

Åsmund Hauge

# Git in an educational context

Master's thesis in Computer Science

Supervisor: Trond Aalberg

July 2021



Åsmund Hauge

# Git in an educational context



Master's thesis in Computer Science  
Supervisor: Trond Aalberg  
July 2021

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





## Abstract

Git is frequently used in computer science education as a tool to support project-based development. Introducing students to Git has benefits not only for collaboration and software development but also for collaborative learning.

This thesis explores the implications on group dynamics and the social aspect of project work when using Git. In addition, a prototype mirroring tool using GitLab data was developed to investigate the data's potential to inform students and educators about aspects of the group work. Ninety-one students answered an online questionnaire about using Git, and semi-structured interviews and demonstrations were conducted with 24 students and 11 educators to collect data on the mirroring tool.

The thesis makes two main contributions. Firstly, the analysis of questionnaire responses implies that Git has an effect on some students' experience with group projects regarding interpersonal dynamics and the social aspect of working together. However, it also verifies previous findings that Git has many benefits for student collaboration. Secondly, results from testing the mirroring tools suggest that GitLab data is well-suited to provide students new insight into their work and educators with an efficient method for monitoring project work and student groups.

This thesis's results can impact how educators teach and make use of Git in their courses and for future research and development on mirroring tools using GitLab data.

## Sammendrag

Git brukes ofte i informatikkutdanning som et verktøy for å støtte prosjektbasert utvikling. Å introdusere studenter til Git har fordeler ikke bare for samarbeid og programvareutvikling, men også for samarbeidslæring.

Denne oppgaven utforsker hvilke implikasjoner Git har på gruppedynamikk og det sosiale aspektet ved prosjektarbeid. I tillegg har en prototype av et "mirroring tool" som bruker data fra GitLab blitt utviklet for å utforske dataens potensial til å informere studenter og lærere om aspekter ved gruppearbeidet. 91 studenter svarte på en spørreundersøkelse om bruk av Git og semistrukturerte intervjuer og demonstrasjoner ble holdt med 24 studenter og 11 lærere for å samle data om prototypen av et mirroring tool.

Opgaven gjør to hovedbidrag. For det første impliserer analysen av svar på spørreundersøkelsen at Git har en innvirkning på noen studenter's opplevelse med gruppeprosjekter med tanke på mellommenneskelig dynamikk og det sosiale aspektet ved samarbeid. Imidlertid verifiserer oppgaven også tidligere funn om Git's mange fordeler for studentsamarbeid. For det andre antyder resultatene fra testing av prototypen at GitLab data er velegnet til å gi studenter ny innsikt i sitt arbeid og lærere med en effektiv metode for å overvåke prosjektarbeid og studentgrupper.

Denne oppgavens resultater kan påvirke hvordan lærere underviser og bruker Git i sine kurs og for fremtidig forskning og utvikling av et mirroring tool som bruker data fra Git.

## Preface

This thesis has been written and submitted as the finale of my MSc in Computer Science at the Norwegian University of Science and Technology (NTNU).

I want to extend my appreciation to Trond Aalberg for his support, guidance and patience. Without his help I would not be able to complete my thesis.

Furthermore, I want to thank all students and educators who partook in interviews and tested my software solution.

*Åsmund Haugse  
Trondheim, 2021*

# Contents

<b>Contents</b>	<b>4</b>
<b>List of Figures</b>	<b>6</b>
<b>List of Tables</b>	<b>8</b>
<b>1 Introduction</b>	<b>9</b>
<b>2 Background and related work</b>	<b>12</b>
2.1 Git and GitLab . . . . .	12
2.2 Git in software development education . . . . .	18
2.3 Collaborative learning in computer science education . . . . .	22
2.4 Peer assessment . . . . .	24
2.5 Learning analytics . . . . .	26
2.6 Monitoring in computer-supported collaborative learning . . . . .	28
2.7 Mirroring tools . . . . .	30
<b>3 Methodology</b>	<b>33</b>
3.1 Case studies . . . . .	33
3.1.1 TDT4140 Software Engineering (Spring 2021) . . . . .	36
3.1.2 IT2810 Web Development (Autumn 2020) . . . . .	37
3.1.3 Comparing the cases . . . . .	39
3.2 Online questionnaire . . . . .	40
3.3 GitLab mirroring tool . . . . .	42
3.3.1 Design and creation . . . . .	42
3.3.2 Interviews . . . . .	44
<b>4 Dashboard design and implementation</b>	<b>48</b>
4.1 GitLab API and data collection . . . . .	48
4.1.1 Limitations and workarounds . . . . .	51
4.2 Software technology used . . . . .	52
4.3 Dashboard components . . . . .	54
4.3.1 Group selection . . . . .	54



4.3.2	Project period’s issues and merge requests . . . . .	55
4.3.3	Project period’s commits and lines of code . . . . .	56
4.3.4	Commit and code line distribution by members . . . . .	59
4.3.5	Project commits list . . . . .	60
<b>5</b>	<b>Results</b>	<b>61</b>
5.1	Questionnaire results . . . . .	61
5.2	Mirroring tool results . . . . .	71
5.2.1	Qualitative attribute assumptions . . . . .	71
5.2.2	Component feedback . . . . .	76
5.2.3	Accuracy of visualized GitLab data points . . . . .	82
5.2.4	As a mirroring tool promoting self-reflection . . . . .	83
5.2.5	Anonymity of presented data . . . . .	85
5.2.6	Would educators use the dashboard? . . . . .	87
5.2.7	New features suggested . . . . .	88
<b>6</b>	<b>Discussion</b>	<b>92</b>
6.1	What are students’ perceptions on and experiences with using Git in an educational setting (RQ1)? . . . . .	92
6.2	How can GitLab’s data points be visualized in a mirroring tool (RQ2)? . . . . .	96
6.3	What does a mirroring tool on GitLab data offer (RQ3)? . . . . .	99
6.4	Ethical concerns . . . . .	105
6.5	Validity of results . . . . .	105
<b>7</b>	<b>Conclusion and future work</b>	<b>107</b>
	<b>References</b>	<b>110</b>
<b>A</b>	<b>Appendices</b>	<b>116</b>
A.1	NSD application and confirmation . . . . .	117
A.2	Interview guide educators . . . . .	121
A.3	Interview guide TDT4140 . . . . .	123
A.4	Interview guide IT2810 . . . . .	125
A.5	Results questionnaire IT2810 . . . . .	127
A.6	Results questionnaire TDT4140 . . . . .	134

# List of Figures

2.1	Screenshot of GitLab’s issue list. . . . .	16
2.2	Screenshot of GitLab board for tracking issues. . . . .	17
2.3	Commit history of a project following the feature branch workflow [6]. . . . .	18
2.4	Commit history of a project following the Gitflow workflow [7]. . . . .	18
2.5	The Collaboration Management Cycle as proposed by Soller et al. [43]. . . . .	29
4.1	How courses can be structured in GitLab. A real course would have more student teams. . . . .	49
4.2	GitLab’s analytics displaying commits per weekday and day of month of a master branch. . . . .	50
4.3	Page for selecting what student group to view. . . . .	54
4.4	Project list with statistics and indicators to help educators choose groups to inspect. . . . .	55
4.5	Line charts displaying a project period’s merge requests and issues. . . . .	56
4.6	Bar chart displaying a project’s commits distributed by dates. . . . .	56
4.7	Bar chart displaying team members’ commits distributed by weekdays with columns per member. . . . .	57
4.8	Bar chart displaying team members’ commits distributed by weekdays with combined columns. . . . .	57
4.9	Bar chart displaying a project’s code lines distributed by dates. . . . .	58
4.10	Bar chart displaying a project’s code lines distributed by dates but suppressed to a commit size of 1000 code lines. . . . .	58
4.11	Doughnut graphs showing distribution of commits and changes to code lines by members. . . . .	59
4.12	Doughnut graph showing distribution of programming languages used in the repository. The right doughnut has one entry removed. . . . .	59
4.13	List view of commits of a project. Can be ordered by date, author, amount of code lines removed or added. . . . .	60
5.1	Composition of answers on why students use Git. . . . .	62
5.2	Student answers on learning Git and ease of use. . . . .	63
5.3	Student answers on what Git features their group used. . . . .	64
5.4	Student answers on using Git in the development process. . . . .	65

5.5	Student answers on Git's transparency and on giving and receiving feedback.	67
5.6	Statements on working with GitLab and Git with an interpersonal topic.	69
6.1	Bar chart and doughnut graph both displaying team members' commits.	99

# List of Tables

3.1	Comparison of TDT4140 and IT2810. . . . .	39
3.2	Qualitative attributes used to categorize student groups before and after interviews. . . . .	45
5.1	Juxtaposition of student answers on previous experiences with Git and web development. . . . .	62
5.2	An overview of the interviews held to demonstrate and test the dashboard. It shows the distribution of interviewees, the average interview length and the course of action for each interview. . . . .	71
5.3	Results of attributing student groups qualitative attributes. For the columns "Difference attributions" and "Difference not attributed", green cells means the attribute was correctly assumed more times than not. Red cells mean the opposite. Accuracies larger than 60% are marked in green, below 60% are marked in red. . . . .	72
5.4	Feature suggestions made by educators and students to improve the dashboard. . . . .	89

# Chapter 1

## Introduction

The version control system Git has become a staple of software development in teams, offering functionality to improve working asynchronously and distributed. Widely used, the industry expects graduating students to know it. There have been several studies on the use of Git in the classroom and its benefits, disadvantages, and challenges related to incorporating it in software development courses. For example, Feliciano, Storey, and Zagalsky [1] found that the use of GitHub in software engineering courses was beneficial to and well-received by most of the students enrolled in the course. Furthermore, Hsing and Gennarelli [2] found that students who used GitHub in the classroom felt a greater sense of belonging in the field than those who did not use GitHub in the classroom, suggesting that Git can provide benefits to the quality of software development education beyond its technical capabilities.

Although using Git in software education has seen positive results, little research has examined the adverse effects or challenges using Git in software development education has on students. The tool is complex and often introduced in courses where many students still find programming confusing or complicated. Furthermore, development with Git is transparent and quantifies contributions by students in a way few have experienced before. This thesis wants to examine how students experience using and learning Git to identify the negative aspects of introducing Git in software development education. Furthermore, this research will examine if using Git affects student groups' social aspect and interpersonal dynamic.

Another side of using Git is that students generate data points during development. This data describes work done by student groups and is accessible to both students and educators whether they use GitLab or GitHub. However, most students pay little attention to the history of their repository, and educators do not have enough resources to inspect all student repositories. This thesis attempts to use data points from Git and GitLab in a mirroring tool to assist both students and educators. For example, can the data give students more insight into work habits or team cooperation? For an educator grading a group's project, can this data give more

insight into their performance and be used for evaluation? Or perhaps, can it be used to more rapidly assess the health of a course and identify student groups struggling?

This thesis tackles Git and GitLab in software development courses from two angles. First, questionnaires provide insight into students' experiences using Git and GitLab, and how Git and GitLab affect students' motivation and the social composition of student groups. The second focus is on a mirroring tool using data from GitLab to provide students and educators with a more intuitive and efficient presentation of a GitLab repository to increase reflection and understanding. Through interviews and demonstrations, the dashboard was presented to and tested by both students and educators.

The research questions the thesis hopes to answer are:

- RQ1: What are students' perceptions on and experiences with using Git in an educational setting?
- RQ2: How can GitLab's data points be visualized in a mirroring tool?
- RQ3: What does a mirroring tool on GitLab data offer?

## **Motivation and approach**

This thesis and the work done was initially only defined by the term "Learning Technology", defined as *software and other technological products that support learning and education*. In collaboration with my supervisor, Trond Aalberg, the focus of this thesis was shaped iteratively through discussions on what was feasible and what research was interesting. Mainly the focus has been on the two subjects below.

- Student attitudes towards and experiences with using Git with GitLab in group projects.
- Presenting students' data points from GitLab in a mirroring tool.

My motivation for researching student's attitudes on Git and GitLab originated in my own experiences using the technologies. Git is a technology most computer science students use during their education, often without a proper introduction. I've spoken to students who describe the process as similar to birds learning to fly, having some powerful technology forced upon you; many learn to use Git but do not understand how it works. Curious to learn how students experience this process, the thesis questions its impact on student groups' motivation, interaction among members, and social aspects.

This work has involved researching related work using Git in an educational setting and formulating three research questions. To best answer the research questions, a questionnaire on experiences and attitudes towards using Git and GitLab. Forty-eight students enrolled in a web development course and 43 students enrolled in a software development course answered the questionnaire, and their answers will be in this thesis.

Another important factor for how this thesis turned out was that I wanted to make something. Many courses at NTNU use GitLab for software development, collecting data from students in one place. However, students use Git differently, and its data points do not retain all information of the development process. Intrigued by the idea of making sense of "*stupid*" data to improve learning, the idea of developing a mirroring tool arose.

# Chapter 2

## Background and related work

### 2.1 Git and GitLab

#### Version control

Version control is a term describing a system that maintains records of changes to a set of files, allowing its users to access specific versions at a later time [3]. These systems track any type of file, from images, code, presentations, and data. One of its main benefits is the ability to revert one or more files to their previous state, compare changes over time, and track who has modified a file. This section will explain VCS concerning developers.

We mainly have three version control systems (VCS): Local, centralized, and distributed. Local VCS is, as the name suggests, version control on a local unit, for instance, an individual's computer. Used to maintain a record of one's work for oneself, not tailored for collaboration with others, local VCS is less used than centralized and distributed.

Centralized and distributed VCS provide functionality for developers to collaborate between different systems and computers. Centralized VCS, such as Subversion and CVS, use a single server to track files, which developers then check out and modify. Although advantageous to a local VCS, it has the downside of being a single point of failure. If the centralized server experiences downtime, its developers cannot access its files during the downtime.

