. The script runs every time a workspace is launched. This bash script ensures that there is always a connection to the already existing repository for academic staff of the course, that holds the assignment plan. The upstream repository for each workspace is set during initialization. The update process of publishing new assignments (that are sequentially tied together) is one single command from an admin.

4.4.2 Test suites and feedback

Testing is a big part of TDT4100 assignments, and initially, the testing framework for the already existing Eclipse setup did not properly work for Gitpod. Thus, an alternative had to be implemented. Non-working test-suites with JExercise[7], forced tests to be executed through maven commands. In spite of non-working tests with JExercise, another issue arose with the alternative of abstracting tests with GUI task commands. Beneath the abstraction where maven commands. Maven commands could not be run without creating them manually. Alternatively, the Java Test Runner extension[8] created an opportunity to have seemingly manageable auto-runnable tests, and create feedback on the tests through descriptive expansion panels. The test-suite extension is shown in figure 8.

[7] JExercise is a test-suite dependency specifically developed for TDT4100

[8] Java Test Runner extension is downloaded from the extension library for Visual Studio Code. It is compatible with Gitpod environments

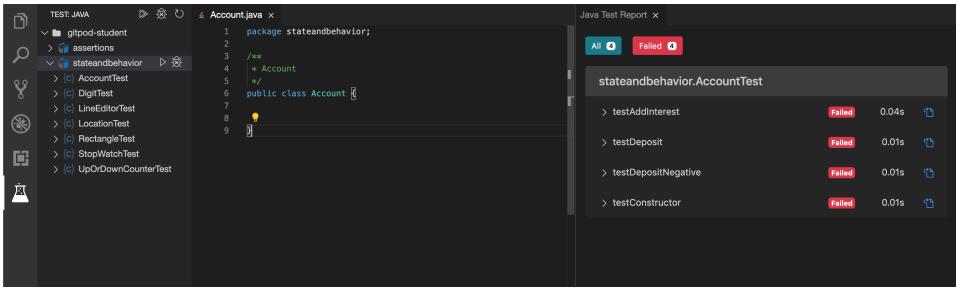


Figure 8: Running a test-suite for Account.java task. Output is the java test runner extension

On the left-hand side of figure 8, the tree view of folder structure given an assignment is shown, while at the far right, the expansion panels are individual tests. In this scenario, every single test failed due to non-implemented methods. The problem of test-suites was fixed with this extension in a reasonable manner; however, the expansion panels were not compatible with Gitpod, and therefore stand as a bug that needs fixing from TypeFox.

4.4.3 Running code

Running code was not an issue, but the building of entire projects was. Due to the nature of Maven, the whole project had to be "buildable" in order for program execution. Accordingly, one specific set of Theia configuration had to be disabled. With the disabled configuration, all future workspaces launched would contain that configuration, hence reproduction of behavior and state.

4.4.4 Sequential exercises

Students progress from assignment to assignment in a sequential manner were executed by command through a GUI task with a single pull from the main repository, due to mono-repo architecture. Configuration task for this can be found in appendix Appendix E.

4.5 Pilot-test

A pilot-test was run and tested on a couple of associates that knew Java from before. Testing the artefact served four purposes.

1. To test that the solution worked
2. To potentially find bugs within the system before the initial tests
3. Guidelines for potential interview questions and tests
4. Putting myself in a comfortable interview position gaining some experience and traction beforehand

The complete configuration and assignment scheme can be found at my github[9]. Discussion from the pilot-test is summarized in Table 4

Gitpod pilot-test	
Topic	Discussion
Experience	They found it easy to use, and was impressed by the simplicity of launching an instance. Subsequently, common IDE features where pleasant. No comments on lagging or sluggishness while programming.
Opportunities and pitfalls	Starting with the course becomes considerably faster. Sometimes, there is not so obvious where certain features where. Saving progress might be problematic. An introduction for Gitpod might be much faster than the current situation. Delivery of assignments were quite easy. A special opportunity for better lectures.
Pedagogy	The availability of an environment might have a considerable effect on learning.

Table 4: Topics discussed during the pilot-test with colleagues

[9]Github repository used for testing assignments `www.github.com/jorgensta/gitpod-student`

5 Research process and methodology

The thesis is trying to answer the question of which opportunities and pitfalls, technically and pedagogically, an object-oriented programming course (TDT4100) would have in introducing Gitpod for mandatory assignments. To understand and further discuss these implications, a robust methodological framework is needed.

Interested in both social science and engineering, I believe that there exists a link between qualitative and quantitative data. Some phenomenons, theories or hypothesis can only be explained through qualitative data, some require quantitative data, and sometimes, we might consider both. In this research section, I will begin by describing my choice of paradigm. After my paradigm description, I will outline the process of my research methodology; *Design and creation* and *the stepwise-deductive induction method*. The last part of my process will describe quality considerations and ethical concerns.

5.1 Pragmatism

How the world should be interpreted has been discussed back to ancient western philosophy, and the debate continues even to this day (Johnson et al., 2007). *Mixed method research* strive towards uniting the extremes of quantitative data (Plato) and qualitative data (the Sophists). *Mixed methods* are striving to find the middle ground between the extremes, respecting the wisdom of both inquiries.

The primary philosophy of *mixed methods* is pragmatism (Johnson et al., 2007). The approach caters to consider multiple viewpoints and not just the consideration of a paradigm being absolute. Pragmatism can be regarded as an alternative paradigm (Yvonne Feilzer, 2010), where the researcher is free from constraints imposed by either postpositivism and constructivism; pragmatism relies on the empirical orientation that imposes real-world problem-solving (Yvonne Feilzer, 2010).

5.2 Qualitative Design and Creation research

An important note in regards to design and creation research is that the process should not just be an illustration of technical prowess, but also should include academic perspectives, such as analysis, argumentation and discussion. Alas, the approach of design and creation

must contribute academically. Oates (2006) proposes three ways an IT-artefact can be academically valid.

1. The artifact is the main focus of the research
2. The artifact is a vehicle for gain in knowledge elsewhere
3. The gain in knowledge is the development process of the artefact

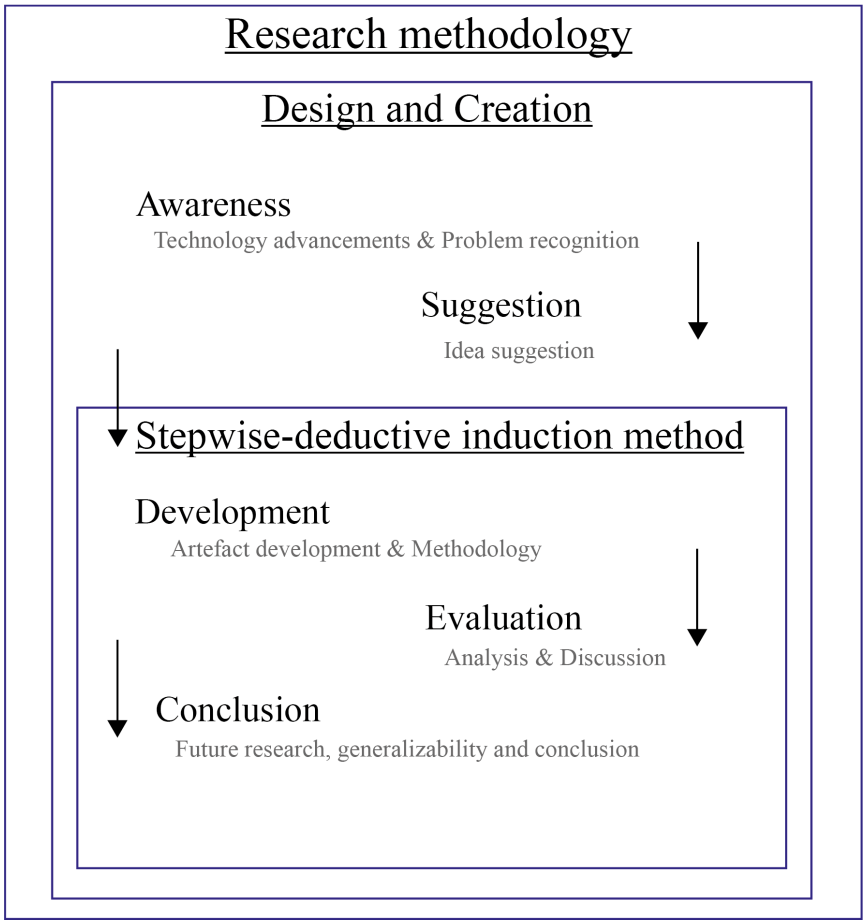


Figure 9: Research methodology

To reiterate on information from the introduction, The approach used for this research, is with the artefact being a *vehicle for gain*

in knowledge elsewhere. The components contributing to the research methodology is outline in figure 9.

5.3 Design and creation

Oates (2006) mentions that the design and creation should be conducted in an iterative fashion; respectively by a five step iteration approach.

1. **Awareness**. The initial state. Oates (2006) explains, it is through the recognition or advancements of a problem. This recognition advancement pattern relates to new technology, new literature that is studied, or practitioners that express needs.

2. **Suggestion**. Creates a leap from the initial awareness of the intricate problem area or advancements. This leap explores and put a tentative idea to life.

3. **Development**. Create or implement something from this tentative idea from the previous step (suggestion). How this artefact is created depends upon the context of the area or field.

4. **Evaluation**. Tests the artefact that has been developed. Assessment of the proposed artefacts worth and deviations.

5. **Conclusion**. Sums up the results and are written up. The gain in knowledge is identified. If the results proves to be loose ends, unexpected results or anomalies, then these might be addressed for further research on the subject.

Figure 10 denotes a timeframe in which order each step in the iterative process should be followed. As with the beginning of this thesis, we have discussed the foundation on which the *awareness suggestion* and *development* are grounded.

In the next subsections I will describe my chosen methods for *evaluation*. Accordingly, evaluation consists of data gathering and analysis. Consequently, I will discuss my data gatherings validity and reliability. I will close this chapter by discussing ethical aspects.

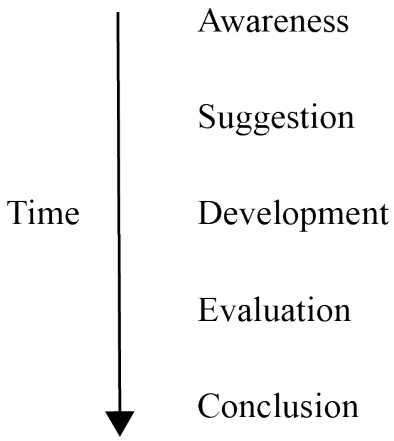


Figure 10: General overview of time proximity for the research design

5.3.1 Data gathering strategy

The strategy used for data gathering has been the stepwise-deductive induction (SDI) method by Tjora (2017). The scope of SDI is illustrated by figure 11.

In SDI the researcher is working in stages; from the gathering of data, all the way until the creation of theories or concepts derived from that data. The upward process illustrate the *inductive* part of the process (Tjora, 2017). The *deductive* aspect of the model refers to connecting previously known theories to the empirical data. The arrow from top to bottom in figure 11 illustrates this deductive pattern.

The data gathering method was an interview with a programming task on a computer. The programming task was captured with screen recording software. The interview distinguished between three-phases: warm-up, the central portion of the interview (questions and programming) and round up. The interview guide can be found in Appendix B.

5.3.2 Interviews

The most popular data gathering method in qualitative research is interviews or some deviation that resembles an interview (Tjora, 2017). Interviews can be divided into three types; structured, semi-structured and unstructured (Oates, 2006).