Distributed CVS (DVCS) differs in that developers not only check out the files they are working on but check out a mirror of the entire repository. Thus, all developers working on a repository maintain a backup, possibly reducing the effect of a server hosting the repository dying. The most popular DVCS are Git and Mercurial; we explain Git more in-depth.



## Git

Today, Git is the most popular VCS used for software development. It is open-source, free to use, fast, well-suited for handling large projects and non-linear development. Created in 2005 by Linus Torvalds, the same person widely recognized for his work on the Linux kernel, its development started as a response to the poor performance of the VCS available at the time.

Git maintains a record of changes by taking snapshots of each file in a repository, represented by a *blob*. Git hashes a blob's content using SHA-1, used to compare files. Blobs make up *trees*, the equivalent of folders, whose hash is computed based on its content. The genius of Git stems from its use of snapshots to compare files and reduce required storage. As opposed to VCS that store copies of unchanged files for each version, Git will instead supply a link to the hash pointing to the last time a file has changed, meaning Git uses a stream of snapshots to present the repository at a given time.

The "timeline" of a Git repository is a graph structure of *commit* objects. Each commit contains a pointer to the snapshot of the changed content, some developer metadata, a commit message, and a pointer to its parent commit or commits. The initial commit has no parent, and the next will point to the initial commit, and so on.

Git only maintains a record of files when told to do so. Each file residing in one of three states: *Modified*, *Committed* or *Staged*. Committed means a snapshot of the file is stored locally, modified means the file has changed, but a snapshot of the change is not stored yet, and staged means a file has been modified and marked for inclusion in the next commit object. This is better understood by explaining the stages a file can be in: *Working Directory*, *Staging Area* or *Local Repository*. If a repository is stored remotely, for instance, using a service like GitLab or GitHub, a fourth stage, *Remote Repository* is also possible.

Git uses the Local Repository to store information about a repository. The Working Directory is a local representation of all files at a single version of a repository, a single checkout of a commit. Git fetches versioned files from the Local Repository. Files changed in the Working Directory are moved to the Staging Area in the shape of a snapshot. Snapshots contain the changes included in the next commit. Using a remote repository hosted using a service such as GitLab, pushing commits updates the state of changed files. Doing so makes the commits available to see for all developers connected to it.

To use Git efficiently, students must know the following commands:

- **git status**
  - Example: *git status*
  - Lists information about your repository, such as the state of files modified since the last commit, what branch you are currently on and if your branch is up to date with the remote repository.
- **git clone**
  - Example: *git clone git@github.com:torvalds/Linux.git*
  - Clones the repository, here called Linux, into a newly generated directory of the same name.
- **git add**
  - Example: *git add userController.js storeController.js*
  - Moves the files *userController.js* and *storeController.js* from your working directory to the staging area.
- **git commit**
  - Example: *git commit -m "Fix sorting bug"*
  - Creates a new commit object, moving the files in the staging area to the local repository.
- **git push**
  - Example: *git push*
  - Pushes commits made locally to a remote repository, for instance, one hosted on GitLab.
- **git pull**
  - Example: *git pull*
  - Fetches new commits from a remote repository and integrates these with your local repository.
- **git branch**
  - Example: *git branch "feature/static-analysis"*
  - Creates a new branch named *feature/static-analysis*, which can then be checked out using *git checkout*. Commits made on the new branch will not be applied to the origin branch.
- **git checkout**

- Example: *git checkout "feature/static-analysis"*
  - Checks out the branch "feature/static-analysis," setting your HEAD to the last commit of that branch. The command can also check out specific commits.
- **git merge**
- Example: *git merge "feature/static-analysis"*
  - Merges the current branch with the branch *feature/static-analysis*, creating a new commit object. The parent commits of this commit will be the latest commit of both the current branch and *feature/static-analysis*.

When using Git, developers choose between the command-line interface (CLI) or a graphical user interface (GUI), such as GitKraken or SourceTree. Both options perform the same but provide an interface to Git's functionality differently, the CLI utilizing commands and flags whereas the GUI uses buttons, text boxes, and other graphical elements. Some developers argue for one interface over the other, stating that, for instance, the CLI is better suited for beginners because it does not present its user with more functionality than they seek. Others state that the GUI is superior because it visualizes how Git works.

## GitLab

GitLab is a web-based tool providing its users with free hosting of Git repositories, issue-tracking, and tools to support developers with DevOps. Created by two Ukrainian developers, Dmitriy Zaporozhets and Valery Sizov, it has grown in popularity since its launch in 2014. It is now the second most popular service of its kind behind GitHub.

GitLab provides its users with some functionality to help its users navigate and perform actions on their repositories, some of which include the ability to view the commit history of a repository, its branches, and files, as well as its *merge request* functionality.

Merge requests provide developers with a three-step approach to Git's *git merge*. The developer first specifies a source and target branch for the merge request, then creates the merge request. Then, other developers on the team can review the merge request, provide feedback, or comment on the merge request before choosing to approve the merge request. Finally, the developer can perform the merge, which combines the two branches into one.

GitLab also provides developers with the option to create *issues*; descriptions of some job or task. Developers can add templates for specific types of issues like bugs, feature requests, or refactoring to improve the quality of the issues. Furthermore, developers can be assigned to an issue to signal that they are working on it, and labeling an issue helps categorize them. Figure 2.1 lists four issues as presented in GitLab.

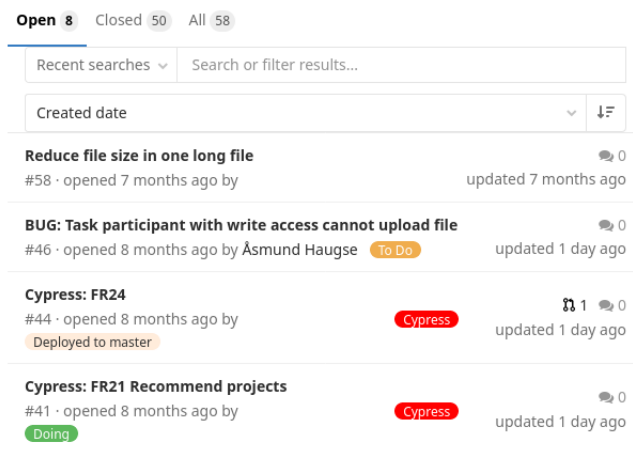
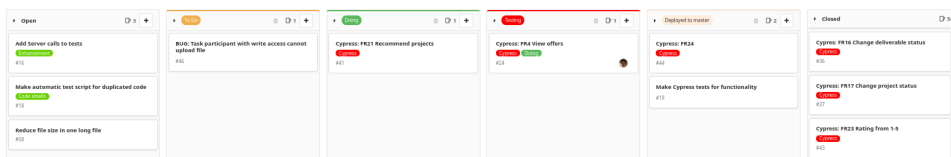


Figure 2.1: Screenshot of GitLab's issue list.



**Figure 2.2:** Screenshot of GitLab board for tracking issues.

Inspired by the physical task board, GitLab has a view for presenting issues in a board format. Issues can be in one column, signaling at what stage of a team's defined workflow the issue is. Columns are added when needed, providing developers with an effective way of seeing the state of their project, as can be seen in Figure 2.2.

## Practices using Git and GitLab

There is no *best* way of collaborating in a team using Git, but we often see some reoccurring practices among developers. This section will introduce a few commonly used practices in the following section.

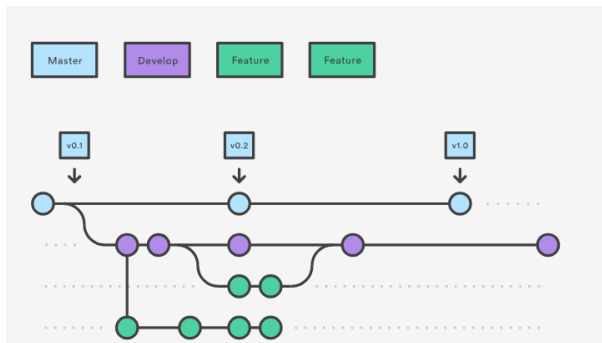
*The master branch*, the primary branch of your repository, should be treated with care, especially for projects with multiple collaborators. A common practice is avoiding or even preventing developers from pushing commits directly to *master*. Using GitLab's merge request feature to introduce changes to *master* gives developers increased knowledge of a repository's codebase and can promote good discussions about its content.

*Commit messages* describe what a commit changes, and are written by developers for all commits. A commonly used rule says that the subject line (The first line of the commit message) should complete the sentence *If applied, this commit will <subject line>* [4]. Use the imperative mood, and the message's length should not exceed 72 characters. Referencing an issue in the commit message, e.g., "Add routing for store #25", links to the related issue in GitLab (Issue #25).

*GitLab's issue boards* provide developers with a tool to plan, organize, and visualize their workflow by pairing issue tracking and project management [5]. Developers can tailor issue boards for specific needs, e.g., a Scrum board, and paying users can have multiple boards linked together. Boards best visualize a team's progress when they are up to date. Thus the best practice is to move the issue accordingly when working on it.

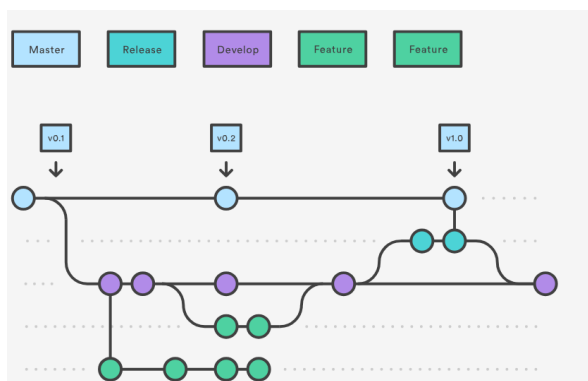
*Branching* is commonly done in a modified fashion of *the feature branch* [6] and *Gitflow* [7] workflow. The former focuses on keeping all development of new features on separate branches, in theory preventing broken code from entering *master*. In short, a developer creates a branch to work on a feature, pushes some commits, and

creates a merge request for the feature. Some developers review the request. Perhaps some contribute to the branch with new commits, some provide feedback, and the request is accepted. Finally, the merge request is accepted, and we merge the feature branch into master. Merging branches create a merge commit.



**Figure 2.3:** Commit history of a project following the feature branch workflow [6].

The *Gitflow workflow* maintains two branches dedicated for a project’s history, the *master* branch and a *development* branch. Extending the feature branch workflow, it is more centered on releases, with features first being merged into *development*, then *development* is branched to a release branch, which is then merged into *master*. If some code on *master* needs to be fixed, a *hotfix* branch is created from *master* and merged with both *development* and *master* when completed.



**Figure 2.4:** Commit history of a project following the Gitflow workflow [7].

## 2.2 Git in software development education

Universities receive pressure from the industry to prepare their students for the work awaiting them and adapt their courses’ accordingly. As a result, the shape,

content, and relevance of a course can vary greatly, and with it, the perceived usefulness of a course varies. With rapid changes in what technologies are relevant in software development, a course whose technology was relevant three years ago may be outdated today. Especially exposed to change is web development, with new libraries, frameworks, and concepts appearing often. However, more than just technology is changing in the field of software development. We have observed a shift towards agile methodologies where students must learn to collaborate in teams. Software development in teams is a skill to master, emphasizing the technical aspects of writing code and the social aspects of working in teams.

Version control systems such as Git have come to stay and remain relevant in the years to come, accommodating distributed software development in teams. Universities teach it in their software development courses early on to give students experience and understanding of how it works. Over time several version control systems have been developed and used in education and the industry, such as Subversion, Mercurial, Git, and Bazaar. However, Git, offering distributed version control and possibly as a result of Git being used to develop the Linux kernel, has grown to be the most popular version control system over the years.

### **Student perspective**

Git is a complex tool that can be difficult to understand, introducing a way of thinking that is not necessarily best taught in introductory software courses. In addition to all other new concepts taught in introductory courses, its complexity may confuse already confused students, even more, especially those who have little to no previous experience with programming. Furthermore, attempting to teach Git without programming may also struggle to present the benefits it provides. Thus, faculties face the challenge of introducing Git early to students to become experienced users before finishing their studies, but not too early to avoid it being a hindrance or element of confusion.

Although we often praise Git for its usefulness for software development in teams, its benefits are also present when working alone. Professors and educators can provide skeleton code for the students to improve upon using integrated version control systems in an IDE such as Eclipse or IntelliJ IDEA. Doing so, students do not need to understand how Git works.

In some courses at NTNU, students link their Eclipse IDE to a remote repository owned by the professor. Then, the professor pushes assignments and the code presented during classes, and students can pull the code to their working environments. Although students have little understanding of what is happening when they pull code from their teacher's repository, they are pleased to have skeleton code from lectures and assignments appearing locally on their computers.

Lawrance, Jung, and Wiseman [8] examined the technical obstacles encountered when computer science faculties attempt to expose students to version control as well as the perceived benefits of the students exposed to Git. Students were often initially confused by Git's inner workings, but they appreciated what it offered and did not feel that learning Git was a waste of time. Notably, by using Git in a class project with multiple contributors, the students learned of the benefits of adopting good habits such as branching by topic and more minor incrementing changes to prevent massive merge conflicts.

Haaranen and Lehtinen [9] suggested using GitHub as an LMS to distribute and collect assignments. Furthermore, they gradually introduced new Git features to students. The reasoning was that students who use Git in a course where they work alone only need a few Git commands to work efficiently. Students can clone each assignment, and depending on the complexity of the course, it introduces different Git functionality. Instructing students to make incremental changes to their software and committing their changes often can help by inducing mass training in adding, committing, and pushing their changes if they are using a remote repository. By instructing students to develop two versions of the same piece of software, each version with a few specific functions implemented differently, they learned Git's branching and merging functionality, which effectively displayed the simplicity of switching between Git branches.

To give students authentic experiences with Git in software development, using Git for assignments where students must work in groups will be advantageous. Arguably the most frustrating aspect of Git for beginners is merge conflicts, occurring when two separate branches have made edits to the same lines of one or multiple files. Although merge conflicts can occur when working alone, they are more common in projects with multiple collaborators. They are even more common in projects where collaborators have little experience with Git.