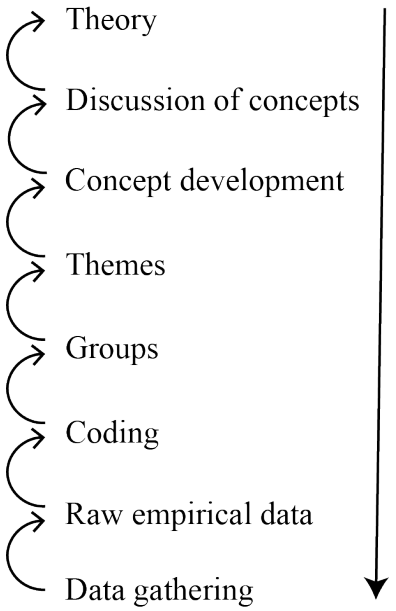


Figure 11: The stepwise-deductive induction method by (Tjora, 2017)

The goal of unstructured and semi-structured interviews is to create an environment where the participant at hand is allowed to speak their mind freely (Tjora, 2017; Oates, 2006). An important note is that these types of interviews should strive to give the participant a relaxed environment that allows for introspection and reflection about the topic at hand. Furthermore, questions are structured openly, giving digressions a particular value; participants can travel into topics that may be either invaluable or valuable. Digressions serve as a defence mechanism. Some aspects may not have been thought through before the interviews, and the digressions might turn out important for the inquiry (Tjora, 2017). The interview method used here is the semi-structured interview. The interview-guide can be found in appendix Appendix B.

Tjora (2017) advocate that interviews should be structured in three parts (introduction, reflection and ending), to ensure a natural flow for the interview. The introduction should contain easy questions that warm up the participant for more thoughtful answers in the central portion of the interview. The ending should normalize the situation,

and be a step where vital information about data confidentiality and treatment is discussed between both parties involved. Furthermore, a standardized procedure is to use audio recordings during the interview (Tjora, 2017; Oates, 2006). Recordings allow the interviewer to gather all that has been said while keeping a strict focus on the interview.

Transcription

Complete transcription of interviews is advocated. How one should transcribe interviews are a different question, where there is no accurate translation from oral dialogue to written text (Tjora, 2017). Consequently, the researcher should often include more details than necessary, where dialects, gestures, atmosphere, uncertainty indicators and verbalization problems might become important (Tjora, 2017; Oates, 2006).

For the interview transcriptions, I transcribed dialectic features when it felt appropriate and avoided tracking gestures. I transcribed uncertainty during the interviews and verbalization problems. All the aforementioned transcribing details contribute to the overall atmosphere and memory of an interview.

5.3.3 Observation

Observational studies have often been connected to anthropology, referred to as ethnography. This line of observation tries to explore social constructs in their natural habitat (Tjora, 2017). The use of observations in regards to computing research is often linked with human-computer interaction (Oates, 2006). These observations are often used to explore how people do their jobs, or in this particular scenario, how students use IDE's. Observation as a data-gathering method has a wide range of approaches, where each approach has its own spectrum (Oates, 2006).

With an observational method, there are different types of participation; the amount of involvement with the interview-object. Oates (2006) outlines four types of observation: complete observer, complete participant, participant-observer and practitioner-researcher. The method used for this inquiry, is an overt participant-observer strategy, with record-keeping using a screen recorder of the participant computer

screen during programming. The observational template can be found in appendix Appendix H.

5.4 Analysis

The main goal of the analysis is to create structured data that allows other readers to gain knowledge in your field of study while removing the need for the reader to go through the raw data themselves (Tjora, 2017). Qualitative data analysis is not as straight forward as a quantitative approach. Quantitative data can use established statistical and mathematical models and procedures, meanwhile, qualitative data depends on the skill of the researcher; intellectual capacity and creativity (Oates, 2006; Tjora, 2017).

Tjora (2017) write about the barrier of starting with qualitative data analysis. A week of gathering data from a field study might generate hundreds of pages of notes (Oates, 2006). SDI aims at reducing complexity into a structured format by trusting the empirical data, going step by step, avoiding premature conclusions and have a systematic foundation.

5.4.1 Coding

Coding is the first step in the qualitative analysis from the standpoint of the SDI model. SDI only operate with one coding level and guarding this with a strictly inductive approach (Tjora, 2017). In grounded theory and SDI, the coding is referred to as *open coding*. Oates (2006) use three levels of codes, *open coding*, *axial coding* and *selective coding*, which resembles SDI in many ways. One hallmark of SDI is that the codes are in close proximity to the empirical data (Tjora, 2017). The analyser should try name codes according to the terminology used from the transcribed interviews, or situational occurrences during the interview; safeguarding the data.

The strategy used to code all the transcribed data was done in an iterative fashion. First, I reviewed all the interviews getting an overview of the data, taking some handwritten notes to find some clues potentially. Tjora (2017) write about jumping straight onto the first document. I reviewed the transcriptions first.

The strategy for coding used for the transcribed interviews where that any interesting statement, phrase, word, metaphor or verb could

eventually become a code. The strategy creates good reference points throughout the data. Unique codes can be a good thing; it maintains the researchers' memory (Tjora, 2017).

The emergence of CAQDAS tools (computer-aided qualitative data analysis software) has been a staple in England and the USA for qualitative data analysis (Tjora, 2017). CAQDAS tools can be beneficial for the researcher, but some argue that the analysis can be restricted by the software capabilities, the learning curve of such a tool, and the disconnectedness from the raw data (Oates, 2006). However, CAQDAS tools have their potency within their ability to maintain the original transcripts during coding, rather doing manual and physical approaches that can distort the original data (Tjora, 2017). Features of CAQDAS are often two parallel systems; one system that tracks codes, and another system that keep track of the edges between codes. For this inquiry the software tool Nvivo[10] was used. In Nvivo codes are referred to as *nodes*. Categories can be coded with nested nodes (a node containing a set of nodes), and themes can be nested nodes of categorical nodes. Either way, Nvivo can systematically form hierarchical structures. Being lightweight and easygoing, Nvivo allowed data to be structured in a reasonable manner, without disrupting original transcripts and maintaining context.

5.4.2 Codes

By interviewing and observing students in a programming interview, two different sets of data had to be managed — transcriptions from interviews and video footage from their programming task. Interviews and video observations were coded separately. There were in total of 14 participants (11 male and three female). Fourteen transcripts and video observations resulted in 78 generated codes from transcripts and 17 codes from observations. More information about these codes can be found in appendix Appendix F and Appendix H.

[10]Nvivo, licence provided by NTNU. More information about Nvivo can be found here `https://www.qsrinternational.com/nvivo-qualitative-data-analysis-software/`

5.4.3 Groups

The next step in the process from the SDI model (figure 11) where the process of grouping together codes inductively. Grouping these codes consists of fitting each code in to a thematic context (Tjora, 2017), whilst filtering irrelevant code.

In table 5 you will find all the categorical groups that formed from transcriptions, and table 6 present the categories from video observations.

Category	Description	Codes
TDT4100 as a course	Students own words and experiences with the course TDT4100. Assignments, logistics, resources and help	8
Challenges with TDT4100	Challenges that students have in TDT4100	7
Learning to code	Students own statements about how they learn programming the most efficiently and thoroughly	16
Student needs	Statements and descriptions of what students feel they need in addition to the original course curricula	3
Past experiences	Past experiences with programming, courses and how they relate to TDT4100	3
Experiences with Gitpod	Students experience with using Gitpod during their interview	15
Experiences with Eclipse	Students exeprience with Eclipse during the semester	9
Experiences with IDEs in general	General IDE experiences gathered from diverse contexts	10
Git; attitude and relevance	Their attidutes and outlook on git as a version control system	2
Programming in Gitpod	Students experience with programming tasks in Gitpod	3

Table 5: Categories formed by the open codes during the grouping process

Observation category	Description	Codes
Uncertainty and obstacles	Students show uncertainty while programming. Package problems, class declarations, folder structure and file extensions	6
Nervousness	Students show hints of nervousness during the programming interview	3
Familiarity and shortcuts	Students have their familirarities and shortcuts that they are used to from previous experiences	4
Effective workflow and feedback	How well students perform using Gitpod during the programming interview. Feedback from the system, error messages and IDE utility	3
Eclipse automation realizations	Moments of realizations occur, where automation provided by Eclipse become obvious	1

Table 6: Categories from video observations

5.4.4 Themes

Whether another step of coding is required, comes down to the number of groups that can be summarized in themes. Tjora (2017) says that a master thesis may suffice with only groups, excluding the creation of themes. Themes were generated from the categories of transcripts, as they had adequate amounts of codes. The categories formed from video observations were fewer, thus no themes were formed. In table 7, generated themes are presented.

The themes in table 7 are interrelated. This creates a dynamic in which the concept development described by Tjora (2017) do not fit. A shift in perspective move towards a conceptual model that connects the themes. Creswell (2002) name this *interrelated themes*.

Theme	Description
Students needs, experiences and learning	Students own experiences with coding, tools and how they learn
TDT4100: A student perspective	A student perspective of the course TDT4100. Thoughts on how the lectures work. How the course is structured. Thoughts and experience with the curricula.
Technical IDE aspects	Thoughts and experience with the relevant technology used in the course and how that students relate to the technology used.

Table 7: Themes formed from categories

5.5 Research quality

The three criteria of *validity*, *reliability* and *generalizability* are often indicators of research quality (Tjora, 2017). I will describe and discuss these criteria in regards to the research methodology presented. First, the metric of *validity* will be discussed. Reliability will follow as the second parameter to the equation. *Generalizability* will be presented as a part of the discussion, in Section 7.4

5.5.1 Validity

In research, validity raises the question about whether or not the answers we find in research are valid (Tjora, 2017). We can strengthen the validity of the methodological processes by interpreting how they conform to the questions asked, and by inviting individuals to reflect upon the relevancy and precision to the choices made by the data generating methods and through relevant theory for analysis (Tjora, 2017). These validity metrics have been adhered to, by presenting clear methodological choices through research questions, methodology and constructed themes.

The researcher also has a personal position in regards to the inquiry. An ideal is that the researcher maintains a neutral or an objective stance. The engagement of the researcher in a given inquiry can

be looked upon as noise that can challenge and influence the results. However, the researcher is essential, but complete objectivity and neutrality in qualitative research do not exist (Tjora, 2017). A researcher should be open-minded for change and insight.

My personal engagement might infer with the results. I initially had a personal preference for Gitpod over Eclipse that could potentially infect the participants while interviewing, giving biased questions and getting biased answers. Meanwhile, I tried to keep an objective stance, hiding my personal preferences. But after a while, that personal preference for Gitpod, slowly but steadily diminished, as the appreciation for Eclipse increased. How influential my engagement were to create biased questions and answers, is debatable, but it is important to describe the situation. However, I do believe that my diminished engagement for Gitpod may have created a more neutral analysis and discussion.

During observations, and interviews alike, some participants were noticeably nervous amid programming while being observed. Participants exhibited covert and overt signs such as: arithmetic that fails, directly stating that being observed was intense, programming hectically and they disremembered and ignore simple tasks and feedback. A major validity concern, as the situation do not resemble a real-life context in which the participants operate as normal.

5.5.2 Reliability

If some other person, team or researcher would have executed the same procedures that have been presented, they would have gotten the same results — a test of the reliability of the research. Tjora (2017) write about the following steps that can strengthen the reliability: clarify verbatim quote selection, clarify how the participants were chosen and describe the relationship between researcher and participants.

The participants where chosen from a pool of 600 students. I had a presentation of my inquiry in a TDT4100 lecture. After my presentation, I sent sheets of paper through the auditorium. Those that were interested would write down their email, and I would contact them after, setting up the interview. I had initially 37 interested students. Due to the lack of email responsiveness and other factors, I got 16 students aligned for interviews. There were two participants that I had a relationship with (through other mediums), while the rest of the

participants were completely new interactions.

Quotes were deliberately chosen to extract the most important aspects of the analysis. The chosen method was to present more than just a single quote for a given code or category, to highlight that a category was not restricted in terms of data. A broad and diverse presentation was also a criterion, with verbatim quotes from each and every participant.

5.6 Ethical concerns

Data anonymization and treatment of participants are two important aspects of ethical concerns with the research presented. The latter represents the presentation of data, while the former relies on participants well-being. Some research might address burdensome personal affairs, while others do not. Information about withdrawal conditions (participants can abort any time) are crucial, and sensitivity for audio recording should be treated and respected accordingly (Tjora, 2017). The concern of the participants for this research was their potential response to the context of being interviewed and programming in front of a stranger — precautions were taken by being polite and lighthearted during the interviews, creating a pleasant atmosphere. The withdrawal conditions were part of the interview guide that was presented at every interview. See appendix Appendix B for the interview guide.

Anonymization of participants when presenting the data is also a concern. Some data cannot be guaranteed to be anonymous. Participants can come from a small pool of eligible people or specific environments. In some contexts, anonymization will greatly reduce analytical capacities. Anonymous presentations of participants are mandatory (Tjora, 2017). The 14 participants for this research were taken from a pool of over 600 enrolled students. A portion of them may be recognized through the raw data. Therefore appropriate measures were induced by avoiding names and age at all costs and presenting participants with anonymous names — interviews where written digitally through Microsoft OneDrive, approved by NSD. Recordings were taken with a password-protected iPhone, offline, with no synchronization options between other cloud providers. Audio-recordings, iPhone, cloud-provider and personal computer where password protected.

Before the gathering of data, the research process and methodology was sent for evaluation to the Norwegian Centre for Research Data

(NSD). The request was registered on the 2nd of February 2020, and was approved on the 6th of February 2020, with reference number 983188. The methodologies used to gather and store data are therefore approved by NSD as congruent with their laws on privacy, correctness, integrity and confidentiality. See appendix Appendix C for both the request and approval.

6 Evaluation and results

In qualitative research, there are two main approaches to present data from a qualitative study. One approach is to present verbatim quotes from the interviews while reporting the key findings of the research (themes). The key findings are discussed in a separate section, with links to the analysis and previous research. The second approach is to merge all three components, with verbatim quotes, key findings and discussion (Burnard et al., 2008). In this section the analysis will be presented first, separating it from the discussion.

The interview transcriptions are originally written in Norwegian, and therefore these have been translated to English. First, I will present my participants broadly and individually. For the remainder of this section, I will use the anonymous names during the analysis. The analysis will combine analysis with empirical data directly from the interview transcripts, describing my themes. Lastly, the model will be presented as a basis for my discussion.

6.1 Participants

There where in total 14 participants. 11 participants where male, and three were female. They where all university students in their second semester. Their names, occupation and integrity will be intact with anonymous names. Table 8 shows a combined statistical overview of their starting points. Individual presentations of my participants can be found in table 9. I will use the anonymous names throughout my presentation and analysis. All students had a 2 month period of using Eclipse (for mandatory assignments) before entering the interviews.

6.2 Students needs and learning

The theme of students needs, past experiences and learning centers the attention around students. The general findings from these interviews about students, is that they are relatively knowledgeable and experienced. Many of the students have previous experience from High School, side-projects, self-interest exploration and even videogames modding. 3 of the students did not have any computer science background entering university studies. From table 8 we see that they have used 16 different IDE's and 14 languages/frameworks, where some

Category	Listing
Previous IDE's	Xcode, Atom, Visual Studio Code, Visual Studio, Emacs, Arduino IDE, Thonny, PyCharm, IDLE, DreamWeaver, Notepad++, Brackets, Sublime, Eclipse, WebStorm, Jyputer Notebook
Languages and frameworks	Python, Javascript, Haskell, C++, Unity, Java, SQL, HTML, CSS, Swift, Arduino, PHP, Action-script, Flash

Table 8: Participant previously used IDE's, IT framworks and languages

Name	Years of IT experience	Favorite language
Jonas	2	Java
Tom	1-2	Python
Mitch	0	Java
Sarah	1.5	Java
Charles	4	Java
Peter	1.5	JavaScript
Evan	1.5	JavaScript
Shane	0.5	Java
Mia	2.5	Python
William	2.5	JavaScript
Ryan	1	Java
Katie	2.5	JavaScript
Logan	2.5	Python
Michael	0	Python

Table 9: Anonymous presentation of participants. Name, years of experience and their favorite programming language

students have used two or more. The range of previous IT experience goes from zero to four years.

6.2.1 Needs

The extent that the participants went in order to fit the course into their specific set of needs, where interesting. The students where typically engaged with explaining their setups. William described his stationary PC setup that were synchronized with his laptop:

> *"At home I've got a stationary computer, and I do not feel like I can synchronize my laptop with my stationary. When I am home, I often use remote desktop that connects to my laptop, so that I don't have to install Eclipse [...] I have created a solution with OneDrive cloud storage "* - William

Another example is that students seem to be using that they like the most, free of choice. Logan politely discussed his time back at High School where they were allowed to find IDE alternatives

> *"We did not have any installation rights on the computer, but we were allowed to download a portable version of Brackets, because it functioned well. Brackets had livepreview of HTML, and we could mark things inside our code, and it changed the livepreview. It was pretty smooth"* - Logan

While some seemingly are more interested and invested in the use of technology, some do not have that previous experience and show a complete contrast to these needs.

> *"I haven't programmed before, so I do not have any prerequisite to know whats good or bad"* - Michael

6.2.2 Learning

The learning category consists of a collection of topics about how they learn, outlooks on learning and their stance at the aspect at learning new things. An interesting finding about learning object-oriented programming, is that there is almost an unanimous vote on the activity of learning the object-oriented paradigm.

> *"I have to write code myself to understand it."* - Sarah

> *"By just coding. Definitely"* - Tom

"Just doing it, enough said" - Jonas

"Learning by doing [...] I try to do every single assignment. I learn so much from it" - Ryan

Throughout the interviews, something interesting occurred. While the students seemed "pro"-techonology and their non-lacking interest for it, there is something interesting about their outlooks on learning new things (IDE, tools, version control). I've called this type of learning activity "Sooner or later". The students show intuition about what they are supposed to learn in future contexts. But this intuition were filled with non-chalant outlooks on things, with statements such as "I just have to" and "I just have to learn it". Something show that the students begin to form a picture of necessity; tools that you just have to learn.

"Yeah but, you just have to learn it anyways [...] You just have to get used to it" - Ryan

Another example from a discussion about the version control system , the phenomena of "sooner or later" presents itself.

"If people eventually have to learn Git, I propose is just an advantage. You'll have to do it sometimes anyways" - Charles

From a discussion about the nuances between Gitpod and Eclipse with Mia, she says profoundly that "you have to".

"What we are using right now looks almost too similar to Eclipse, you'll have to grasp Eclipse also, you cannot just open Eclipse" - Mia

This type of expectancy from the students is quite interesting. Rather than just doing what the course material is expecting, they also have a futuristic mindset for future endeavours.

"I'll have to learn terminal soon" - Shane

6.3 TDT4100; A student perspective

The students do think that TDT4100 is a good course. It has loads of resources connected to it. A great support system of student assistants, with a great selection of assignments that fit to both ends of the spectrum of programming skills. Either way, there is one thing that we find; that is that the entry level is hectic, frustrating and a big leap from previous courses.

> *"It has been loads of trouble with Eclipse. It has been, especially in the starting phases, loads of problems with setting it up correctly"* - Jonas

> *"The biggest challenge is that object-oriented programming is a big leap from our introductory course"* - Tom

> *"Its the logic [..] how these logical structures are cooperating"* - Sarah

> *"It is the language that people use, like Professors and lectures. When they write assignments we are supposed to to, I will usually interpret those assignments for hours on end. That is quite challenging"* - Mitch

> *"The introductory course I thought were pretty simple. When we came here though, for some weird reason, everything became much more challenging"* - Mitch

Even though the entry level for object-oriented programming and TDT4100 seems hectic and hard, the amount of resources and availability of logistics seem to be more than acceptable. There seems to be nothing wrong about how the course is structured, as the grandiosity of the course seems to lay the foundation for learning to take place.

> *"I feel like we have so much resources available."* - Mia

> *"We've got a whole web page filled with all sorts of things that we're supposed to learn. And we've got discussion groups as well"* - Mia

"I feel the assignments are ... I learn so much from them [...] And the assignment scheme is very good too" - Ryan

"I think that here are so much available resources compared to other courses" - William

"I think that the assignments have been good, because you can choose which difficulty you want" - Charles

6.4 Past and present IDE experiences

This theme concentrates on how students are using gitpod, what they think about using such a tool, what students think about using Eclipse as an IDE, and all the general IDE knowledge put together. This section also includes programming observations.

6.4.1 General knowledge and preferences

In the introduction, in table 8, the overall experience of the students are presented. These ranges, from 0-4 years of experience, are a little surprising and impressive. Most of these students have had past experiences with programming either through previous university courses or high school courses (IT1, IT2). With all the previous exeprience, they begin to explain why they seem to prefer a certain IDE.

"(Atom IDE) is generally more cleaner and easier to concentrate on the code itself" - Evan

"I am used to VScode. It loads quite quickly" - Peter

But rather interestingly, even with the combined experience with IDE's within this sub-set of the students, there are some that do not even know the logic and purpose of an IDE, even though they have used such tools for quite some time.

"We have used it all the time, but I've never been given a formal definition for what it is" - Micheal

"The thing is, I've only used IDEs [...] There's something with it, I do not know what's behind the scenes" - Katie

While the more experienced students do often recognize that some IDE functionality may not have any purpose.

> *"It is a fullblown IDE that has so much functionality, we don't need all of that"* - Shane

Seemingly, the amount of knowledge about IDE's seem to be rather naturally partitioned, one particularly good comment came from Logan, with 2.5 years of experience.

> *"If I suddenly just opened Notepad and began writing a java class from scratch, then I do not think that i've ... it would've been shitloads of syntax errors, and I've probably forgotten include that, those, yidda, yadda, you know"* - Logan

6.4.2 Eclipse

I asked my participants what their experience with Eclipse was, and their outlooks. Not so interestingly, most of them have had problems with the installation process. A part from installing Eclipse, they mostly seemed positively indifferent.

> *"Nah, we haven't had so many bad experiences with it [...] It was a small hurdle in the starting phases with installing it [...] For me it has been pretty smooth"* - Mitch

The functionality of auto-suggestion with methods really stood out as exiting for Logan, and as a potential for learning more about the capabilities for the Java language, and the dependency that you have with using an IDE.