Feliciano [10] conducted a study of using GitHub in multiple software development courses. Looking at student perceptions of GitHub's benefits and what challenges students encountered, multiple benefits and challenges arose. Among its benefits noted by students was gaining experience with an industry-standard tool. Its transparency in seeing when and how their team members work helped keep each other accountable for the group project. Additionally, students found it helpful to use GitHub as a portfolio for their projects.

All the above benefits apply to GitLab, although students tend to use GitHub over GitLab for their portfolio due to its popularity. In addition, employing Git with GitLab for software development courses can provide multiple benefits to students that they can later apply outside the educational context.



## **Educator perspective**

Incorporating Git into a software course is not as quickly done as it is said. The correct course to introduce Git is not apparent, and one might argue for and against introducing Git in an introductory programming course. Furthermore, Git can hardly fill a course on its own. Not using Git in a relevant context makes it harder to drive home the point of using it. Faculties should be hesitant to replace their existing courses with courses focusing entirely on Git, as developers can come a long way knowing only the most common functionality of Git.

GitLab supports any file format, meaning educators can upload course material such as presentations, illustrations, and more, aggregating course material in one place. In addition, with its merge request feature, students can provide feedback on course material in the form of improvements or corrections, which the teacher can review and accept with little effort. Similarly, students could choose to open an issue requesting more materials or as a discussion forum, depending on how the educator wished to use GitLab.

Kelleher [11] found that students enjoyed using Git knowing that they were using industry-standard technology. Similar to the study of Lawrance, Jung, and Wiseman students encountered technical issues when using Git, but these diminished as the students became more experienced. Students were exposed to the most common functionalities of Git in conjunction with GitHub, such as branching, commits, and remote repositories. Noted benefits were GitHub being free to use, saving faculty resources by not needing to host their repositories, improved quality of submissions (gitignore files aided in leaving out irrelevant files, no duplicate versions of files), and student satisfaction stemming from the use of relevant tools.

GitLab can collect and distribute assignments by enabling educators to upload an assignment to a repository that all students enrolled in a course can fork. Thus, the educator can view all repositories forked for an assignment, gaining more insight into how a student or group of students has worked than they would have received from students only delivering their project files [10]. In addition, GitLab offers a graph view of all commits made to a project. Suppose students are to complete several tasks for an assignment. In that case, the educator can instruct students to tag each task in the commit or merge request message, making it easier for the educator to find their implementation.

Learning management systems such as Blackboard also provides solutions for distributing and collecting assignments, but knowing how to use Blackboard is not relevant to work as a software developer in the industry. Therefore, faculty must decide whether to shape most or all aspects or only some aspects of course education around Git and GitLab. Aiming only to replace some parts of an existing course

requires less work compared to a complete overhaul. However, it may very well still achieve the desired outcome of teaching students relevant technologies if done right [8].

## 2.3 Collaborative learning in computer science education

Software development courses that use Git can benefit from employing project-based learning in teams. Grouping students together naturally increases the capability of what they can achieve, giving educators more leeway in designing course assignments and curriculum, allowing for more open-ended assignments and improved learning [12]. The approach also introduces relevant aspects of software development such as agile methodologies, software system design and implementation, and clean code.

Sindre et al. [13] presented how project-based learning can be implemented in courses throughout IT education. They classified how to apply project-based learning in courses to help educators implement project-based learning in their courses, building on more informed decisions about what works and what does not work and the consequences of their decisions.

*Range of implementation* and *freedom of the deliverable and process* are ingredients educators must consider when designing project assignments of a course. A heterogeneous student group working on larger projects spanning over a semester is advantageous in its realism, often increasing complexity. Stricter guidelines for the project can simplify evaluation of whether criteria are met or not but restrict students in creativity and innovation. More ambiguous guidelines promoting innovation and creativity can, in turn, result in a more significant amount of design decisions being made [13].

Accurate design decisions engage the students in solving a complex problem themselves, being responsible for their learning. Dochy et al. [14] suggests that project-based learning had a positive effect on students' acquisition of knowledge and the skills to apply this knowledge. Larger and more complex problems also proved to increase collaboration and the development of communication skills as problems often proved too difficult for individual effort [15].

Software development in teams requires skills beyond that of technical competence. Teams must communicate remaining work, decision-making as well as communicating the progress of the team [13], [9] [16] [17]. A group of students seldom consists of equally talented software developers, with some students having more experience than others. Ideally, this leads to exchanging knowledge with the more experienced students explaining or giving pointers to the less experienced students. Curşeu and Pluut [18] claims that group diversity is a necessary condition for collaborative

learning and that groups with similar members have reduced opportunity to learn from each other.

Educators can further motivate peer teaching in software development courses by encouraging pair programming. Pair programming is an alternative to solo programming, stemming from the growth extreme programming has seen over the last years. When pair programming, students have to discuss implementation ideas to find a solution to the problem at hand, often both programmers seated in front of the same computer.

McDowell et al. [19] conducted a study investigating the effects pair programming had on student performance and enjoyment working with assignments. The study compared students who enrolled in a Data Structures course, some opting for pair programming and some opting to work alone. Students who chose to pair program had significantly higher passing rates and were more likely to do better in their courses afterward. Another benefit noted by the authors was that the quality of the code produced was better for students who pair programmed, perhaps due to having to meet the expectations of two students before moving on. Furthermore, students who pair programmed reported higher confidence in their program, suggesting that pair programming can lead to more students being satisfied with their education choice.

Returning to the discussion of group diversity, although arguments suggesting it creates increased learning opportunities have been presented [20], diversity is not a guarantee for success. Research on collaboration and group dynamics present several negative scenarios (e.g., internal conflicts, free-loaders, competitive behaviors) that can occur in collaborative work [20]. Curşeu and Pluut [18] thus argues that diversity will have disruptive effects on teamwork, in turn, affecting the positive effect diversity has in collaborative learning.

Methods used for grouping students together for projects vary between faculties, courses, and contexts. Students being allowed to choose whom they work with offers safety to students who choose the same teammates in several courses, having a known team dynamic and knowledge of each other's strengths and weaknesses. However, this reduces the opportunity to work with students with different perspectives, backgrounds, or experiences [21].

Teaming students who have no previous experience working with each other encourages discussion of how to approach the task at hand, structure work, and divide labor through exchanging previous experiences. Repeatedly working with the same students also reduces exposure to different ways of working. Although faculty pairing students together can lead to some groups having internal conflicts or problems, those who have experienced conflicts may have learned how to deal with

the conflict for future group projects [22].

Another concern is slackers or free-riders, group members who do not contribute their part to the group project [22]. Colbeck, Campbell, and Bjorklund [21] found that the potential for slacking increased as team size increased. They reported that students knew what students to avoid based on previous projects. Thus, students would try to avoid teaming up with the slackers. Having a slacker can be detrimental to the learning of all members of the group.

In smaller group projects, this is especially bad, for instance, in a group of three students where only two students contribute, increasing the amount of work, time, and effort they must put in to compensate for the slacker not pulling their weight. Hall and Buzwell [23] concludes that many students' frustration with slackers stems from the slackers often receiving the same mark as those who contributed the most, due to the final product often being evaluated instead of the process leading up to it. The paper suggests employing journals to document how a team works throughout the project period. Educators then review the journals when marking students alongside the final product.

Hendry, Ryan, and Harris [24] discusses problems occurring in problem-based learning. In the paper, both educators and students rank the frequency of problems occurring when working in groups. The third most frequent problem, ranked equally by both students and educators, was that of *the dominant student*, someone who talks a lot and tries to control the direction of the discussion, often preventing others from contributing and getting a word in. Furthermore, students reported that the case of dominant students was difficult to solve and that problems arose because of how the group was working, not because of the task at hand.

## 2.4 Peer assessment

Peer assessment is commonplace when using Git in the form of feedback on merge requests and code. In short, peer assessment is a process where learners themselves are responsible for grading their peers and providing feedback on their work. The use of peer assessment to assess the quality of work produced by students has been studied in a wide range of contexts, such as academic writing [25], oral presentations [26] and software development [27], all of which reported both benefits and problems.

Challenging students to assess the work of their peers promotes their higher cognitive skills by applying one's own skills and knowledge to understand and evaluate the work that has been done [27]. It requires critical reflection, deep thinking, and learning [28], which in turn leads to an improved learning outcome for students.

Peer assessment can aid educators in relieving them of parts of the workload

related to marking students. Over the course of two weeks, a student group has the capability to develop relatively large software systems, each group choosing to complete the assignment differently. A student reviewing the assignments of three other student groups is obviously done quicker than an educator reviewing sixty or more assignments.

Instructing multiple students to review the assignment of one group can help in reducing the effect of some students poorly assessing an assignment. If an educator finds that the reviews of one group's assignment vary greatly in grading, the educator can assess the quality of the assignment themselves to better assess what grading it should be given. Topping [29] found that the quality and mark given when students peer-assessed the work of other students did not vary greatly from the assessment of an educator. Furthermore, the difference in grading was further diminished when several students assessed the same assignment, suggesting that both the validity and reliability of peer assessments are adequate [25].

Students often want their assignments to be reviewed sooner rather than later, and through peer assessment, this is made possible. If students are given a deadline for when they have to have assessed other students' assignments, students know that they will have feedback on their assignments by that time. Furthermore, if each assignment is to be reviewed by three students, this often means that students receive a greater amount of feedback than they would have had only the educator assessed their assignment [25]. The content of each assessment may also vary as a result of what students look for and their opinions, in turn possibly giving students feedback on an increased amount of aspects of their assignment.

Sitthiworachart and Joy [27] discusses the results of peer assessment performed by students in the context of a software development course. For the study, students were tasked with assessing the quality of other students. To further motivate students to perform the assessment more in-depth, the quality of the feedback they provided affected their own grades for the course. Sitthiworachart and Joy [27] attempted to simplify the marking criteria and reducing the number of choices per category. What they found was that students found it difficult to pick the choice best suited for a category as a result of there being too few choices. They suggested using a 5-point Likert scale to provide students with enough options to fairly judge student performance.

Orsmond, Merry, and Reiling [30] reported that students were unenthusiastic about having to define criteria for peer assessment themselves, preferring to grade according to criteria defined by an educator. Providing students with both grading criteria and a guideline for how the criteria should be applied helps with student understanding of how to grade [27]. This may also help in reducing the variety of

student assessments of the same assignment because there is less ambiguity in how students should evaluate different parts of an assignment.

Sitthiworachart and Joy [27] provided students with both a defined list of criteria and guidelines for grading their assessment to help with the process of peer assessments. Even so, it was reported that students had a difficult time assessing the quality of other students' assignments when they themselves did not feel qualified on the subject. Equally, students doubted the quality of assessments given by other students who they felt were not qualified. Some students reported feeling that they received peer assessments that were unfair and inconsistent, but the results of their research suggested that this was not the case. The research of Vu and Dall'Alba [31] also found that students had concerns related to both the fairness and accuracy of the grading provided both by themselves and other students.

Topping et al. [25] and Sluijsmans, Dochy, and Moerkerke [26] found that students who assessed other students in turn improved the quality of their subsequent work. Sitthiworachart and Joy [27] and Sitthiworachart and Joy [32] studied the use of peer assessment in a software development course. When developing software, there is no perfect solution, meaning must apply critical thinking to evaluate its correctness and quality.

Students can learn from seeing a clever implementation of a problem they themselves would have thought to implement differently, plus reviewing code gives insight into more of the software system's code [27]. Closely related to code reviews, a method commonly used in the software industry, peer assessment aids in preparing students for the work awaiting them. Code review is when a developer reviews code written by another developer, inspecting its quality and implementation. Through code reviews, students partake in collaborative and peer teaching, giving pointers, feedback, and correction of each other's code. This has a doubly positive effect, both for the students who review and those whose code is being reviewed. Students whose code is being reviewed receive constructive feedback, pointers, or questions on their implementation, which they can use to improve its quality.

## 2.5 Learning analytics

Using Git in software education is in line with the digitization of the last decade. However, a typical approach is to use Git to develop a project and leave it at that. With the rise of learning analytics, researchers may be able to use Git data from an educational setting. Learning analytics has close ties to business intelligence and data mining and aims to provide more insight. Although widely employed by technological companies and governments, educational systems were comparatively slower to shift towards the analysis of big data [33]. Higher education institutions worldwide use

learning management systems (LMS) like Blackboard, collecting more and more data. Acknowledging the potential of Big Data, the Obama Administration announced their "*Big Data Research and Development Initiative*" in 2012, urging industry, research universities, and non-profits to make use of all opportunities created by Big Data [34].

The Society for Learning Analytics Research (SoLAR) defines learning analytics as: "The measurement, collection, analysis, and reporting of data about learners and their contexts, for the purposes of understanding and optimizing learning and the environments in which it occurs" [33]. Despite being a reasonably young research field, researchers have been generous in defining its potential. Research on the area of LA has, among other topics, inspected how to improve student performances (e.g., knowledge acquisition, cognitive gains, learning outcome, and skill development) and from the perspective of learning support and teaching [35].

Institutions expect that learning analytics can help improve student performances. Arnold and Pistilli [36] piloted the software Course signals (CS), a learning analytics system providing feedback to students based on predictive models. Using data from students, it attempted to determine in real-time what students were at risk to help educators deliver meaningful interventions to improve students' chances of success. Their findings suggested that students enrolled in CS courses saw higher retention rates than those with fewer CS courses. Furthermore, comparing students' grades of a course that implemented CS for one semester with results from the previous semester, their research showed a significant improvement. Other studies also showed improvements in students' performances through the use of learning analytics tools [37] [38] [39] [40].