> *"When you press dot in Eclipse, and it suddenly shows 100 different methods, that is when I think about the use cases, there is so much I could've used"* - Logan

6.4.3 Gitpod and observations

The Gitpod programming session, and the round-up interview after the programming, show many things. One common observation during the interviews, was that many participants were nervous during the act of programming in front of another person. They show both covert and overt signs; Arithmetic that fails, telling me directly that it was intense, the programming gets hectic and they forget and ignore simple task definitions, feedback and IDE features. However, the nervousness diminishes after a while. Furthermore, the students are relatively quick adapters. There seem to be no problems in using and navigating the tool, and they learn quick from mistakes that occurred earlier. Nevertheless, nothing more seemed to be problematic. The programming task went well for almost all participants involved.

On the contrary, one key finding from the programming interviews, is the lack of understanding and experience with the package system, class definitions and file formats while programming with Java in Gitpod.

> *"Eclipse just create everything for me, I've never tought about it"* - Michael

> *"In Eclipse, those semicolons automatically appear when you're writing"* - Peter

> *"This does Eclipse for you"* - Logan

> *"I usually would've gotten the usual stuff (package, class, .java) if I would've done this in Eclipse"* - Katie

A comment from Ryan after he forgot to declare a package statement in the top of the file that he was working in, and was given a hint.

> *"Ahhh, of course ... Eclipse just does it for me".* - Ryan

Before the programming session, one participant breaks down this difficulty of understanding the folder structure and packages.

> *"[...] (Eclipse) It's structure is quite weird to what I am used to, projects and stuff."* - Evan

Many of the participants where quite used to the Eclipse IDE features. Productive features such as auto-generation of getters and setters, constructors and shortcuts. A common act of asking for these features where prominent during the observations.

After the interviews, I had a closing session with summarizing questions. The default reaction from students where that they all seemed moderately positive, it was quite similar to Eclipse and other types of IDE's that they had used before.

> *"Its very smooth that it is online and such, with regards to that it isn't dependent upon your PC. Its actually super smooth, if you suddenly loose all your stuff or something"* - Micheal

> *"Almost the same as Eclipse actually"* - Katie

Even though moderate passivity was prominent, some students were moderately skeptic at a certain point, having a lack of trust in the system.

> *"I feel that I am little sceptical for everything that is in the cloud, I don't feel like I've got it properly. What happens with this internet page? What if its down for maintenance. I think its best to have it on my own PC."* - Sarah

The programming started with the same trend; nervousness. After a while, they all seemed rather relieved and felt that they had mastered something. It was somehow their own brain-freeze that kept them from doing what they wanted, not the IDE itself.

> *"I thought that it should be considerably harder to use when it was something that I had not seen before"* - Mitch

> *"The thing that was hard, was my own brain freeze"* - Shane

6.5 The unified programming course

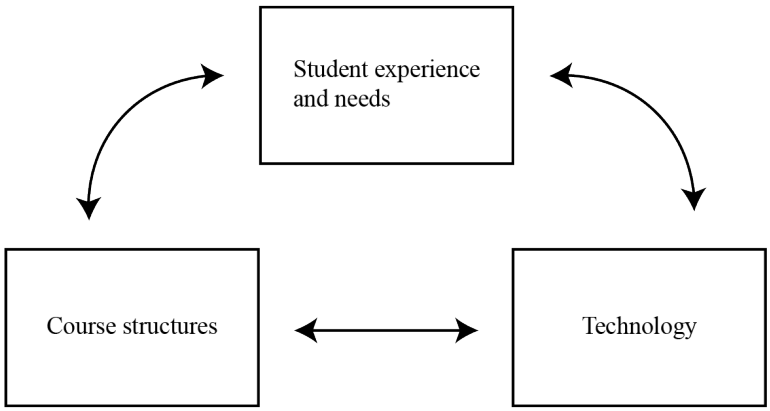


Figure 12: The unified programming course model. Described by the derived themes and their interconnections

My model of the unified programming course, derived and modelled after the generated themes consists of three interrelated components. There is the student that has thoughts about current technology usage, their identity derived from the set of needs to explore and configure, and how they learn programming. The component of looking at the course as a whole do not entirely derive from what technology that is used, rather, it is the whole combination; exercise selection, support systems, teaching styles and flow of information and resources. The technical aspects of IDE's show us that the technical is connected to the students, and the course-material that is taught.

7 Discussion

In the previous section, the interview and observational data were presented, and the model of the unified programming course. The model and data will furnish the upcoming discussion. In Section 2.1 the structure of the course TDT4100 were presented, in Section 2.2 the Gitpod environment and in Section 3 pedagogical theories and previous work were presented. All these sections provided a set of definitions, theories and previous work that will be used in coordination with the analysis for a discussion. The topics that this discussion is concerned about is the opportunities and pitfalls for introducing Gitpod for mandatory assignments in TDT4100, with a focus on pedagogy and a comparison to Eclipse.

7.1 Pedagogical implications

7.1.1 Mental constructs and containers

The results from this study indicate that the participants show a tendency to forget and have a slightly weird sense of discourse when operating with folders instead of Eclipses' package system. From Figure 2 we can see that Eclipse abbreviates folders to capture the internal Java structure of packages, with icons that look dissimilar to folders. The Theia IDE in Figure 3 show a folder structure that is usually found in computer GUIs. Also, the participants were found uncertain while declaring new files and their extensions. Supposedly, writing normal Java files and navigating packages for two months prior, did not transfer over to Gitpod. Eclipse creates all file extensions, package declarations and general class structures for you, creating a potential disconnect from actual Java programming and structural hierarchies, whereas Gitpod forces you to think about file declarations and the package system by having no direct support for such generation of file-extensions and declarations.

The interfered capability with Java structures is something that can have multiple causes. The results show that nervousness was usual amongst the participants. The causes for why participants show disability may therefore be even less clear. A plausible answer for the obscurity found might be that abstraction of notional machines may not be fully developed. Gitpod might have a different notional machine from the Eclipse environment. However, Eclipses package system

might not transfer students knowledge over to Gitpods regular folder structures. A language requires more than just learning the language itself (Du Boulay, 1986), and if pragmatics and key structural semantics are missing for how Java programs are structured, there may be potential issues with how Java is taugh, or Eclipse. Chen & Marx (2005) experimented with a more gradual introduction to Java semantics by working directly with the JDK, such that IDE usage would be recognized from students as a purposeful tool. Subsequently, using a more primitive tool (such as Gitpod) might be a healthy step towards a more refined understanding of Java programs in TDT4100 through strengthening of semantics, pragmatics and more defined notional machine of java structures. However, there were multiple statements from the participants about how they got stuck (programming within Gitpod) when Eclipses package system always generated those package declarations and file extensions for them. Alas, the lack of transferability from Eclipses package system to other IDE structures might be the issue, if such initial pragmatics do not exist. However, the participants did get a slight introduction for how Gitpod was packaged and structured. In turn, all packages and folder structures was identical to that of Eclipse. The only difference was the icons with package symbols found in Eclipse to that of traditional folders in Gitpod.

Eclipse is a standalone desktop application that is located at personal computers. Gitpod is an environment that consists of container images and cloud infrastructure. While the results show that students think that the IDE's are similar, the fact is that they are two facets of technology. The *notional machine* for such a cloud-distribution with Gitpod might create further distance and wrong interpretations. how does it work? A key component in using such technology and a concern is the difficulty in which students potentially create mental constructs that relates to cloud technology.

Students mental construction and understanding of how local environments work, has been seen as a complex system that students try to deal with simultaneously (Du Boulay, 1986). Adding another layer into the already existing complex system, with containers, might create more difficulty in understanding. A correct and effective notional machine for Gitpod, most likely requires effort, and wrongdoing of the mental model could potentially occur. Teaching must include guidance for these mental constructs, and how such a distribution works.

The students seemed rather surprised by how similar both IDE's where (Gitpod and Eclipse), to such extent that both of the two IDE's may have similar notional machines, if we look past the cloud environment from Gitpod. While, seemingly, as seen by these results, with some minor support during programming, all students were getting better at the programming task, and almost no issues were prohibiting the assignments from continuing (apart from time constraints) — standard goals with sums, loops, conditions and functions where becoming notably easy. Alas, using Gitpod for the given purpose of working with programming tasks, might be just as complicated compared to that of Eclipse.

Meanwhile, if the goal of creating independent students are of importance, other factors must be considered. Socio-cultural theory issues a paradigm in which mental constructs and knowledge are constructed through social interaction. Furthermore, efficient and productive learning should be localized in the zone of proximal development (Skaalvik & Skaalvik, 2013), and should, after that, be scaffolded appropriately (Belland, 2017) . Ultimately, to independently use Gitpod beyond the assignment scheme may create the same issue, where if students were given a choice in preference, they would use something else (Chen & Marx, 2005). Correspondingly, the results show that there were in total of 16 different IDEs used across all participants, and most of them had used two or more. Interestingly, more experienced students show indications of independently choosing other mediums to code, while less experienced students do not have enough knowledge, and therefore stick with the given IDE proposed by the course. However, the results found that some of the more experienced students found Gitpod more aesthetically pleasing than Eclipse, intuitive and similar to those IDEs that they had previously used. However, if Gitpod would be used outside of TDT4100, workspaces would need to be appropriately configured and stored in repositories on version control distributors. Given the experience of the students, there is little to no experience with repositories and Git. However, this might prove to be an incentive to incorporate more version control syllabus into the curricula, as the results found that students show signs of futuristic inclinations towards technology they "just" have to learn. However, such an imposed complexity might be useful if a prolonged time is given for adaptation, as more time yields the most results (Silva-

Maceda et al., 2016). A more concurrent teaching strategy might be appropriate. Meanwhile, if we are to understand Kölling et al. (2003), the introduction of object-oriented programming through professional IDEs limits the learning outcomes, while others suggest otherwise (Vihavainen et al., 2014). Meanwhile, the mechanism might also become increasingly complex and hard to understand (Du Boulay, 1986).

7.1.2 Pedagogy and IDE

BlueJ proposes a clear and concise pedagogy for teaching object-oriented programming. However, none of the presented techniques from the BlueJ system relates to existing pedagogical theories. Subsequently, BlueJ is not a substitute for professional IDEs and should be used thereafter. Theory on pedagogy relating to the use of IDEs are lean, where BlueJ (Kölling, 1999), IDEOL (Tran et al., 2013) and others (Itahriouan et al.; Goldman et al., 2011) are more proof of concepts. Collaboration through IDEs might be important to facilitate and support learning — illustration and concise frameworks alike. The usefulness of collaboration through IDE initiatives might be a result of a different shift in perspective. Either it is for educational purposes or technical prowess and suggestions. It can be debatable whether collaboration can be seen as a socio-cultural approach to teaching, or a cognitivistic approach (Dillenbourg et al., 2009). However, proof of concepts can give clear directions. While directions are clear, there seem to be lack of tests, proven by metrics that can give readers a more clear indication of whether such technology improves computer science learning and programming. While the discussion evolves around IDEs, in particular, the model of the unified programming course stands as a more wholesome picture; the components in courses constitute the building blocks for learning. IDE's cannot be seen as a single unit that stands for all problems or solutions related to learning programming. Learning programming should incorporate measures in regards to learning techniques, activities, students, external and internal factors.

Most IDEs that have been presented have their goals more oriented towards proof of concepts. My model of the unified programming course outlines a more wholesome picture. Through a set of strictly defined exercises that sequentially build upon one another (Table 1), TDT4100s assignments may scaffold over time (Sentance et al.,

2019). The lab-hours with assistants create opportunities for students to transfer knowledge from a MKO. Sharing might increase the availability of resources that increase the capacity for individual follow-up, where more adaptive and dynamic assessments can occur (Belland, 2017). Nevertheless, it is the utilization of the Gitpod functionalities that proves and adhere to socio-cultural theory and collaboration. Therefore, an important factor in deciding whether such pegagogy succeeds, is tightly related to how such technology is used.

7.2 Comparing IDEs

In order to compare the two technologies, some assumptions are necessary. Eclipse is a good IDE. The results found that participants have their own opinions about Eclipse, and they are mostly positive. This discussion revolves around how Gitpod differs and how those differences may give us more opportunities to explore alternatives for building blocks that constitute teaching and learning. First, the sharing capabilities of Gitpod are discussed. Lastly follows the metric of accountability.

7.2.1 Implications of sharing

The sharing component and browser compatibility of Gitpod are the key distinctions that separate Gitpod and Eclipse. To reiterate: Eclipse is a desktop application installed on a local computer, that can be configured to use API services. Gitpod is a web-based IDE (or system) that launches workspaces from repositories. The Theia IDE is the environment that programmers use in the browser and has all the utility described by declarative instructions given by the repository where a workspace was launched from.

A Snapshot taken from a workspace is a complete replica of the state and files. Eclipse, for TDT4100, have a sharing option resided in a one-to-many solution, while Gitpod can work as a many-to-many option. Eclipse can export code from staff to students, whereas export from students to teachers is more complicated (the complicated issue is that code has to be transferred through other mediums). Gitpod allows both sides of the equation (students and teachers) to copy, share and make code more accessible.

Accessibility is the contrasting difference between Gitpod and Eclipse. It allows for sharing of code between all parties involved. However,

there might be limited knowledge for how such utility can be utilized productively and whether sharing of environments can be a medium that increases learning outcomes. Do Students actually benefit from having such capabilities? Do academic staff and tutors eventually utilize the functionality to their advantage? Do such patterns impose more overhead than necessary? With little to no research on the topic of shareable environments, it is difficult to pinpoint exactly what accessibility is, or how it relates to learning environments in regards to programming.

Non-curricular activities

The non-curricular activities might be a byproduct of using Eclipse for assignments. The results show that some of the participants either have had problems with installing or reinstalling Eclipse trying to fix bugs. The results found that some of the participants showed a certain degree of mistrust in Eclipse when they tried to personalize it. Errors and bugs were related to each student's local computer, with a different distribution of operating systems, Eclipse and JDK versions. However, if a workspace gets buggy, or rather, exploring the IDE to such extents that it creates conflicts, students always have the option of starting with a new workspace that is provided by course administrators, and do not rely on "fixing" the problem themselves by reinstalling. However, one cannot anticipate that Gitpod will be free of bugs and errors, and therefore a centralized repository might become more problematic, as everyone is dependent upon that single repository for new workspaces. However, the results indicate that utilizing a centralized repository functions well. The non-curricular activity of installing or re-installing Eclipse is completely removed, Where the results show that all participants started their environments without any troubles. However, more time is necessary to answer the questions of whether Gitpod actually prove to be less prone to non-curricular activities.

Value in understanding

The results from this study found that many of the students gradually show a certain degree of understanding of how Gitpod works behind the scenes. They showed appreciation for the effectiveness of how the delivery of assignments may look like. Furthermore, they showed

interest for how they potentially could collaborate with each other, and how replication may help serve as a more effective way of getting help when they cannot anticipate or know how specific bugs/errors occur or how they are fixed. The results also showed scepticism of using such cloud technology. Results indicate that some of the students get bothered with how Gitpod handles security measures, while some see the value of not losing their progress due to potential computer failure. Some see the intrinsic value of using Gitpod on multiple units. An internet-only usable IDE also bothers some of them. Overall, the results indicate that they have quite a positive outlook on the use of Gitpod for TDT4100.

7.2.2 Accountability

TDT4100 currently has a structure in which each student has their own installation of Eclipse. The accountability of maintaining a correct state within those files, impose a position where there are multiple places where a condition may create an error. Every student is prone to errors. However, a common source of truth and more responsibility shifted to the maintainers of the repository, create another issue where if failures happen, it is not students that are held accountable for the errors. If a potential bug or error occurs at a repository responsible for assignments, all future workspaces launched from that repository contain errors. The transfer of accountability may relieve students of frustration, and more responsibility for the distribution is shifted to academic staff. From the tests of using Gitpod with sequential assignments, the results indicate that Gitpods ecosystem is relatively easy to operate, where only a single error occurred throughout 14 interviews. From my perspective, Gitpod is a liable system that continuously reproduces environments effectively and predictably.

7.3 Opportunities and pitfalls

The incorporation, testing and analysis of using Gitpod on students might better adhere to a collaborative and socio-cultural way of teaching. Subsequently, snapshots and workspaces create opportunities for how assignments can be distributed and shared amongst peers. However, repositories become the single source of truth for whether new instances work. Gitpod might create more complex notional machines, due to its cloud infrastructure; however, the participants adapt quickly to the new environment. Opportunities and pitfalls are outlined in table 10.

Opportunities	Pitfalls
Better adherence to socio-cultural and collaborative teaching approaches	Modification of course structures
Quick adaptation and similarity to other IDEs	Problematic for individual exploration
Reliable environments	Difficulty in creating mental constructs and understanding for how environments work
No installation procedures for the browser IDE	Preferences for desktop applications and technical difficulty in creating environments individually
Sharing of environments	Prone to plagiarism
Removal of non-curricular activities	New non-curricular activities might occur
Better understanding of internal java structures	
Starting with programming assignments immediately	
Student satisfaction and new technology	

Table 10: Opportunities and pitfalls from introducing Gitpod for mandatory assignments in TDT4100

7.4 Generalizability

The reader should interpret these results and the discussion with a *naturalistic* mindset. *Naturalistic generalization* describes a pattern of generalizability, where the reader (researcher) assess whether these findings are relevant for them (Tjora, 2017). Common terms that may describe the situation is *comparability*, *translatability* and *transferability* (Tjora, 2017). However, naturalistic generalizability goes against the nature of trying to generalize and can be seen as a way to elude the question of generalization. However, with *modarate* generalization the researcher identify which aspects might be generalizable and valid (Tjora, 2017). By the results proposed, courses with similar design might find the same results.

From my perspective, didactics and pedagogy in programming courses are areas that are yet fully established. Furthermore, the literature was difficult to find. Simultaneously, this was my first time doing research, where I feel that I do not have the capacity and knowledge to address whether such findings are generalizable. The reader should interpret my discussion as a new way of approaching pedagogy and didactics in programming courses, and how Gitpod might be a part of that approach.

7.5 Future work

This study has shown and discussed an approach of introducing a new tool for practical programming in an object-oriented course. From introducing the new tool, a new set of themes emerged. The themes were an interconnected scheme, where students experience and depend upon different notions of reliant components. Future work might be to explore these components in more depth, including how students learn programming and how that learning is mediated through the interconnected components of technology and course design. Perhaps the technology we use is a more critical factor than course design? Or rather, is the support systems of teachers and assistants more important?

However, future research on the topic should have a more practical and problem-solving approach. Longitudinal studies should incorporate Gitpod (and other programming tools), advancements in course design and teaching methodologies. Adherence to pedagogy and inspiration from other fields should play a role in designing programming

courses and teaching, for instance, looking at different kinds of methodological entries over a prolonged time with allowance for exploring new ideas. An example of this might be using Gitpod as a component in sequential courses over an entire study program; procedural programming, object-oriented programming, databases, software engineering, networking and distributed systems. Practical and theoretical unification of components. These approaches should incorporate both qualitative and quantitative measures: qualitative understanding of environments, log-file-analysis and statistical models for how they use tools that measure against other factors of quantitative measures and more in-depth explanations for how learning environments work.

8 Conclusion

With introducing Gitpod for object-oriented programming, the study has tried to uncover possible pedagogical implications, whether Gitpod has opportunities compared to Eclipse and probable pitfalls. By interviewing and observing students during programming in Gitpod, the research shows a mild indication of possible misconceived mental constructs. The Eclipses package system and auto-generating mechanisms might interfere with students structural understanding of traditional folder structures, and how Java structures operate on a deeper level. Subsequently, Gitpod might prove to be a viable technological component that eliminates installation procedures and configuration, holding academic staff more accountable and can furnish collaboration. However, Gitpod might be a complex environment for first-year students to use independently efficiently. Meanwhile, the component of an IDE within programming courses do not equate for all learning-related problems. The results indicate that the course structure and support systems are equally important factors. Overall, the participants have positive attitudes, recognises use-cases and similarities from previously used tools and adapt rather quickly with the use of Gitpod.

The next step is to produce knowledge, that can be put into a toolkit for various teaching purposes. Teachers of all types should incorporate something new, that could be utilized in different teaching contexts. Information technology is becoming more popular every day and has become a crucial part of society. We see changes in syllabuses in K-12 schools and the emergence of new study programs. Teachers should put effort into getting an overview of all the possibilities out there because there are no established norms for how we are supposed to teach coding. Exploration and experimentation should be of high importance for every teacher out there.

References

Belland, B. R. (2017). Instructional scaffolding in stem education. *Cham: Springer International*, .

Bosse, I. K., Armstrong, N., & Schmeinck, D. (2016). Is cloud computing the silver lining for european schools? *International Journal of Digital Society (IJDS)*, *7*, 1171–1176.

Bower, M., & Falkner, K. (2015). Computational thinking, the notional machine, pre-service teachers, and research opportunities. In *ACE* (pp. 37–46).

Burnard, P., Gill, P., Stewart, K., Treasure, E., & Chadwick, B. (2008). Analysing and presenting qualitative data. *British dental journal*, *204*, 429.

Campbell, S. (2016). Teaching cloud computing. *Computer*, (pp. 91–93).

Chen, Z., & Marx, D. (2005). Experiences with eclipse ide in programming courses. *J. Comput. Sci. Coll.*, *21*, 104–112.

Creswell, J. W. (2002). *Educational research: Planning, conducting, and evaluating quantitative*. Prentice Hall Upper Saddle River, NJ.

van Deursen, A., Mesbah, A., Cornelissen, B., Zaidman, A., Pinzger, M., & Guzzi, A. (2010). Adinda: a knowledgeable, browser-based ide. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 2* (pp. 203–206).

Dillenbourg, P. (1999). What do you mean by collaborative learning?

Dillenbourg, P., Järvelä, S., & Fischer, F. (2009). The evolution of research on computer-supported collaborative learning. In *Technology-enhanced learning* (pp. 3–19). Springer.

Docker (n.d.). What is a container? URL: `https://www.docker.com/resources/what-container`.

Du Boulay, B. (1986). Some difficulties of learning to program. *Journal of Educational Computing Research*, *2*, 57–73.

Dutta, M., Sethi, K. K., & Khatri, A. (2014). Web based integrated development environment. *International Journal of Innovative Technology and Exploring Engineering*, *3*, 56–60.

Gaikwad, T., Dhavale, P., Gaware, K., & Shivale, N. (2014). Web based ide. *International Journal of Research in Information Technology*, *2*.

Goldman, M., Little, G., & Miller, R. C. (2011). Real-time collaborative coding in a web ide. In *Proceedings of the 24th annual ACM symposium on User interface software and technology* (pp. 155–164).

Itahriouan, Z., Aknin, N., Abtoy, A., & El Kadiri, K. E. (). Building a web-based ide from web 2.0 perspective. *International Journal of Computer Applications*, *975*, 8887.

Johnson, R. B., Onwuegbuzie, A. J., & Turner, L. A. (2007). Toward a definition of mixed methods research. *Journal of mixed methods research*, *1*, 112–133.