The iterative process of designing course curriculum and content is traditionally affected by student feedback, changes in the software development field, input from the industry, and more. Thus, the research field of learning design stands intertwined with learning analytics, both impacting each other. Learning design aids educators in designing learning activities, and learning analytics make use of the metrics and data used to inform and influence the design process [41]. Research on learning design has generally focused on supporting teachers in defining pedagogical approaches and educational objectives to make improvements and decisions on courses and reflection.

Robles and Gonzalez-Barahona [42] looked at mining student repositories in a learning analytics context. The paper presents techniques for mining software repositories that are transferrable to an educational environment. Furthermore, they implemented an almost automated solution to gather data from students' programming assignments using Git to use in a learning analytics context. They did this to assess code quality, plagiarism, automated feedback, and the creation of personal exams. Students appreciated receiving automated feedback, and the

code of students improved. In addition, using the system was well-received by the instructors, who saw little need for manual work outside of some inspection and evaluation. However, the authors believe that complete automatization is hardly possible because some fine-tuning will be necessary to adapt to specific courses.

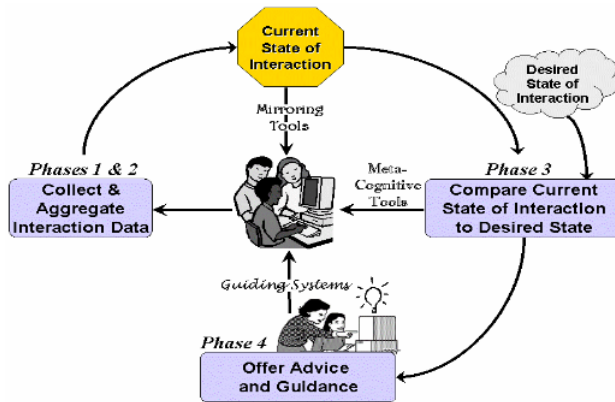
## 2.6 Monitoring in computer-supported collaborative learning

The increase in data points generated by students in learning situations introduces possibilities to monitor more aspects of student groups. For example, some data is unambiguous (e.g., timestamp of a deliverable or downloading a resource). In contrast, other data leaves more room for interpretation if viewed by itself (e.g., timestamp and duration of a Zoom-meeting). Thus, monitoring may assist educators and students in providing insights into student groups and detecting good and bad collaboration patterns.

Managing collaboration can also prove easier with increased opportunities to monitor activity. For example, in the form of indicators displaying student groups' performance, educators can make information-based decisions on what groups to contact. Likewise, offering students relevant resources based on their interactions with the system can help them move past a problem.

Soller et al. [43] presents a framework for describing the process of collaboration management to help define a model of desired interaction and monitoring current interaction (see Figure 2.5) which I will present. The first phase is the collection of interaction data. Whether analysis on the model will be activity-based or state-based is an important decision bearing implications on how and what format to record data points. Activity-based analysis requires historical logs of user interaction over time, whereas state-based analysis requires logging of snapshots of interaction without history information [44]. Transforming said logs into holistic and useful information is difficult, time and resource-demanding, often requiring human manipulation of data sets [33].





**Figure 2.5:** The Collaboration Management Cycle as proposed by Soller et al. [43].

The second phase is to construct a model of interaction, meaning defining how to represent the current state of interaction based on high-level variables and metrics. In this phase, one should consider the needs of educators and students, perhaps separately, to best provide an efficient and unambiguous state representation.

The third phase compares the current state of interaction to the desired state, used to classify attributes of the interaction based on the metrics and variables defined in phase two. Soller et al. defines the desired model as a set of indicator values that differentiate between productive and unproductive interaction states. In turn, the definition of a productive interaction state will vary from domain to domain. For example, in software development courses, productive interaction states can be similar work distribution and a steady stream of completed story points/issues as a team.

Phase four is to advise or guide the interaction. If the current and desired interaction does not match, the system can suggest mitigating actions to help move towards the desired interaction state. Less complex suggestions use few metrics (e.g., changes in work distribution based on story points completed). In contrast, more complex actions require more complex analysis (e.g., suggestions on resources to improve development speed when using specific JavaScript libraries). The actions may be suggested by the system or by humans depending on implementation.

Before returning to phase one of the collaboration management cycle, we pass through a fifth phase where we evaluate the action suggested in phase four and the interaction of the cycle. Both humans or the system can evaluate to improve its ability to present the interaction state. If necessary, the desired state is changed to meet the learning goals better.

## The locus of processing

The *locus of processing* describes where decisions on the quality of student interaction are made and how to facilitate interaction [43]. The locus of processing can lie with students, educators, or the system, or a mix of the three, depending on the goals and requirements of the learning activity. Depending on where the locus of processing lies, we distinguish between three types of computer-based support options, *mirroring tools*, *metacognitive tools*, and *guiding systems* and vary in what phases of the collaboration management cycle they perform.

Mirroring tools describe systems that collect and aggregate data about students and reflect the information to students or educators. They perform only phases 1 and 2 of the collaboration management cycle and leave the locus of processing entirely with the user. In other words, users must define what is the desired state of interaction and decide how to act themselves. Designed to improve self-reflection and awareness of students' actions, visual presentations of behavior and information give an efficient and intuitive representation of the state of interaction. Mirroring tools will be explained more in Section 2.7.

Metacognitive tools display an indication of the desired state of interaction and the current state of interaction. It performs phases 1, 2, and 3 of the collaboration management cycle and can be considered a superset of the mirroring tool. These systems provide users with indicators necessary to diagnose the interaction, meaning the locus of processing also lies with the system. However, the user is still responsible for choosing how to act on the indicators and the diagnoses.

Guiding systems perform all the phases of the collaboration management cycle and help users by proposing how to act on deviations from the desired state of interaction. Often not displayed to the students, the desired model of interaction and the system's assessment of the students are used by the system to help moderate their interaction.

Designing a system to support student interaction involves evaluating the needs of both students and educators and available computational resources. Based on gathering information and presenting it in accordance with a model, the three systems are similar. Their difference lies in the location of the locus of processing: the system, educators, students, or some mix of all.

## 2.7 Mirroring tools

The following chapter will go more in-depth on research done on mirroring tools. To recap: mirroring tools describe systems that collect and aggregate data points from a user's interaction and reflect this information to the users. Typically, students in a

learning situation generate data points, which an educator or the students reflect on through graphs, tables, and other visualizations. Aiming only to present data more efficiently, the locus of processing lies with its users, who must attempt to draw conclusions on the information and decide how to remedy deviations in the interaction model. The same information can be helpful in different ways depending on whether a teacher or a student uses it. For students, these systems aim to enhance self-awareness of one's actions and behavior [45] [43] to improve upon for future work. Educators can use mirroring tools to gain insight into how students work and to know what students or student groups need guidance.

We can think of undesirable situations in group projects (e.g., free-riders, varying participation, gaps in actual and expected skill) and their desirable opposites. Quantifying participation and contributions to a group project may be helpful to provide insight into how much and when team members have contributed. Ideally, this would indicate equal participation among members, but perhaps just as practical, it may indicate that some members have participated far less. Dietsch et al. found that a mirroring tool for student activity in a collaborative software development setting helped identify reoccurring roles in student groups. For example, some groups had a lead coder (who contributed the most lines of code); others had one or two students in the role of designers. Furthermore, they identified students in the role of free-riders as students who were responsible for less than 10% of the code.

A primary goal of mirroring tools is to increase self-awareness and behavioral regulation. Self-awareness, in general, has numerous proven benefits in multiple domains. For example, athletes wear heart rate monitors to improve their workouts, and fitness watches can give feedback on sleep quality. The ability to quantify aspects of our lives improves self-reflection and helps us realize bad habits in our behavior [46]. In uncovering the causes of problematic behavior, self-monitoring tools have been shown practical and valuable to both monitor and maintain changes in behavior. In their 2013 study on behavioral change from presenting statistical patterns, Bentley et al. observed that some participants pieced together two or more of their data representations to understand their habits and patterns better. Participants enjoyed being presented with their data and found it helpful to draw their correlations, which for most of us can be more accurate than assumptions and correlations presented by someone they do not know. Interestingly, this could indicate that data points from a group can be helpful in different ways depending on which group member views it.

Meyer et al. [48] performed a study on software developers in the workplace where they looked at what they expected from a mirroring tool and implemented this. They concluded that increasing software developers' self-awareness about productivity through self-monitoring improved both productivity and engagement. Furthermore, 40.7% of their participants stated that the increased self-awareness motivated them

to adapt their behavior. Their system mainly looked at users' unique collaboration and communication patterns through data on meetings, emails, instant messaging, and code review behavior. Still, they noted that improved insights into how teams coordinate and communicate could help both developers and managers, for instance, with scheduling meetings at more optimal times.

Mirroring tools designed for educators are referred to as *teacher dashboards* and are visual displays that provide analytics of learners. The usefulness of teacher dashboards comes down to how efficiently and effectively they convey information to a teacher [49]. Dashboards should convey what it presents and how it has been aggregated data and should not confuse its user. Furthermore, to prevent confusion, dashboards must be designed with caution to avoid displaying too much information at once. Mazza and Dimitrova [50] notes that systems should be designed differently depending on their users, especially if their users are not well-versed with computers.

Research on the usefulness of teacher dashboards has shown that they can be both helpful and insightful to teachers and that they can be used to give teachers more information on their students' activity [51]. However, research on mirroring dashboards also shows that they do not consistently improve the detection accuracy of a teacher [50] [52] concerning student groups. Teachers cannot always use a dashboard to their advantage, for instance, because the information is overwhelming or because data is interpreted wrong. Concerning teachers, mirroring dashboards, in turn, introduce yet another source of information to consider. The information displayed in the dashboard is possibly already accessible to the teacher and should provide new or improved insight. Failing to do this risks the system being an obstacle instead of a tool to its users.

# Chapter 3

## Methodology

### 3.1 Case studies

This thesis performed two case studies for data generation and analysis. The case study bases itself on Oates, who defines a case study as follows;

*A case study is an empirical inquiry that investigates a contemporary phenomenon within its real-life context, especially when the boundaries between phenomenon and context are not clearly evident.*

Furthermore, the following traits characterize case studies:

- Focus on depth rather than breadth, i.e., the researcher obtains as much information as possible about an instance of the investigated phenomenon.
- Natural setting, i.e., the case existed before the study came.
- Holistic study, i.e., the focus is more on the complexity of relationships and processes and how they are interconnected than isolated individual factors.
- Number of sources, i.e., the researcher uses a wide range of data sources.

Oates [53] defines five choices one must make when performing a case study. They are:

1. The type of case study
2. Approach to time
3. Approach to selecting cases
4. Approach to generalizations
5. Selection of data generation methods

This section will present the choices made and the rationale behind them for this study.

## The type of case study

Three types of case studies are defined, namely:

- **Exploratory study:** "Used to define the questions or hypotheses to be used in a subsequent study. It is used to help a researcher understand a research problem."
- **Descriptive study:** "Leads to a rich, detailed analysis of a particular phenomenon and its context. The analysis tells a story, including discussion of what occurred and how different people perceive what occurred."
- **Explanatory study:** "Goes further than a descriptive study in trying to explain why events happened as they did or particular outcomes occurred. The case study analysis seeks to identify the multiple, often interlinked factors that had an effect, or compares what was found in the case to theories from the literature in order to see whether one theory matches the case better than others."

This thesis chose an exploratory study approach to provide insight into students' perceptions of and experiences with Git (RQ1). The reasoning for this choice was that little previous research discusses how Git affects group dynamics and the social aspect of project work. Thus, it can provide insight used in later studies.

The research on a mirroring tool on Git data chose a descriptive study approach. The approach explores and describes use cases for the software system. Furthermore, exploratory studies on Git data and mirroring tools exist (although few combine the two topics), reducing the need for an exploratory study.

## Approach to time

Case studies vary in their approach to time. Oates defines three approaches:

- **Historical study:** "Examines what happened in the past by asking people what they remember about earlier events and analyzing documents produced at a time."
- **Short-term, contemporary study:** "Examines what is occurring in the case now."

- **Longitudinal study:** "Involves the researcher investigating the case over time."

This thesis chose a historical study for RQ1. The approach relies on students' memories of how they used and experienced using Git. In addition, the research wants to provide insight into how students feel about using Git in group projects. Thus, asking students about a finished project makes sense.

Student and educator interviews took a contemporary study approach to inspect how interviewees experienced using the system. For the study, we ask what students and researchers obtain from viewing Git data visualized, both concerning previous experiences and in general.

### **Approach to selecting cases**

Case studies focus on an instance of the topic under investigation. Thus, an important factor in selecting what case or cases to study. Oates suggests basing this decision on the following five instances:

- **Typical instance:** "The chosen case is typical of many others and can therefore stand as representative of the whole class. Findings from the one case should be generalizable to the whole class."
- **Extreme instance:** "The case is not typical of others but provides a contrast with the norm."
- **Test-bed for theory:** "The case contains elements that make it suitable for testing an existing theory."
- **Convenience:** "People in the chosen case have agreed to give you access, and it is convenient in terms of time and resources."
- **Unique opportunity:** "The chance arises to study something that you had not previously planned for, and that may not occur again."

Two cases were selected, the courses TDT4140 and IT2810, described in Section 3.1.1 and Section 3.1.2 respectively. This thesis chose IT2810 for convenience, being a course taught by this thesis' supervisor. TDT4140 is a typical instance of a software development course, an attribute it shares with IT2810, and thus a well-suited case to investigate. Similarities between the two courses will substantiate claims made and help identify experiences that are not course-specific. Furthermore, differences in the courses allow for comparing results based on varying aspects of the courses.

## Approach to generalizations

*Generalizations* are an approach to make conclusions that are relevant beyond the cases investigated. Even if some factors are unique to the case, other factors will typically appear in other cases as well [53]. Oates defines four main types of generalizations, used independently or in combination:

- **Concept:** "A new idea or notion that emerges from the analysis."
- **Theory:** "A collection of concepts and propositions with an underlying world-view."
- **Implications:** "Suggestions about what might happen in other similar instances, possibly with specific recommendations for actions."
- **Rich insight:** "What we might glean from reading a case study that does not fit neatly into the three categories of concept, theory or implications, but nevertheless give use important new understanding about a situation."