Kölling, M. (1999). The problem of teaching object-oriented programming, part 1: Languages. *Journal of Object-oriented programming*, *11*, 8–15.

Kölling, M., Quig, B., Patterson, A., & Rosenberg, J. (2003). The bluej system and its pedagogy. *Computer Science Education*, *13*, 249–268.

Lautamäki, J., Nieminen, A., Koskinen, J., Aho, T., Mikkonen, T., & Englund, M. (2012). Cored: browser-based collaborative real-time editor for java web applications. In *Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work* (pp. 1307–1316).

Oates, B. J. (2006). *Researching Information Systems and Computing*. Sage Publications Ltd.

Redhat (2009). URL: `https://www.redhat.com/en/topics/middleware/what-is-ide`.

Sentance, S., Waite, J., & Kallia, M. (2019). Teaching computer programming with primm: a sociocultural perspective. *Computer Science Education*, *29*, 136–176. URL: `https://doi.org/10.1080/08993408.2019.1608781`. doi:10.1080/08993408.2019.1608781. `arXiv:https://doi.org/10.1080/08993408.2019.1608781`.

Silva-Maceda, G., David Arjona-Villicaña, P., & Edgar Castillo-Barrera, F. (2016). More time or better tools? a large-scale retrospective comparison of pedagogical approaches to teach programming. *IEEE Transactions on Education*, *59*, 274–281.

Skaalvik, E. M., & Skaalvik, S. (2013). *Skolen som læringsarena: selvoppfatning, motivasjon og læring*. Universitetsforlaget.

Sommerville, I. (2013). Teaching cloud computing: a software engineering perspective. *Journal of Systems and Software*, *86*, 2330–2332.

Sorva, J. (2013). Notional machines and introductory programming education. *ACM Transactions on Computing Education*, *13*, 8:1–8:31. doi:10.1145/2483710.2483713.

Tjora, A. (2017). Kvalitative forskningsmetoder i praksis. 3. utgave. *Oslo: Gyldendal norsk forlag AS*, .

Tran, H. T., Dang, H. H., Do, K. N., Tran, T. D., & Nguyen, V. (2013). An interactive web-based ide towards teaching and learning in programming courses. In *Proceedings of 2013 IEEE International Conference on Teaching, Assessment and Learning for Engineering (TALE)* (pp. 439–444). IEEE.

TypeFox (2019). Smart tools for smart people. URL: https://typefox.io/.

Vihavainen, A., Helminen, J., & Ihantola, P. (2014). How novices tackle their first lines of code in an ide: Analysis of programming session traces. In *Proceedings of the 14th Koli Calling International Conference on Computing Education Research* Koli Calling '14 (p. 109–116). New York, NY, USA: Association for Computing Machinery. URL: https://doi.org/10.1145/2674683.2674692. doi:10.1145/2674683.2674692.

Yvonne Feilzer, M. (2010). Doing mixed methods research pragmatically: Implications for the rediscovery of pragmatism as a research paradigm. *Journal of mixed methods research*, *4*, 6–16.

Appendix A Informed consent letter

Jørgen Helgå Stamnes
Institutt for datateknologi og informatikk
Sem Sælandsvei 9, 7491 Trondheim
41403910
jorghs@stud.ntnu.no

Trondheim, Februar 2020

Vil du delta i forskningsprosjektet

"En studie av IT-studenters erfaringer og bruk av programmeringsverktøy"?

Dette er et spørsmål til deg om å delta i et forskningsprosjekt hvor formålet er å se nærmere på erfaringer og meninger om bruk av IDE, og øvingsopplegget i objektorientert programmering. Samtidig ønsker vi å få erfaringer ved å prøve ut en alternativ løsning . I dette skrivet gir vi deg informasjon om målene for prosjektet og hva deltakelse vil innebære for deg.

Formål

Formålet med denne studien er å se på studenters erfaringer med integrated development environments (IDE) og samtidig prøve ut en alternativ IDE. Målet er å samle inn data for å kunne finne ut av hvilke pedagogiske og tekniske konsekvenser et slikt alternativ medbringer, og er med på å styrke forståelsen av hvordan NTNU driver undervisning, spesielt i Objektorientert programmering. Arbeidet er en del av min masteroppgave som skal ferdigstilles juni 2020.

Hvem er ansvarlig for forskningsprosjektet?
Prosjektet er i samarbeid med førsteamanuensis Hallvard Trætteberg og stipendiat Madeleine Lorås ved Institutt for datateknologi og informatikk på NTNU.

Hvorfor får du spørsmål om å delta?
Utvalget er trukket, og er en del av, alle studenter som studerer og er oppmeldt i faget objektorientert programmering**.**

Hva innebærer det for deg å delta?

Hvis du velger å delta i prosjektet, innebærer det at du blir med på et programmeringsintervju. Intervjuet er en kombinasjon av et semi-strukturert intervju, og en deltagende observasjon i form av en programmeringseanse (brukbarhetstest). Opplysninger som blir samlet inn er følgende: skjermopptak av programmeringen, lydopptak av intervjuet, loggføring av observasjonen og egne intervjunotater. Det kommer til å være satt av én til halvannen time til hver informant som velger å delta.

Det er frivillig å delta

Det er frivillig å delta i prosjektet. Hvis du velger å delta, kan du når som helst trekke samtykke tilbake uten å oppgi noen grunn. Dette kan gjøres muntlig, skriftlig eller elektronsik. Alle opplysninger om deg vil bli anonymisert. Det vil ikke ha noen negative konsekvenser for deg hvis du ikke vil delta eller senere velger å trekke deg. Deltagelsen vil ikke ha noen innvirkning på evaluering og karaktersetting i faget objektorientert programmering.

Ditt personvern – hvordan vi oppbevarer og bruker dine opplysninger

Vi vil bare bruke opplysningene om deg til formålene vi har fortalt om i dette skrivet. Vi behandler opplysningene konfidensielt og i samsvar med personvernregelverket.

- De som har tilgang på dataene vil være student Jørgen Helgå Stamnes, førsteamanuensis Hallvard Trætteberg og stipendiat Madeleine
- Tiltakene vi gjøre for å sikre at ingen uvedkommende får tilgang på personopplysningene er følgende: Navn og kontaktopplysninger vil bli lagret med en nøkkelkode som er adskilt fra øvrige data, datamaterialet lagres på Microsoft Office Cloud, som har avtale med NTNU. Dataene blir kryptert under forsendelse og lagring. Dataene har adgangsbegrensinger.
- Hvis en mulig publikasjon skulle være relevant, vil de som har deltatt ikke være gjennkjennbare.

Hva skjer med opplysningene dine når vi avslutter forskningsprosjektet?

Prosjektet skal etter planen avsluttes 02.06.2020. Innsamlede data vil bli slettet etter prosjektets slutt, senest 1. September 2020.

Dine rettigheter (obligatorisk)

Så lenge du kan identifiseres i datamaterialet, har du rett til:

- innsyn i hvilke personopplysninger som er registrert om deg,
- å få rettet personopplysninger om deg,
- få slettet personopplysninger om deg,
- få utlevert en kopi av dine personopplysninger (dataportabilitet), og
- å sende klage til personvernombudet eller Datatilsynet om behandlingen av dine personopplysninger.

Hva gir oss rett til å behandle personopplysninger om deg?

Vi behandler opplysninger om deg basert på ditt samtykke.

På oppdrag fra institutt for datateknologi og informatikk har NSD – Norsk senter for forskningsdata AS vurdert at behandlingen av personopplysninger i dette prosjektet er i samsvar med personvernregelverket.

Hallvard Trætteberg: e-post (Hal@ntnu.no) eller mobil 91897263
Madeleine Lorås: e-post (Madeleine.loras@ntnu.no) eller mobil 92885006

Hvor kan jeg finne ut mer?
Hvis du har spørsmål til studien, eller ønsker å benytte deg av dine rettigheter, ta kontakt med:

- Institutt for datateknologi og infromatikk ved Hallvard Trætteberg, Jørgen Helgå Stamnes eller Madeleine Lorås
- Vårt personvernombud: Thomas Helgesen, e-post (thomas.helgesen@ntnu.no) tlf: 93079038
- NSD – Norsk senter for forskningsdata AS, på epost (personverntjenester@nsd.no) eller telefon: 55 58 21 17.

Med vennlig hilsen

Hallvard Trætteberg
(Forsker/veileder)

Jørgen Helgå Stamnes
(Student)

Samtykkeerklæring

Jeg har mottatt og forstått informasjon om prosjektet ***En studie av IT-studenters erfaringer og bruk av programmeringsverktøy***, og har fått anledning til å stille spørsmål. Jeg samtykker til:

- Å delta i et intervju med deltagende observasjon av programmering

Jeg samtykker til at mine opplysninger behandles frem til prosjektet er avsluttet, ca. 2. Juni 2020

(Signert av prosjektdeltaker, dato)

Hallvard Trætteberg: e-post (Hal@ntnu.no) eller mobil 91897263
Madeleine Lorås: e-post (Madeleine.loras@ntnu.no) eller mobil 92885006

Appendix B Interview guide
Intervjuguide

Informant nummer ___

Introduksjon

Presentere meg selv og litt om forskningsprosjektet

- Jørgen, 25 år, studerer lektor (lærer) i informatikk og matematikk, men skal begynne å jobbe som IT-konsulent. Vært java-utvikler i sommer, har konsulentjobb på deltid, B i OOP
- Jobber nå med en masteroppgave som omhandler IDE bruk og ny IDE teknologi. I dag skal vi teste ut Gitpod; en IDE som kjøres i netteleseren og er operativsystem uavhengig.
- Du er her fordi du har faget TDT4100 - Objektorientert programmering - og meldte deg på.

Informasjon om opptak

- Jeg ønsker å ta opp intervjuet på lydopptak, der gjør det lettere å analysere dataene i etterkant, og er en anbefalt metode/strategi for kvalitative studier. Jeg kommer også til å ta skjermopptak mens du programmerer, dette også for å gjøre det lettere å analysere i etterkant.
- Du blir helt anyonymisert. Det er kun jeg, bi-veileder og veileder som har tilgang på disse dataene.
- Opptakene slettes i løpet av sommeren 2020

Litt om intervjuet

- Kommer til å være et todelt intervju; programmering og vanlig intervju. Starter med litt oppvarmingsspørsmål, deretter beveger vi oss inn i intervju-delen. Etter det blir det en programmeringseanse, og til slutt flere spørsmål med avsluttning
- Kommer til å stille urelevante og relevante spørsmål. Alle svar er gode svar, jeg er mest interessert i høre hva du har å si!
- Du kan avbryte intervjuet når som helst

Innledning/oppvarming

1. Hva er ditt favoritt programmeringsspråk?
2. Cola eller pepsi?
3. 1-10, hvor god er du til å progge?
4. I hvilken posisjon/tid/omgivelse er ditt favoritttindspunkt for å progge?
5. Har du brukt git, github eller gitlab før? Hvor mye?
6. Hvorfor valgte du å studere informatikk / data (mulige variasjoner) på NTNU?
7. Hvor lange har du drevet med programmering / IT / Data?

Hoveddel

8. Hva er dine erfaringer med bruk av IDE (Eclipse)?
    a. Hvorfor tror du vi bruker IDE's?
    b. Hva synes du er mest utfordrende med dagens løsning?
9. Hvordan føler du at du lærer programmering best? Er det knyttet til verktøyene, teorien, forelesningene, prosjekt, øvinger, osv.
10. Føler du at det er noe du mangler i fag som dreier seg om programmering?
11. En IDE automatiserer mange prosesser og gjør dem usynlige for sluttbrukeren. Har det å bryte ned disse automatiske prosessene konsekvenser for læring?

Prøver ut et alternativt opplegg med Gitpod, og jeg trer inn i en deltage obersvasjonsrolle. Programmeringsseanse.

12. **Gjennomføre en virkårlig øving i OOB, med gitpod**
    c. **Kjøre gitpod container fra et github repository**
    d. **Informant skal selv velge oppgave**
13. **Kjøre tilhørende tester under programmeringen**
14. **Levere øvingen (fiktiv leveranse) med snapshots**
15. **Avlsutte arbeidsområde**
16. **Starte og begynne på ny øving**
17. **Lagre endringene (arbeidsområde)**

18. Hvordan syns du det gikk? Noe som kommer opp i tankene som første erfaring med dette?

19. Hvordan synes du det fungerer i bruk? Forstår du flyten i hva vi eventuelt prøver å få til? Hva synes du om det?
20. Jeg har fortalt deg litt om rammene rundt gitpod, ser du noen umiddelbare muligheter/begrensninger dette kan medføre?
21. Gitpod krever mer git kunnskaper, hva tenker du om det?
22. Det er kjent at eclipse til tider kan føre til mye hodebry i startfasen m.m., hvordan tror du en slik ready-to-code approach har for det å lære det å progge?
23. Tror du dette kunne blitt brukt i andre fag? Eventuelt Hvordan?
24. Noe du savner i den nåværende løsningen?
25. Totalinntrykk?
26. Noe du har lyst å tilføye? Noe som har dukket opp på tampen?

Appendix C NSD

NSD NORSK SENTER FOR FORSKNINGSDATA

Meldeskjema 983188

Sist oppdatert

04.02.2020

Hvilke personopplysninger skal du behandle?

- E-postadresse, IP-adresse eller annen nettidentifikator
- Lydopptak av personer

Type opplysninger

Skal du behandle særlige kategorier personopplysninger eller personopplysninger om straffedommer eller lovovertredelser?

Nei

Prosjektinformasjon

Prosjekttittel

En studie av IT-studenters erfaringer og bruk av programmeringsverktøy

Begrunn behovet for å behandle personopplysningene

Behovet er for å ta lydopptak av intervjuene, for videre analyse. Tjora (2012) påpeker at i intervjuer bruker man alltid en eller annen form for lydopptak.

Tjora, Aksel (2012) - Kvalitative forskningsmetoder i praksis

Ekstern finansiering

Type prosjekt

Studentprosjekt, masterstudium

Kontaktinformasjon, student

Jørgen Helgå Stamnes, jorghs@stud.ntnu.no, tlf: 41043910

Behandlingsansvar

Behandlingsansvarlig institusjon

Norges teknisk-naturvitenskapelige universitet NTNU / Fakultet for informasjonsteknologi og elektroteknikk (IE) / Institutt for datateknologi og informatikk

Prosjektansvarlig (vitenskapelig ansatt/veileder eller stipendiat)

Hallvard Trætteberg, hal@ntnu.no, tlf: 91897263

Skal behandlingsansvaret deles med andre institusjoner (felles behandlingsansvarlige)?

Nei

Utvalg 1

Beskriv utvalget

Studenter i ett bestemt fag; objektorientert programmering

Rekruttering eller trekking av utvalget

Organisasjonen tar kontakt med sine studenter. Masterstudent tar kontakt med organisasjonens studenter.

Alder

19 - 30

Inngår det voksne (18 år +) i utvalget som ikke kan samtykke selv?

Nei

Personopplysninger for utvalg 1

- E-postadresse, IP-adresse eller annen nettidentifikator
- Lydopptak av personer

Hvordan samler du inn data fra utvalg 1?

Personlig intervju

Grunnlag for å behandle alminnelige kategorier av personopplysninger

Samtykke (art. 6 nr. 1 bokstav a)

Informasjon for utvalg 1

Informerer du utvalget om behandlingen av opplysningene?

Ja

Hvordan?

Skriftlig informasjon (papir eller elektronisk)

Tredjepersoner

Skal du behandle personopplysninger om tredjepersoner?

Nei

Dokumentasjon

Hvordan dokumenteres samtykkene?

- Elektronisk (e-post, e-skjema, digital signatur)
- Manuelt (papir)

Hvordan kan samtykket trekkes tilbake?

Ved muntlig, elektronisk eller skriftlig forespørsel.

Hvordan kan de registrerte få innsyn, rettet eller slettet opplysninger om seg selv?

Ved skriftlig eller muntlig forespørsel til ansvarlig vil alle data bli utlevert/slettet

Totalt antall registrerte i prosjektet

1-99

Tillatelser

Skal du innhente følgende godkjenninger eller tillatelser for prosjektet?

Behandling

Hvor behandles opplysningene?

- Maskinvare tilhørende behandlingsansvarlig institusjon
- Ekstern tjeneste eller nettverk (databehandler)

Hvem behandler/har tilgang til opplysningene?

- Prosjektansvarlig
- Student (studentprosjekt)
- Interne medarbeidere
- Databehandler

Hvilken databehandler har tilgang til opplysningene?

Microsoft Office 365, NTNU har databehandleravtale.

Tilgjengeliggjøres opplysningene utenfor EU/EØS til en tredjestat eller internasjonal organisasjon?

Nei

Sikkerhet

Oppbevares personopplysningene atskilt fra øvrige data (kodenøkkel)?

Ja

Hvilke tekniske og fysiske tiltak sikrer personopplysningene?

- Opplysningene anonymiseres
- Opplysningene krypteres under forsendelse
- opplysningene krypteres under lagring
- Adgangsbegrensning

Varighet

Prosjektperiode

02.03.2020 - 02.06.2020

Skal data med personopplysninger oppbevares utover prosjektperioden?

Nei, data vil bli oppbevart uten personopplysninger (anonymisering)

Hvilke anonymiseringstiltak vil bli foretatt?

- Personidentifiserbare opplysninger fjernes, omskrives eller grovkategoriseres
- Koblingsnøkkelen slettes
- Lyd- eller bildeopptak slettes

Vil de registrerte kunne identifiseres (direkte eller indirekte) i oppgave/avhandling/øvrige publikasjoner fra prosjektet?

Nei

Tilleggsopplysninger

NSD Personvern
06.02.2020 15:49

Det innsendte meldeskjemaet med referansekode 983188 er nå vurdert av NSD.

Følgende vurdering er gitt:

Det er vår vurdering at behandlingen av personopplysninger i prosjektet vil være i samsvar med personvernlovgivningen så fremt den gjennomføres i tråd med det som er dokumentert i meldeskjemaet 06.02.2020 med vedlegg. Behandlingen kan starte.

MELD VESENTLIGE ENDRINGER
Dersom det skjer vesentlige endringer i behandlingen av personopplysninger, kan det være nødvendig å melde dette til NSD ved å oppdatere meldeskjemaet. Før du melder inn en endring, oppfordrer vi deg til å lese om hvilke type endringer det er nødvendig å melde:
nsd.no/personvernombud/meld_prosjekt/meld_endringer.html

Du må vente på svar fra NSD før endringen gjennomføres.

TYPE OPPLYSNINGER OG VARIGHET
Prosjektet vil behandle alminnelige kategorier av personopplysninger frem til 02.06.2020.

LOVLIG GRUNNLAG
Prosjektet vil innhente samtykke fra de registrerte til behandlingen av personopplysninger. Vår vurdering er at prosjektet legger opp til et samtykke i samsvar med kravene i art. 4 og 7, ved at det er en frivillig, spesifikk, informert og utvetydig bekreftelse som kan dokumenteres, og som den registrerte kan trekke tilbake. Lovlig grunnlag for behandlingen vil dermed være den registrertes samtykke, jf. personvernforordningen art. 6 nr. 1 bokstav a.

PERSONVERNPRINSIPPER
NSD vurderer at den planlagte behandlingen av personopplysninger vil følge prinsippene i personvernforordningen om:

- lovlighet, rettferdighet og åpenhet (art. 5.1 a), ved at de registrerte får tilfredsstillende informasjon om og samtykker til behandlingen
- formålsbegrensning (art. 5.1 b), ved at personopplysninger samles inn for spesifikke, uttrykkelig angitte og berettigede formål, og ikke viderebehandles til nye uforenlige formål
- dataminimering (art. 5.1 c), ved at det kun behandles opplysninger som er adekvate, relevante og nødvendige for formålet med prosjektet
- lagringsbegrensning (art. 5.1 e), ved at personopplysningene ikke lagres lengre enn nødvendig for å oppfylle formålet

DE REGISTRERTES RETTIGHETER
Så lenge de registrerte kan identifiseres i datamaterialet vil de ha følgende rettigheter: åpenhet (art. 12), informasjon (art. 13), innsyn (art. 15), retting (art. 16), sletting (art. 17), begrensning (art. 18), underretning (art. 19), dataportabilitet (art. 20).
NSD vurderer at informasjonen som de registrerte vil motta oppfyller lovens krav til form og innhold, jf. art. 12.1 og art. 13.

Vi minner om at hvis en registrert tar kontakt om sine rettigheter, har behandlingsansvarlig institusjon plikt til å svare innen en måned.

FØLG DIN INSTITUSJONS RETNINGSLINJER
NSD legger til grunn at behandlingen oppfyller kravene i personvernforordningen om riktighet (art. 5.1 d), integritet og konfidensialitet (art. 5.1. f) og sikkerhet (art. 32).

Microsoft Office 365 er databehandler i prosjektet. NSD legger til grunn at behandlingen oppfyller kravene til bruk av databehandler, jf. art 28 og 29.

For å forsikre dere om at kravene oppfylles, må dere følge interne retningslinjer og eventuelt rådføre dere med behandlingsansvarlig institusjon.

OPPFØLGING AV PROSJEKTET
NSD vil følge opp ved planlagt avslutning for å avklare om behandlingen av personopplysningene er avsluttet.

Lykke til med prosjektet!

Tlf. Personverntjenester: 55 58 21 17 (tast 1)

Appendix D Assignment descriptions

Appendix D.1 Assignment

Øving 1: Objekter og klasser, tilstand og oppførsel

Øvingsmål:

- Bli kjent med Java-syntaks og bruk av Eclipse
- Lære (enkel) objektorientert tankegang
- Lære å lage enkle Java-klasser og -programmer

Øvingskrav:

- Kunne tegne enkle tilstandsdiagrammer
- Kunne deklarere klasser, med data og kode, iht. oppgavespesifikasjon
- Kunne skrive main-metoder for å teste objekter
- Kunne bruke standardtyper og -metoder (e.g. toString()-metoden)

NB: Viktig beskjed!

For å få testene og eventuell kode til øvingene lokalt brukes systemet git. I Eclipse kan du klikke på Git --> Pull i menylinja for å hente den nye øvingen ved hjelp av dette. Om du ikke har denne i menylinjen, er det også mulig å høyreklikke på en av prosjektmappene og velge Team --> Pull.

Dette må du gjøre

Oppgavene for denne øvingenskal du lagre i `ovinger/src/stateandbehavior`. Test-filene som kjøres for å versifisere ligger i `ovinger/tests/stateandbehavior`.

Hvis du ikke allerede har gjort det, må du installere Eclipse med det ferdigkonfigurerte oppsettet for TDT4100. Se denne wikisiden.