This thesis approaches the generalization of student perceptions on and experiences using Git by presenting implications and theories. The selected approach is suitable for the time constraints and what insights quantitative data offer. The goal is to explore the research area and provide theories and implications for future research.

For the mirroring tool, the thesis approaches generalizations through implications and rich insight into how students and educators perceive a mirroring tool on Git data. In addition, the approach gives insight into what the mirroring tool offers students and educators and how well its components convey information.

## Selection of data generation methods

The thesis uses two approaches for data generation; questionnaires and interviews. The former is described in Section 3.2 and the latter in Section 3.3.2.

### 3.1.1 TDT4140 Software Engineering (Spring 2021)

#### Curriculum and course dynamics

The course covers software project management and software process types, focusing on agile methods such as scrum and extreme programming (XP). The course provides students with a set of user stories for a software system, which they develop throughout the semester. They are free to choose what technologies to use, some opting for more familiar technologies (Java or Python), and some groups opt for technologies not taught in their previous courses (e.g., TypeScript with Vue or React). The course's



focus is on core IT development for students to build foundations on, and to apply the programming knowledge taught in previous introductory courses.

Student groups must choose how to divide their user stories into two or more sprints and are graded on their ability to apply agile methodologies in software development. The students must complete a set of mandatory software demonstrations, presentations, and group-based deliverables throughout the semester. These account for 100% of the grade in the course.

Groups consist of 7 students and are put together at random across study programs. Within each team, we often observe groups dividing members into roles such as scrum master, designers, testers, front-end, and back-end developers.

### **Who are the students?**

The students typically study Informatics, Computer Science or Industrial Economics, and Technology Management in their fourth semester. Previous knowledge with programming stems from introductory courses on process- and object-oriented programming and the course *IT1901 — Informatics Project 1*. The latter course also introduces agile software development practices and introduces students to both Git and GitLab.

The course is mandatory for the above students. Thus, the student population contains both students who are enthusiastic about programming and those who are not. Students' foundation are based on previous courses, which in regard to technical courses for the most part are the same for all enrolled students.

Students have some experience with Git and GitLab from IT1901, but the course introduces its concepts along with many others. Thus, for many students, TDT4140 is their first time using Git for extended time in a group project with focus on development.

## **3.1.2 IT2810 Web Development (Autumn 2020)**

### **Curriculum and course dynamics**

The course covers technologies and methods used when developing web-based solutions. Students must use multiple popular frameworks, libraries, and archeological design patterns throughout the course. The workload is high, and students are evaluated on one individual project and three group projects. The course covers JavaScript, TypeScript, ReactJS, React Native, RESTful APIs, GraphQL, and databases (Students choose whether to use SQL- or document-databases).

Students work in groups of up to three students and choose to work with friends or be randomly assigned groups. Each project period lasts for about a month, throughout which many students must first learn the relevant technologies before they can start programming. In addition, students have to meet some functional requirements for each project, such as using specific technologies in their applications. Otherwise, they are left to their own in terms of implementing their solution. The course introduces the technologies for each project, but students are primarily responsible for their learning.

Following each project assignment, each student has to review the repositories of 2-3 other groups using the software service *Peergrade*, assessing both quality and implementation of their assignments. Equally, their repository would be reviewed and assessed by 2-3 other students. For each review, the student has to give feedback on how well the group has completed the assignment using free-text answers and a score on a 5-point Likert scale. Many students have little to no previous experience assessing the quality of other student's assignments, let alone reviewing code. Furthermore, this means that all code written by the students will be available for others enrolled in the course to see. Students are anonymous when performing their peer assessment but have access to the GitLab repository of the group they are reviewing.

Finally, at the end of the semester, an exam accounts for 15% of a student's grade in the course. It covers the technologies used throughout all the projects.

### **Who are the students?**

The course is available for fifth semester students admitted to a bachelor/master in Computer Science or Informatics as part of their study programs. Students from other study programs with less programming experience also enroll in the course, e.g., students studying interaction design or cybernetics. It is considered an advanced course listing the following courses as recommended previous knowledge:

- IT2805 Web Technology
- TDT4100 Object-oriented programming
- TDT4180 Human-Computer Interaction
- TDT4145 Data Modelling, Databases and Database Management Systems
- TDT4140 Software Engineering

Students of Informatics enroll in IT2805 Web Technology (covering HTML, CSS, and some JavaScript) in their first semester, whereas Computer Science students do not traditionally enroll in the course. Therefore, students who did not choose JavaScript-based technologies in TDT4140 Software Engineering could very well have no prior knowledge of web development or JavaScript when enrolling in this course. Students of other study programs who enroll in IT2810 typically have less

experience with development, perhaps only from an introductory course. Thus, the course sees a large discrepancy in how well students tackle the projects. Students with less experience are often overwhelmed and report spending many more hours on projects than those studying IT.

Most, if not all, students have used Git and GitLab in one or more previous courses. However, good practices of Git and GitLab are not the curriculum of any previous courses.

### 3.1.3 Comparing the cases

Table 3.1 juxtaposes some qualities of TDT4140 and IT2810. The listed qualities make comparing the two courses interesting to identify both unique and shared factors for both student perceptions on Git and the mirroring tool. We look more closely at their qualities and how they can affect results.

	TDT4140	IT2810
What semester	4th semester	5th semester
Mandatory course?	Yes	No
Course difficulty	Intermediate, course teaches core concepts	Advanced, high workload
Who are students	Mainly students of informatics, computer science and industrial technology.	Students of informatics and computer science and students without IT backgrounds who want to learn web development.
Course focus	Core IT concepts including agile development, project management, development, group coordination, communication with users and customers	Technology-oriented on web development. Teaches technologies and methods used in development of web solutions. Frameworks, architectures, languages, formats and standards used in development of web-applications and services
No. of projects	1	3
Group size	7 students	3 students
Group assignment	Random	Students choose groups or to be randomly assigned a group
Development type	Scrum and sprints	No guidelines
Technology used	Free to choose	Must use specified technologies
Project type	Students use scrum and sprints. Evaluated on their ability to apply agile development methodologies	Students work freely but must complete a set of functional requirements. Evaluated on end product
Version control	Git with GitLab	Git with GitLab

**Table 3.1:** Comparison of TDT4140 and IT2810.

Students of IT2810 have more experience with development when enrolling than those of TDT4140. This is a result both because students are in their fifth semester (compared to fourth semester for TDT4140), and its high workload. Students who enroll in IT2810 are in general more enthusiastic about programming and choose it knowing it requires a lot of work. TDT4140 on the other hand contains the entire student population, including students who are less enthusiastic about programming. The same applies for Git, where TDT4140 students have less experience than IT2810 students. Thus, it is likely that some differences between how the populations experience learning Git occur.

Furthermore, the two courses approach group composition differently. TDT4140 assigns students random groups of seven students, compared to IT2810 where students choose their own groups of three students. As discussed in Section 2.3 random groups increase diversity and larger group sizes increases the occurrence of slackers [22]. Furthermore, in the random groups of TDT4140 not all students initially know each other. Comparing IT2810 groups where many students work with friends, one can imagine some differences in how students experience working in teams. For instance, students might feel differently about Git’s transparency in regard to working with friends compared to strangers.

The difference in how students work throughout the semester will likely affect their Git data points. Students use sprints in IT2810 focusing on the development process, and in IT2810 students focus more on finishing projects and learning technologies. Furthermore, in groups of seven it is expected that student contribution will vary more than in groups of three, at least in regard to difference between the least and top contributing students. Another aspect of larger group sizes is that it is harder to track what all team members do. In teams of three the contribution of each member probably has a larger impact on the end product.

### **3.2 Online questionnaire**

Section 2.2 describes how researchers have explored the advantages and disadvantages of employing Git with GitHub or GitLab in the classroom as an alternative platform to support classroom activities. These studies have assessed how both students and teachers have experienced the switch regarding ease of use, learning benefit, resources used, and more. This thesis will address how students use Git and their experiences to support previous findings and as an exploratory study on Git’s implications on group work. Namely, it studies how students experience using Git, focusing on ease of use, collaboration with other students, and how students personally feel about both giving and receiving feedback on code. It also looks into less explored areas like its effect on the overall social aspect of group work, social interactions, and motivation.

To answer RQ1, a survey was conducted to collect data on students’ experiences and attitudes. Survey research uses sampling and questionnaires to measure characteristics of a population to provide answers and enables the comparison of groups. On the other hand, quantitative research is useful to give quantitative answers to quantify populations’ opinions, attitudes, and behaviors. Furthermore, it is suitable for testing hypotheses and exploring a topic [54].

## Questionnaire Design

The two questionnaires ask mostly the same questions, but some words differ. For instance, the questionnaire for IT2810 has questions about projects, whereas the questionnaire for TDT4140 has questions about sprints. Furthermore, we divide questions into four types, presented in this section.

The first type of question aims to establish what students are participating. They question students' motivation for using Git, their previous experiences with Git and software development, and how their group worked together (physically together or separated).

The second type of statement asks about students' experiences and habits using Git to develop software development in teams. This section aims to give insight into its perceived usefulness and usability.

The third type of statement asks about students' experiences with receiving and giving code reviews and Git's transparency. These questions aim to provide insight into how students personally feel about the interpersonal aspect of peer-reviewing code. Confidence in one's programming skills can vary [55] which may impact how students feel about Git's transparency and also how they experience the feedback process.

The fourth and final type of statement asks what implications Git and GitLab have on the social dynamics of student group collaboration. How does Git's transparency affect the impressions of group members and affect collaboration patterns?

## Data collection

All students of IT2810 and students of TDT4140 were invited to answer the questionnaires via Blackboard and e-mail. IT2810 students were invited in January and TDT4140 students were invited in May. In total 91 students answered the questionnaires, 48 from IT2810 and 43 from TDT4140. To incentivize answering the questionnaire participants had a chance to win a gift card by submitting their email address in a separate questionnaire. The questionnaire is hosted online using the service *Nettskjema*, developed by the University of Oslo, which provides functionality for designing forms to be used in research. They are configured to not store IP-addresses, emails or any similar information about its participants for anonymity.

The questionnaires aim to provide insight into students' perceptions of Git without using any free text answers. Questions are worded as statements to which the student answers with how much they agree with the claim. For example, one statement is *I only want to share code that I know is good*, to which the answer options are Not relevant, Strongly disagree, Somewhat disagree, Neither nor, Somewhat agree or

Strongly agree. The average time spent answering the questionnaire was 8 minutes and 6 seconds.

## **NSD application**

Although none of the individual questions of the questionnaire ask about personal information, an application to conduct the project was sent to the Norwegian Centre for Research Data (NSD). This decision was made based on the assumption that someone could guess what student had answered the questionnaire if viewing answers from an individual.

NSD reports a 30-day response time for applications. For this study, two applications were sent — one time for the IT2810 questionnaire and then again modified the TDT4140 application. The response times were 33 days and four days, respectively.

## **3.3 GitLab mirroring tool**

### **3.3.1 Design and creation**

A popular strategy in computer science research, *design and creation* focuses on developing new software, an artifact. This approach is suitable when the research requires developing a new element of a system or a new system as a whole. Typically, the artifact represents an approach to solve a problem. Oates [53] defines five phases of the design and creation process:

1. Awareness
2. Suggestion
3. Development
4. Evaluation
5. Conclusion

This section succinctly describes each phase. The phases are not rigidly structured as steps but instead form a more fluid, iterative cycle [53]. For instance, during the evaluation phase, researchers gain increased awareness; evaluation results in new suggestions, and so on.

#### **Phase 1 — Awareness**

This phase involves understanding the problem at hand; what to do with GitLab data. The result of this phase was RQ2 and RQ3. A literature review on mirroring tools,

learning analytics, collaborative learning, and mining of Git data was conducted. The literature review provided insight into designing a mirroring tool and what Git data has provided previous researchers. The consensus of mirroring tool research is to opt for visualizations that efficiently convey data and avoid overwhelming users. Furthermore, the background theory emphasizes the importance of selecting and understanding data points.

For this thesis, the initial identified problem was to make use of data points from GitLab both for students and educators. Research on mirroring tools described in Section 2.7 and Section 2.6 describes how mirroring tools can aid in self-reflection with students and monitoring for educators. Varying challenges students encounter when working in teams are described in Section 2.3. Thus, the problem of aggregating student data to help identify uneven contribution, last-minute work, and more was defined.

## **Phase 2 — Suggestion**

The suggestion phase takes a creative leap from understanding a problem to a tentative approach to addressing the problem. In the case of a software system, this means suggesting one or multiple systems and their parts and comparing them. For example, the suggestion can be in the form of paper prototypes or working software.

Early suggestions for this thesis were simple. All commits of a repository were fetched and ordered by hours of the day in a JSON format, which worked to present the idea of aggregating GitLab data by common attributes (e.g., weekdays or authors). Later suggestions involved different graphs, data points, and dashboard functionality.

## **Phase 3 — Development**

The third phase of development sees the implementation of ideas from phase two. The specific implementation will vary depending on the artifact, but the development of the mirroring tool involved software development. Early development iterations revolved around developing the foundations for automatically fetching and storing data from GitLab in the backend and developing a simple frontend system. Later development focused on comparing student groups and combining individual components into a coherent dashboard.

## **Phase 4 — Evaluation**

The evaluation phase aims to evaluate and assess the worth of a created artifact through data collection. Valuable insight from this phase is both what works and what does not work. Throughout development for this thesis, the supervisor of this thesis evaluated components. In addition, an evaluation of the dashboard by

students and educators took place through interviews and demonstrations, described in Section 3.3.2.

## **Phase 5 — Conclusion**

The final phase presents the results of the design process in the form of knowledge gained and ideas for further research. The results of evaluation are presented in Section 5.2 and discussed in Section 6.2 and Section 6.3.