Du skal velge og gjennomføre minst tre av de følgende oppgavene angående Tilstand og oppførsel.

- Account (Lett)
- Location (Lett)
- Digit (Lett)
- UpOrDownCounter (Medium)
- Rectangle (Vanskelig)
- LineEditor (Vanskelig)
- Stopwatch (Vanskelig)

Oppgavene er merket med en vanskelighetsgrad relativt til hverandre. Det er en god idé å begynne med de lettere oppgavene dersom du ikke er komfortabel med pensum så langt, men det er anbefalt å prøve seg på de vanskeligere oppgavene om du synes de første oppgavene er uproblematiske. Dersom du allerede føler deg trygg på punktene i øvingskravene kan du forsøke å gå rett på de vanskeligere oppgavene. Du er selvfølgelig velkommen til å løse flere oppgaver enn minstekravet, hvilket lurt gjøres med tanke på eksamen og et langt liv som programmerende.

Før du setter i gang kan det vært lurt å lese wiki-siden om Tilstand og oppførsel nøye, samt ta en titt på det tilhørende `Counter`-eksempelet. Forelesningene og tilhørende øvingsforelesning er selvsagt også lure å få med seg

Det finnes også masse ressurser på wikien om hvordan ulike metoder skal fungere. F.eks `toString`-metoden og metoder for teksthåndtering. Naviger deg litt rundt om du lurer på noe.

Hjelp/Mistanke om bugs

Ved spørsmål eller behov for hjelp konsulter studassen din i saltiden hans / hennes. Du kan også oppsøke andre studasser på sal eller legge ut et innlegg på Piazza.

Godkjenning

Last opp kildekode på Blackboard innen den angitte innleveringsfristen. Innlevert kode skal demonstreres for stud.ass innen én uke etter innleveringsfrist. Se for øvrig Blackboard-sidene for informasjon rundt organisering av øvingsopplegget og det tilhørende øvingsreglementet.

Appendix D.2 Task

Tilstand og oppførsel – Account

Oppgaven handler om en `Account`-klasse, som håndterer data i en konto. Tilstanden i `Account`-objekter er som følger:

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

`Account`-klassen har fem metoder, med følgende oppførsel:

- `deposit(double)` - øker konto-beløpet med den angitte argument-verdien (et desimaltall), men kun dersom det er positivt
- `addInterest()` - beregner renta og legger det til konto-beløpet
- `getBalance()` - returnerer beløpet som er på kontoen.
- `getInterestRate()` - returnerer rentefoten
- `setInterestRate(double)` - oppdaterer renten til å være den nye verdien

Del 1 - Tilstandsdiagram

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

Del 2 - Java-kode

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

Lag en passende `toString()`-metode og en `main`-metode, slik at du kan sjekke at oppførselen stemmer med tilstandsdiagrammet (bruk samme start-tilstand og sekvens av kall)

Testkode for denne oppgaven finner du her: tests/stateandbehavior/AccountTest.java. Original-koden (jextest) finner du her: tests/stateandbehavior/Account.jextest.

Appendix E Gitpod setup

Appendix E.1 Gitpod.yaml

```
image:
  file: .gitpod.dockerfile

tasks:
  - init: sdk use java 12.0.2.j9-adpt
    command: bash command.sh

vscode:
  extensions:
    - vscjava.vscode-java-test@0.22.2:ZBznuSuKUJo5X8dZe6b9Nw==
```

Appendix E.2 Gitpod dockerfile

```
FROM gitpod/workspace-full-vnc

USER gitpod

#
# Installs hub CLI for managing github activity
#
RUN sudo apt-get update
RUN sudo apt-get install hub -y

#
#   Installs sdkman, and Java 12
#
RUN bash -c ". /home/gitpod/.sdkman/bin/sdkman-init.sh \
    && sdk install java 12.0.2.j9-adpt \
    && sdk default java 12.0.2.j9-adpt"
```

Appendix E.3 Task definitions

```
{
        "version": "2.0.0",
        "tasks": [
        {
```

```
            "label": "Hent oppdateringer (ny uke, nye øvinger)",
            "type": "shell",
            "command": "git pull origin master"
        }
        ]
}
```

Appendix E.4 Command bash scripts

```
#
# Removes any upstream branch
#
git remote rm upstream

#
# Adding upstream branch for lecturer/teacher main repository.
# works if and only if a correct maintainer opens the repository
# (github organizations, team-members)
#
git remote add upstream https://github.com/jorgensta/gitpod-test.git
```

Appendix F Interview codes

Code	Files	References
Main impression of Gitpod	9	11
What was it like to program?	8	12
Git experience	13	18
Git impression	1	3
past IDEs	12	34
Surprised by lack of IDE knowledge	1	2
New IDE's all the time	1	1
Jyputer notebook experience	2	5
General IDE knowledge	3	4
General comments on IDE	9	12
Challenges with IDEs	2	2
Positive outlook on IDEs	2	5
Experience with Gitpod	1	1
Intuitiveness with Gitpod	3	3
Impressions with Gitpod	11	20
Gitpod vs Eclipse comparison	3	3
What was it like to program with Gitpod	5	5
Positive attitude towards Gitpod	10	26
Negative attitude towards Gitpod	4	6
Recognising possibilities with Gitpod	9	19
Challenges with Gitpod	1	2
Skepticism with Gitpod	6	8
GUI comments	3	4
Teaching suggestions with Gitpod	3	5
Understanding the Gitpod flow	3	5

Code	Files	References
Preferences in general	2	4
Preferences	4	7
Adaptation (learning)	1	1
Start phases are critical	1	1
Problem solving	1	1
Position to Git	1	1
Working conditions	1	4
Knowledge and prerequisites	1	4
Job relevance	1	5
How to learn Gitpod	1	2
Help from an IDE	1	1
Hard facts	5	6
Lectures	6	10
Course relevance	1	1
Course focus	2	3
Learning activity examples	1	2
Ephiphany	1	1
Favorite programming language	11	11
Favorite drink	13	13
Wishes to explore	1	1
problems with assistants	1	1
Challanges with OOB	1	1
Challenges with OOB	3	6
OOB in general	1	1
OOB help	1	1
Course improvements	1	2
Teaching	1	1
organization of the course	1	1
Testculture	2	4

Code	Files	References
Experience with Gitpod	10	22
Starting phases with Eclipse	1	2
There are better tools than Eclipse	1	1
Challenges with Eclipse	6	18
Positive towards Eclipse	6	9
Straight out negative about Eclipse	5	8
Hindrance for teaching with Eclipse	1	1
Experiences with Eclipse	11	24
Past experienced (broadly)	9	19
programming basis	2	2
early background	2	2
shortcut preferences	1	2
positive towards current course design	4	5
Assingments distribution and diversification	1	2
Feelings of mastery	1	1
Current course structure	9	21
application assignments	1	3

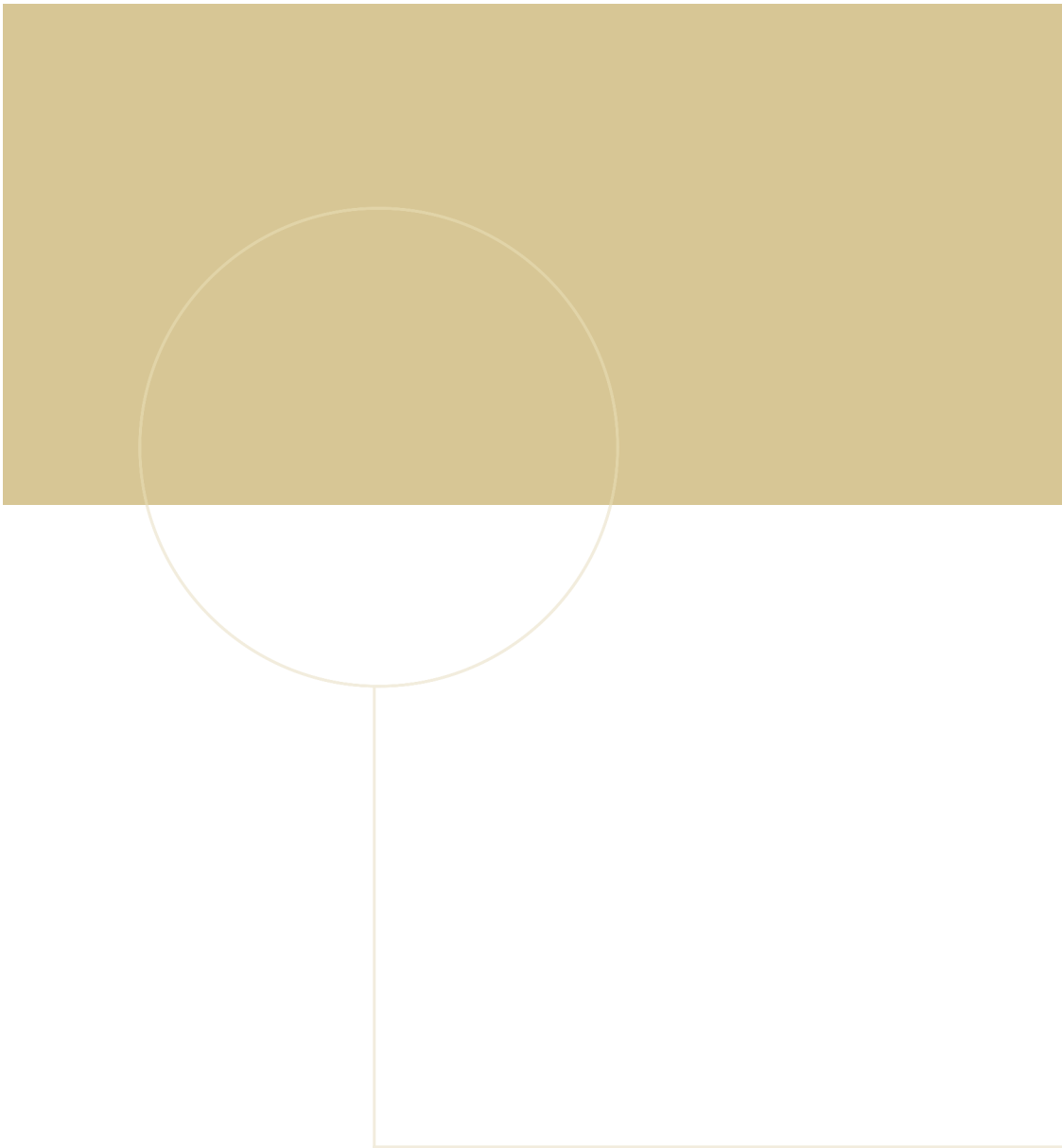
Appendix G Observation codes

Code	Files	References
Eclipse automation that surprises	6	7
Good workflow	9	10
Reading errors well	1	1
Using test feedback	1	1
Effective use of auto-completion	2	2
shortcut preferences	1	1
Questions about auto generation	6	6
Questions about shortcuts	2	2
Arithmetic that fail	2	2
Nervousness statements	2	2
Do not read tasks definitions well enough	1	1
Ignoring errors	4	7
Class definitions uncertainty	3	3
File type uncertainty	3	5
Folder structure uncertainty	2	2
Package system uncertainty	6	8
Programming difficulties	1	1

Appendix H Observation schema

The observational schema was used to capture ongoing events during the interview. Timestamps where captured and written, what happened during that timestamp, where in the program did the event occur and a description of why the event did occur.

When did it happen	What happened	Where did it happen	Why did it happen



NTNU

Norwegian University of
Science and Technology