### **3.3.2 Interviews**

As a part of phase 4 of evaluation, semi-structured interviews were held with students and educators to collect qualitative data on the mirroring tool.

All students of IT2810 and TDT4140 received an invitation via email to participate in one-on-one interviews in May 2021. Nine students from IT2810 and 13 students from TDT4140 participated. To incentivize participation, participating students received a free meal and soft drink. In addition, to collect feedback on the educator's view, 11 educators who use version control (GitLab or GitHub) in their courses were interviewed.

The process of collecting and analyzing data is summarized below:

1. Create a list of qualitative attributes.
2. Design interview guide.
3. Schedule the interviews.
4. Attempt to categorize student groups based on qualitative attributes.
5. Conduct the interviews. Have students categorize their group using the qualitative attributes.
6. Transcribe the interviews.
7. Analyze the transcribed interviews by grouping answers into categories or themes.
8. Compare assumed attributes with students' attributions.

The following sections will describe each of the above steps to explain how results were collected and make this research reproducible.



## Create a list of qualitative attributes

Qualitative data deal with characteristics that cannot easily be measured. As such, qualitative attributes are nominal and represent some state or category. All qualitative data is binary; either they apply to a group or do not. In total each group has been categorized using 8 attributes, listed in Table 3.2. For the lack of a better English word M4 uses the Norwegian word *skippertak*, defined by doing most of the work before the deadline and doing less work during most of the project.

ID	Statement
M1	Some members have contributed more than others on the programming
M2	Some members have been free-loaders on the programming
M3	The group has worked evenly through each exercise/sprint
M4	The group has done <i>skippertak</i> every exercise/sprint
M5	The group started working early in each exercise/sprint
M6	The group worked in planned spaces of time (e.g. 8-13 Mondays)
M7	One group member was responsible for setting up the project
M8	Everyone in the group contributed an even amount in the programming

**Table 3.2:** Qualitative attributes used to categorize student groups before and after interviews.

## Design interview guide

Following an interview guide, interviews were semi-structured, enabling the interviewer to ask follow-up questions and new questions as needed. The same interview guide was used to interview students from both courses and separated the interview into three parts. First, interviews opened with some questions about the student's group to get to know the interviewee and to make them comfortable in the setting. Then, students answered the question, "What kind of visualizations of GitLab data can help you understand your progress as a group?" and what these visualizations should look like (e.g., graphs, tables, text).

Students received an introduction to the dashboard in the second part and its functionality described to them. Then students are instructed to view the dashboard at their own pace and reflect out loud what the visualizations mean to them. Students were suggested to use sentences structured as "This [surprises, makes sense to, is interesting to] me because" and to express what they like and dislike about the dashboard's features.

The third and final part is adevelfter students have explored all or most of the dashboard. Students answer a few questions about the visualizations and the dashboard. Finally, they categorize their group using the attributes from Table 3.2.

## **Scheduling interviews**

All interviews took place over a week. First, students signed up using the software service Doodle[CITE] by selecting time slots and leaving their email addresses. They were then contacted and invited to participate in an interview lasting 30 minutes. All students interviewed were given information on the study, the interview, and how their data would be collected and used and sign a consent form. Students also said what group they belonged to analyze their group, described in the next section.

## **Attempt to categorize student group based on qualitative attributes**

Viewing only the dashboard solution before interviewing students, student groups received attributes from Table 3.2. This process involved viewing a group's visualizations and making assumptions based on these. For instance, answering M4 for IT2810 student groups involved looking at dates of commits and merge requests. As a result, student groups with a significant spike in commits and merge requests near the end of the project received the attribute of "skippertak"-groups.

## **Conduct interviews and have students categorize their group**

The interviews were conducted on campus and lasted around 30 minutes, and all but three participants spoke English. As can be seen in Appendix A.2, the interviews started with information about the project and data management to make the interviewee comfortable with the interview setting. Then, each section of the interview guide was gone through. If students answered imprecisely or ambiguously, follow-up questions served to clarify. Although the scheduled length of interviews was 30 minutes, some students were interested in testing the dashboard for longer. If one had more time, one could collect more data by asking more questions or letting the interviewee navigate longer.

## **Transcribe the interviews**

All recorded interviews have been transcribed using the software solutions Temi [56] and oTranscribe[57]. Temi is a paid service for transcribing, offering machine-automated transcriptions for 0.25 USD per minute. In total, the interviews tallied 18 hours. The quality of transcriptions varied from interview to interview; thus, omatic transcriptions served to do the heavy lifting of initial transcribing. All interviews were listened through, and the generated transcriptions were corrected and refined (e.g., remove repeated words). The manual transcription of Norwegian interviews used oTranscribe.

Anonymization happened during transcription, replacing students' references to other students, group IDs, and names. However, the value of data remained the same, as the information itself is not relevant for this research.

At the end of each interview, students categorized their group using the qualitative attributes from Table 3.2.

### **Analyze the transcribed interviews**

Interviews were all analyzed using NVivo [58], a qualitative data analysis software. Its core functionality allows users to read through textual data (e.g., transcribed interviews) and code/classify sections of documents or text of common themes. Text coded with the same theme can easily be retrieved and compared by reviewing the themes.

### **Comparing Qualitative Attribute Categorizations**

Students' assumptions are compared with assumptions made before interviews, and discussed in Section 5.2.1. Points of interest are how many assumptions were correct and incorrect. Furthermore, how many times groups correctly *were not* attributed is of interest.

# Chapter 4

## Dashboard design and implementation

### Introduction

To answer RQ2 and RQ3, a web-based dashboard was implemented. The dashboard supports both students and educators and consists of two views categorized as mirroring tools. Students have their data points from GitLab mirrored back to themselves, whereas educators can view all student groups accompanied by group performance indicators. The dashboard is a proof of concept to aid student groups in self-reflection, educators in gaining insight into student groups' performance, and to test whether qualitative assumptions on student groups can be made based on data points from GitLab.

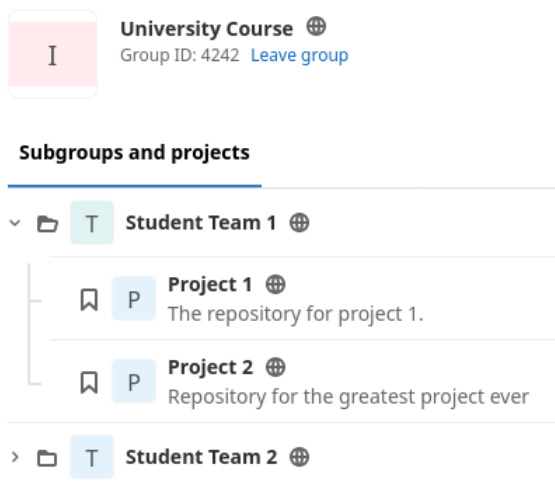
The dashboard is simple in architecture, consisting of a browser-based dashboard, a MongoDB database, and a Node backend for intermediate communication between the dashboard and GitLab.

### 4.1 GitLab API and data collection

GitLab has a well-documented API listing how to query specific resources. Resources belong to either *Projects*, *Groups* and *Standalone*, but for this implementation, only *Projects* and *Groups* are relevant. *Groups* logically group one or more *projects* (repositories) or even more *groups*. *Codebases* are hosted as *projects* and have accompanied *issue trackers*, *repositories*, *merge requests*, *CI/CD*, and more. Most data of *Groups* and *Projects* are retrieved using API [59].

The code written to populate the dashboard works best for courses structured in a specific manner. A course must have an overarching group containing all student groups within, which in turn contain projects (See Figure 4.1). The dashboard queries the Group API for all subgroups of a course and all projects of each subgroup. Because GitLab places a rate limit on their endpoints, one should use either query for new resources upon using the system or semi-regularly to maintain an up-to-

date representation of its data. Data is stored in an intermediate database to increase performance, reduce the number of queries sent to GitLab. The current implementation requires manually fetching new data when needed.



**Figure 4.1:** How courses can be structured in GitLab. A real course would have more student teams.

## How data is retrieved

Data of a specific course (GitLab Group) is queried for and stored iteratively through a manual process. The first step is identifying what top-level GitLab groups to query. Here we are interested in the groups that contain student groups, in Figure 4.1 this is the group with an ID of 4242. The request-response contains an array of subgroups.

All queried groups are stored in a database and iterated through, querying for and storing their projects. Metadata about groups and projects are stored, and then project-specific data is queried and stored. For the dashboard, this meant querying for commits (also containing information on lines changed) and storing this. GitLab's API allows for quickly extending the dashboard to include more data (e.g., merge requests, issues, or pull requests).

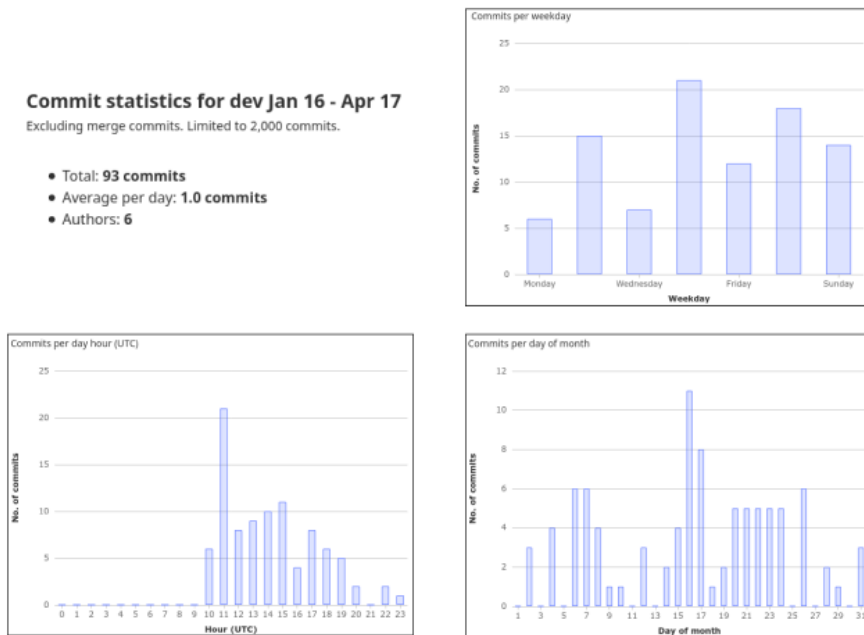
Most API requests will only return public information if not authenticated. Authentication methods include OAuth2 tokens, project access tokens, personal access tokens, and more. In addition, all queried data requires authentication, which correlates to one's role in the course group on GitLab (E.g., educator or student). For example, access to all subgroups and projects requires Maintainer-role or above,

whereas students have access to their projects and can view other public projects in the course group.

## Data aggregation and transformation

GitLab returns much data for queried resources; for instance, each project resource consists of some ninety attributes. Many of the attributes are flags of no interest to the regular GitLab user, e.g. *emails\_disabled*, *wiki\_access\_level* and *lfs\_enabled*. Although useful in the GitLab ecosystem, they do not intuitively provide any value in a mirroring tool. The system’s goal is to increase awareness about actions and behaviors in students. Thus, presenting the same data in the same way GitLab does will hardly provide anything. To meet this goal, we aggregate data from GitLab and present it differently without mutating the data, leaving the locus of processing in the hands of students or educators.

One approach to infer information that is not explicit from GitLab data is to group data points by similar attributes. As seen in Figure 4.2 GitLab provides some visualizations of commit data, providing insights into each branch separately. Commits represent some amount of work, and when presented distributed by weekdays, a developer team may infer what days of the week they produce their work.



**Figure 4.2:** GitLab’s analytics displaying commits per weekday and day of month of a master branch.

The dashboard visualizes four data points from projects: merge requests, project issues, project commits, and code line changes from commits. Because Git places few restrictions on its users, the intrinsic value of each data point will vary from project to project and viewer to viewer. This concern was willfully disregarded during the dashboard development, focusing on presenting data points and measuring their perceived usefulness.

#### 4.1.1 Limitations and workarounds

Although GitLab offers an extensive API, it has some limitations that the dashboard must overcome. This section presents both limitations and workarounds to provide insight into some challenges encountered during development.

##### Commit authors

In the dashboard, commits and lines written (calculated from commits) are presented both on a project basis and separated by author (as shown in 4.7 and Figure 4.11). All commits have the field *author\_name*, being the person that authored the commit. However, the courses in this study use FEIDE, an SSO service, for authentication with GitLab. Thus, all commits authored in GitLab have students' full names as the *author\_name*. Having full names would not pose a problem if it were not for the fact that most students have their local Git configurations set to use a different name. From the development of the dashboard, it is clear that most students have the field *user.name* of their local Git configuration set to their username on GitHub.

The problem arises when their local *user.name* and their GitLab name vary. Naively displaying commits and lines by *author\_name* often results in students occurring twice (sometimes thrice or more), reducing the readability of the data presented. For instance, the author of this thesis would appear twice, as *Åsmund Haugse* and *asmundh*. In addition, often, Git names and real names are correlated, perhaps including first name or surname.

The dashboard uses an algorithm to combine commits of the same student with different *author\_name* fields to remedy this. Commits are combined if the three first letters of its author's name are equal to the three first letters of a different commit's author. The Norwegian letters æ, ø, and å, are compared to the letters ae, o, and aa. Although a somewhat naive solution, it worked remarkably well. Before each student interview, an extra control of the combinations took place, and if the algorithm failed to combine all names of a person, the algorithm was manually tweaked. Ironically, the algorithm would fail to combine the author's commits.

## Advised Group Selection — Viewing each project

The dashboard presents educators with student groups clustered by what project they want to view (as seen in Figure 4.4). For example, IT2810 had three group projects. Thus three buttons were added to switch between them. However, not all student groups followed the same naming convention when naming their repositories. For example, for the second project, names included "Prosjekt2", "Prosjekt 2", "Project 2", and "Project2", but also less conventional names like "reacting-poetry." The casing on letters also varied.

Consequently, the query attempting to fetch all related repositories (projects) must accommodate the variable names. Populating the projects assign an extra attribute to all projects returned from GitLab, namely *name\_normalized*, removing white spaces, using lowercase letters, and replacing the letters æ, ø, and å with ae, o, and aa. For example, when fetching the second project of the course, we queried for "prosjekt2" and "project2", "prosjekt3" and "project3" for the third project. A similar workaround would have to manually be implemented for other courses if grouping other projects on this page.

Educators can impose restrictions or naming conventions for each project repository to reduce the manual configuration of this page. The current implementation only displays projects adhering to the above naming scheme. For TDT4140, where students only complete one project, the dashboard fetches all repositories and has no options to choose a project.

## 4.2 Software technology used

The development of this proof-of-concept application has focused more on rapid development than on clean code and development methodologies. Thus, the application uses technologies that allow a rapid implementation to examine GitLab data better.

The application uses the React framework using JavaScript in the frontend. All graphs use the open-source library Chart.js [60] which provides an easy-to-use API for visualizing data. The main development challenges have been to efficiently perform aggregating operations on GitLab data to reduce re-renders and improve loading speeds. Some data aggregation occurs in the frontend and some in the backend application.

A focus has been on writing reusable frontend components to create an extendable application. Chart.js offers different graphs (bar, doughnut, pie, line charts, and more) requiring datasets to be structured differently. Several implemented graph components support a scalable amount of datasets, an exciting feature to compare students or student groups.



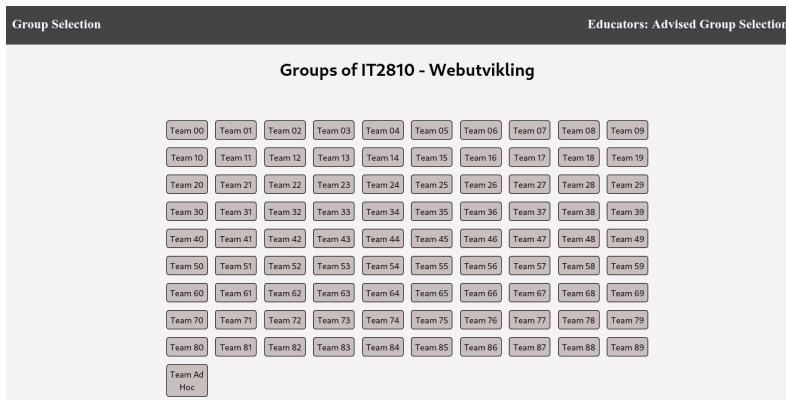
The backend application has been implemented with Node.js using JavaScript. It has three main concerns: fetching data from GitLab, storing data from GitLab, and providing data to the frontend application. Data storage uses the document database MongoDB and the ODM Mongoose. Koa was chosen as an HTTP server to serve data to the frontend application because it is lightweight and offers an easy API for rapidly implementing endpoints.

Database population worked by sending HTTP requests using Postman to endpoints in the backend application. Populating the database prepares the system for use in a new course. Depending on the request, this triggers a series of HTTP requests to GitLab, all authenticated with a GitLab Access Token. For each request, GitLab checks whether the access token is authorized for the resource queried. Access tokens and the URL of GitLab (can vary between institutions) are stored as environment variables to improve security and reduce the time needed to adapt the system for use by others.

## 4.3 Dashboard components

### 4.3.1 Group selection

Users of the dashboard are interested in viewing student groups or projects, and currently, there are two ways to gain insight. Figure 4.3 displays a simple list of student teams by name. An educator might be interested in viewing a student group that has contacted them, or a student group may be interested in viewing their data.

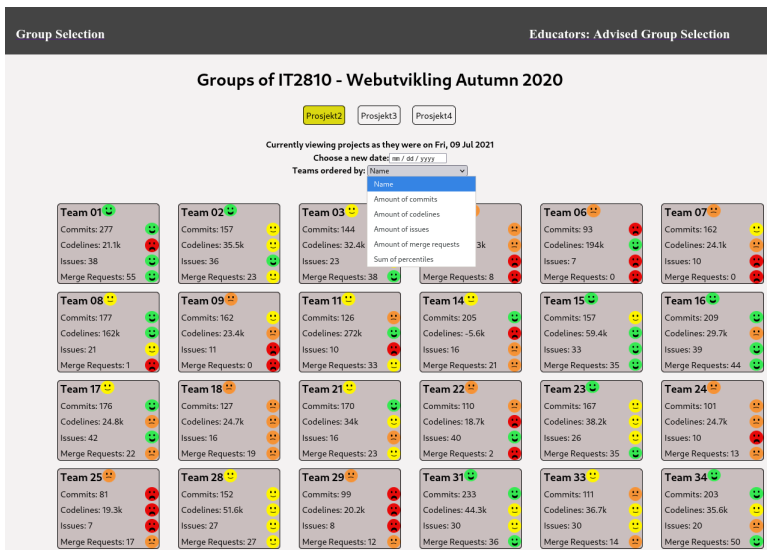


**Figure 4.3:** Page for selecting what student group to view.

Educators may be interested in insight into how student groups perform throughout a project period, out of curiosity, or perhaps help student groups that struggle actively. Aiding educators in choosing what groups to inspect more closely, the menu uses smiling faces in varying colors to indicate what quartile the student group is in for a metric. This is shown in Figure 4.4. From bad to good, the colors are red, orange, yellow, and green, red meaning the group is in the bottom 25 percent of all groups for that metric, orange 25-50%, yellow for the 75-50% , and green for 75-100%.

Instead of displaying student groups, an ordered list shows specific projects. The possible orders are by name, the number of commits, lines, issues, or merge requests, or by the sum of their metric quartiles. When calculating the sum of metrics, each metric is weighed equally. Red faces hold a value of 25, orange 50, yellow 75 and green 100. If their sum is in the range 100-200 they get a red face, 200-300 orange and so on.

Because educators may be interested in viewing course health at a given points in the semester, a calendar menu was added. The list view only uses data points from the selected date and earlier. In other words, selecting September 17th displays a course as it were on that date.



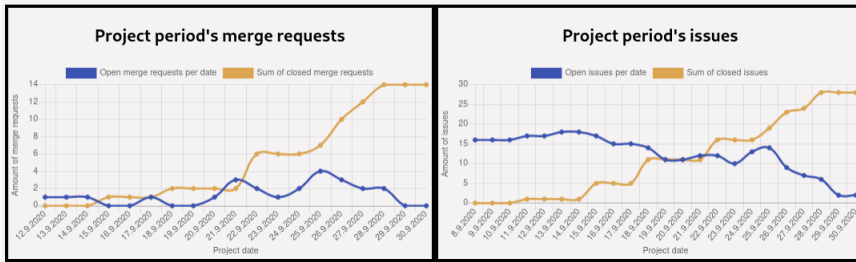
**Figure 4.4:** Project list with statistics and indicators to help educators choose groups to inspect.

### 4.3.2 Project period's issues and merge requests

To display merge requests and issues of a project, a line chart component to display one or more datasets was implemented. Typical datasets of line charts represent how data values change over time. A cubic interpolation function is used on the data points to smoothen the changes of values. Figure 4.5 shows the component used to display changes in open and closed merge requests and issues over time. The orange-colored lines in each chart display the sum of closed merge requests or issues on a given date, whereas the blue lines display the number of open merge requests or issues on a given date.

The rationale behind a line diagram of merge requests boils down to how most students work. Students using feature branches or the GitFlow workflow mainly create merge requests for code that has been completed. Thus, merge requests are slower to rise than commits; one merge request can come from many commits. Nonetheless, merge requests can also be tiny.

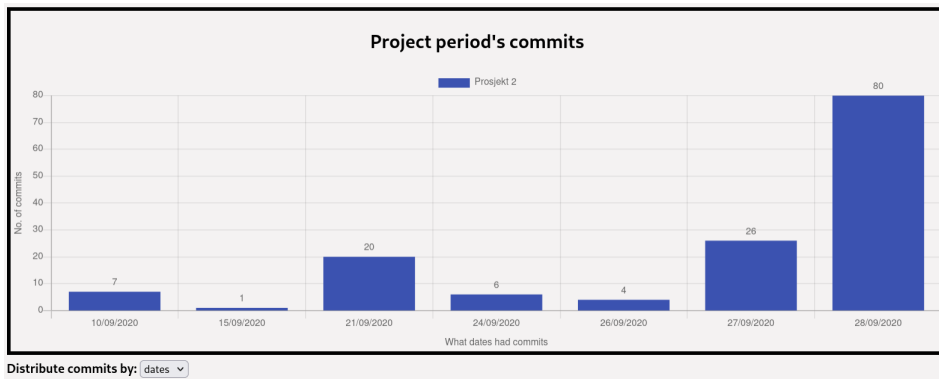
The above rationale applies for issues. Students close issues when they are completed. Some students delay closing issues, others close them immediately. Furthermore, viewing the amount of open issues at a given time is interesting. It provides insight into a team's work backlog, and viewing when students create issues give insight into how they work. Do they create issues only during sprint planning or open issues concurrently?



**Figure 4.5:** Line charts displaying a project period’s merge requests and issues.

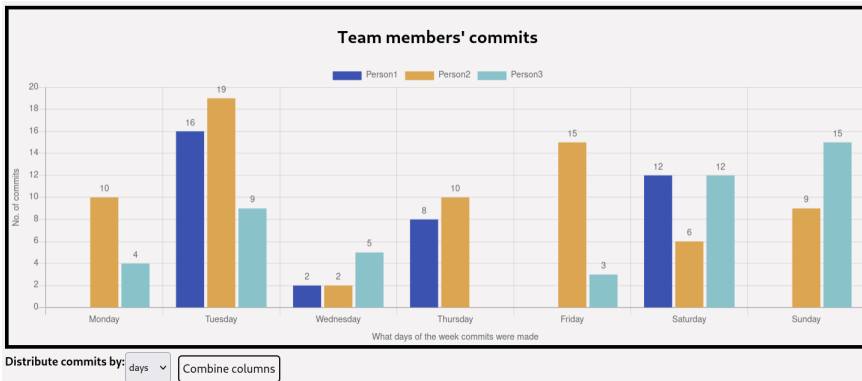
### 4.3.3 Project period’s commits and lines of code

Bar charts present categorical data using rectangular bars. The length or height of bars is proportional to the values they represent. This component displays one or more datasets using vertical bars. Along the X-axis are the categories (e.g., weekdays or calendar dates), and the Y-axis and height of bars show the corresponding values. The components display commits and additions or deletions to lines over time for each project. All bar charts can display the same data distributed by weekdays, dates, or hours of the day using the dropdown below the chart. Figure 4.6 shows a simple bar chart displaying commits per date of the project period. Note that it only lists dates that have one or more commits.

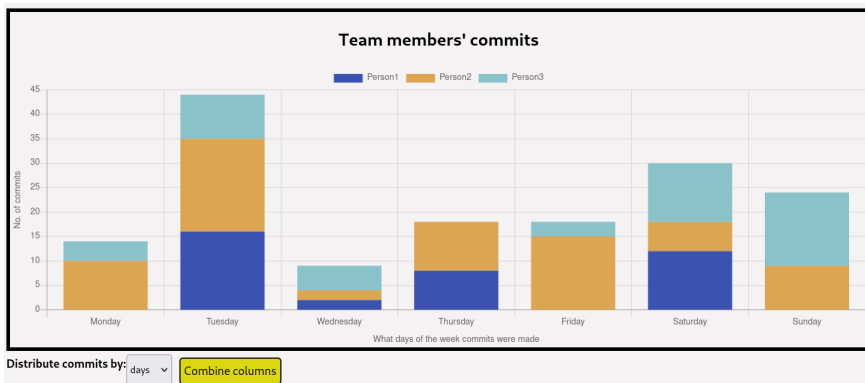


**Figure 4.6:** Bar chart displaying a project’s commits distributed by dates.

If the component is given more than one data point it will display them using multiple bars as shown in Figure 4.7. If provided with many groupings on the X-axis and many entries per grouping, it can appear cluttered and hard to read. As a remedy bar charts with multiple data points have the option to combine columns as shown in Figure 4.8.

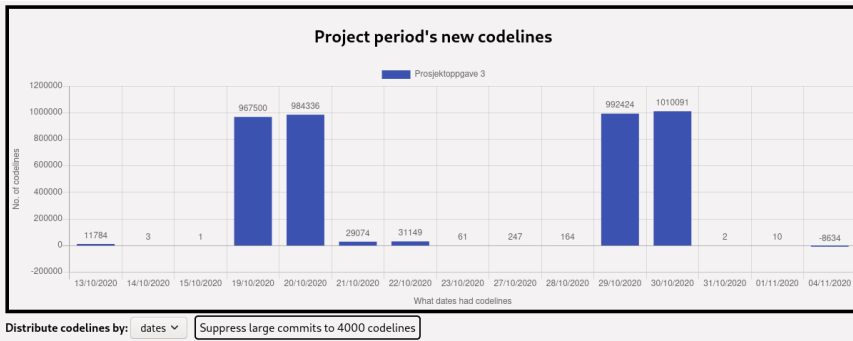


**Figure 4.7:** Bar chart displaying team members' commits distributed by weekdays with columns per member.

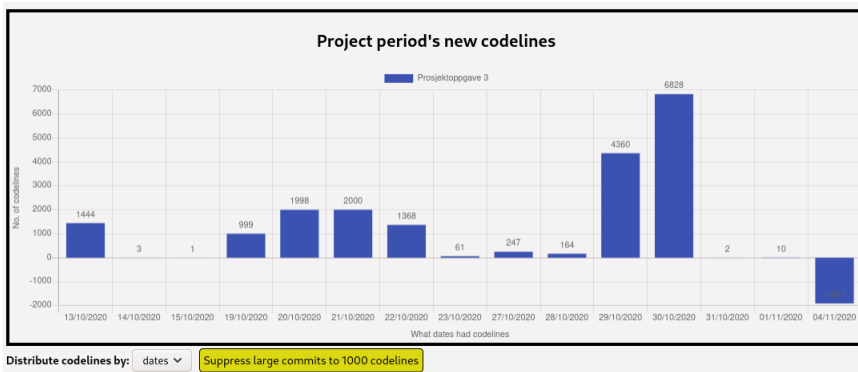


**Figure 4.8:** Bar chart displaying team members' commits distributed by weekdays with combined columns.

The code line metric from commits can see some large numbers (millions of lines changed) in projects where the gitignore configured poorly. Changes in code lines occur twice when merging branches, one for the initial commit and once for the merge commit. The option to *suppress* the size of large commits from their original size down to 1000 lines helps prevent skewed graphs. Skewed graphs were often occurred in JavaScript projects of inexperienced groups where commits contain millions of lines changed because of node modules.



**Figure 4.9:** Bar chart displaying a project's code lines distributed by dates.

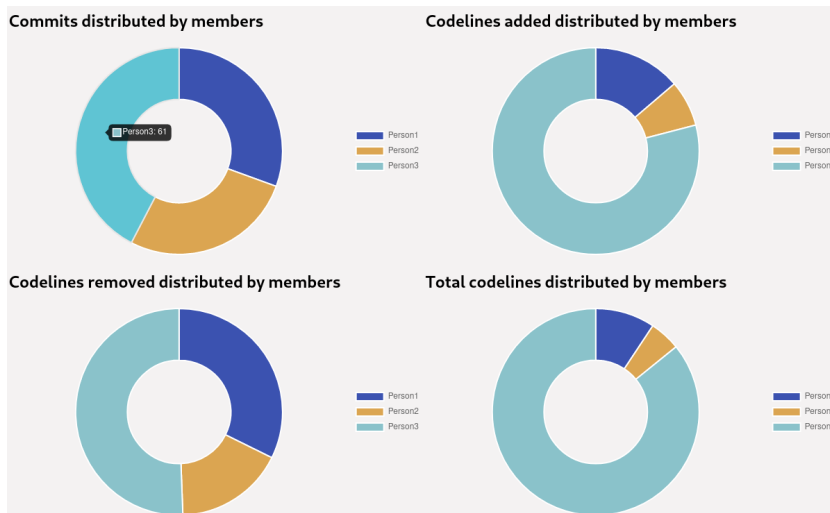


**Figure 4.10:** Bar chart displaying a project's code lines distributed by dates but suppressed to a commit size of 1000 code lines.

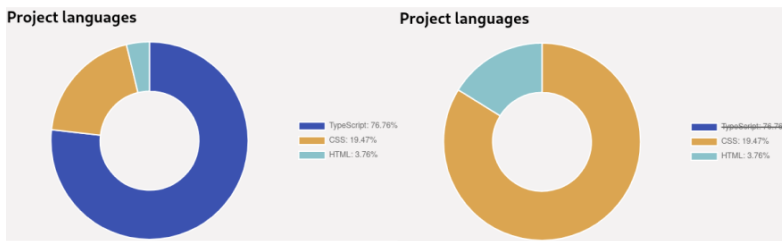
Figure 4.9 and Figure 4.10 illustrate how large commits can skew this graph and reduce its readability. A concern is information loss when suppressing commits, but "human-written" commits are seldom larger than 1000 lines. Also, note that the code lines graphs show the sum of both code lines added and removed.

#### 4.3.4 Commit and code line distribution by members

Doughnut charts convey values relative to each other, each slice representing their share of some total value. Their readability diminishes when the amount of slices increases and slight differences in values are hard to see, but to remedy this, hovering over a slice will display its value (as seen in Figure 4.8). The doughnut charts present commits and code lines distributed by members and show programming languages used in a project.



**Figure 4.11:** Doughnut graphs showing distribution of commits and changes to code lines by members.



**Figure 4.12:** Doughnut graph showing distribution of programming languages used in the repository. The right doughnut has one entry removed.

### 4.3.5 Project commits list

GitLab provides a list view of a repository's commits displaying concise information about each commit. The dashboard recreated this list (see Figure 4.13) but only display the commit message, author, date, and changes in code lines. The component provides information to help understand large spikes in the graphs showing code lines or commits. Should the user want more information, they can press *View in GitLab* to it in its entirety. This component is less about providing new information and more about clarifying the information conveyed in the graphs.

Commit Message	Author	Date	Codelines Added	Codelines Removed	Action
merged duplicate fronte...	Name of Person1	29 Oct 2020	+235 lines	-95 lines	<a href="#">View in GitLab</a>
implemented login and d...	Name of Person1	29 Oct 2020	+125 lines	-126 lines	<a href="#">View in GitLab</a>
local merge	Name of Person1	29 Oct 2020	+98 lines	-76 lines	<a href="#">View in GitLab</a>
edited file names and im...	Name of Person2	29 Oct 2020	+689 lines	-9 lines	<a href="#">View in GitLab</a>
Merge branch 'fix_merge...	Name of Person2	29 Oct 2020	+785 lines	-83 lines	<a href="#">View in GitLab</a>
Merge branch 'fix_merge...	Name of Person3	29 Oct 2020	+785 lines	-83 lines	<a href="#">View in GitLab</a>
deleted duplicated file	Name of Person3	29 Oct 2020	+0 lines	-24 lines	<a href="#">View in GitLab</a>
Merge branch 'fix_merge...	Name of Person1	29 Oct 2020	+0 lines	-24 lines	<a href="#">View in GitLab</a>

Order by:  ▼

- Date
- Codelines added
- Codelines removed
- Author

**Figure 4.13:** List view of commits of a project. Can be ordered by date, author, amount of code lines removed or added.



# Chapter 5

## Results

### 5.1 Questionnaire results

This section will present the results from the two questionnaires. Answers to similar questions will be juxtaposed to display similarities and differences. Findings related to the research questions are discussed in Section 6.1. The results include personal characteristics, experiences, and habits using Git, perceptions on Git's transparency and code reviews, and finally, Git's implications on student group collaboration. IT2810 students are more experienced than students of TDT4140, which likely has an effect on the results. Thus, the answers may give insight into how knowledge and experiences develop in the student populations. The statements presented have all been translated from Norwegian, but the sentiment of the statements remains the same.

#### Personal characteristics

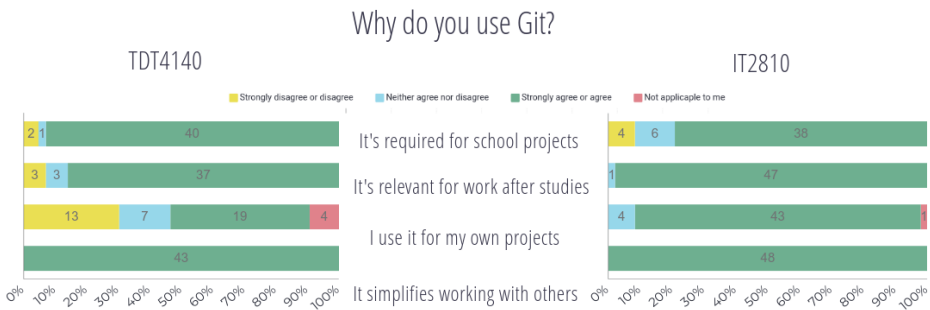
This section gives insight into who our respondents are and their experiences before taking the courses. Furthermore, we want insight into how they perceive using Git for version control.

We may correlate students' perception of Git to how much experience they have with the tool. In Table 5.1 we see how much experience students had with Git and web development before the courses. For TDT4140, 72.1% answered they were somewhat (N=25) or not (N=6) experienced with Git, compared to IT2810 where 81.3% of students said they were quite (N=32) or very (N=7) experienced with Git. In both courses, groups do web development, although it is a smaller part of their projects in TDT4140. Experiences with Git may be affected by their experiences with programming. 83.7% of TDT4140 students report having little (N=19) or no (N=17) prior experience with web development, whereas 81.3% of IT2810 students had little (N=19) or some (N=20) prior experience. Only one student of IT2810 reported having no prior experience with development, compared to 17 (39.5%) of

Characteristics	TDT4140		IT2810	
	N	%	N	%
<b>How experienced are you with Git?</b>				
Not experienced	6	14%	0	0%
Somewhat experienced	25	58.1%	9	18.8%
Quite experienced	9	20.9%	32	66.7%
Very experienced	3	7%	7	14.6%
<b>How much previous experience did you have with web development before the course?</b>				
No experience	17	39.5%	1	2.1%
Little experience	19	44.2%	19	39.6%
Some experience	6	14%	20	41.7%
Much experience	1	2.3%	8	16.7%

**Table 5.1:** Juxtaposition of student answers on previous experiences with Git and web development.

TDT4140. The results suggest that the student population of IT2810 are more experienced compared to TDT4140.



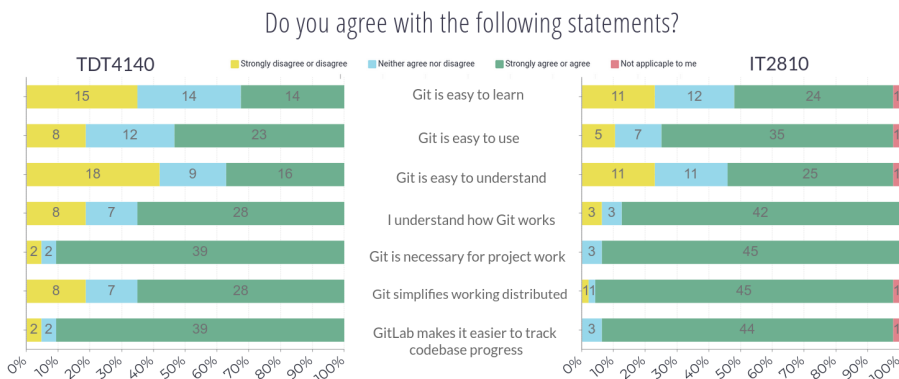
**Figure 5.1:** Composition of answers on why students use Git.

As seen in Figure 5.1, students of IT2810 appear to have a slightly different relationship with Git compared to TDT4140 students. 100% of respondents from both courses agree or strongly agree that Git simplifies working with others. Substantiating the claim that IT2810 students are more experienced we observe that 89.6%, N=43, of IT2810 students use Git for their projects compared to 44.2%, N=19, of TDT4140. These results also suggest that students of IT2810 to a larger degree do programming outside of courses compared to TDT4140 students.

Unsurprisingly, most perceive Git as valuable for work after studies, 97.9%, N=47

and 86%, N=37, of IT2810 and TDT4140 students respectively agree or strongly agree. Lastly, 93%, N=40, of TDT4140 students use Git because it is required for school projects compared to 79.2%, N=38, of IT2810. A possible explanation is that many TDT4140 students have mainly used Git when they had to in courses, compared to IT2810 which saw more students using Git for their own projects as well.

## Experiences with and habits on using Git

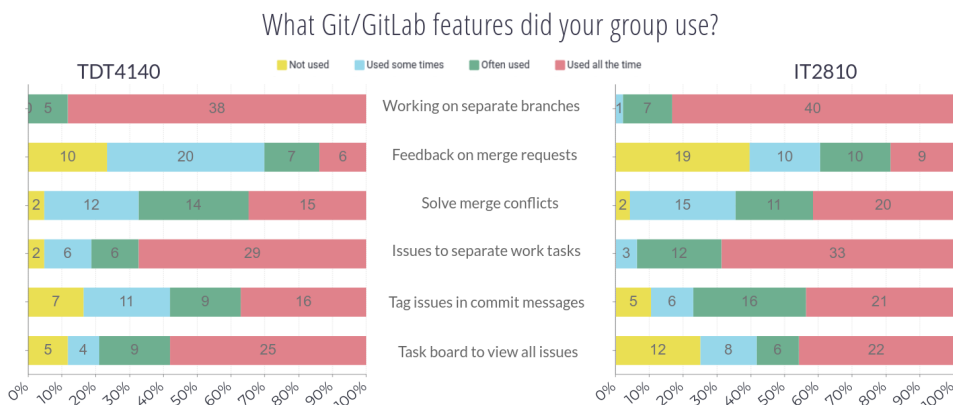


**Figure 5.2:** Student answers on learning Git and ease of use.

This section will present responses to questions related to using Git for development to assess where students stand with Git as a tool. We juxtapose the first three answers and observe that for IT2810, only 50%, N=24, agree that Git easy to learn, 72.9%, N=35, agree it is easy to use, and 52.1%, N=25, agree that it is easy to understand. In contrast, for TDT4140, only 32.6%, N=14, agree Git is easy to learn, 53.5%, N=23, agree it is easy to use, and 37.2%, N=16, agree it is easy to understand. The large amount of students who find Git agree that Git is hard to learn may be a result of how the respondents were taught Git. They had little or no formal introduction of Git and many are self-taught. Considering that the average student of IT2810 has a better grip of programming compared to the average TDT4140 student, it makes sense that fewer TDT4140 students think Git easy to learn. The same reasoning offers an explanation for why fewer TDT4140 students think Git is easy to use or understand.

Student answers show that fewer TDT4140 students understand how Git works (65.1%, N=28) compared to IT2810 (87.5%, N=45). 18.6%, N=8, of TDT4140 students say they do not understand how Git works, even after using Git for an entire semester. However, although not all understand Git, most think it is necessary for project work and that it makes it easier to track progress in the codebase (90.7%,

N=39 of TDT4140 and 93.8%, N=45 of IT2810). Most of IT2810 students (93.8, N=45) agree that Git simplifies working distributed, but only two-thirds of TDT4140 students report the same. This disparity between the courses may be a result of both how students have used Git and how much experience they have. 18.6%, N=8, of TDT4140 students disagree with the statements, suggesting Git makes it more difficult to work distributed. Perhaps these students struggle with programming or Git and prefer working around others. 16.3%, N=7, of TDT4140 neither agree nor disagree, possibly because they have no experience working distributed with Git.



**Figure 5.3:** Student answers on what Git features their group used.

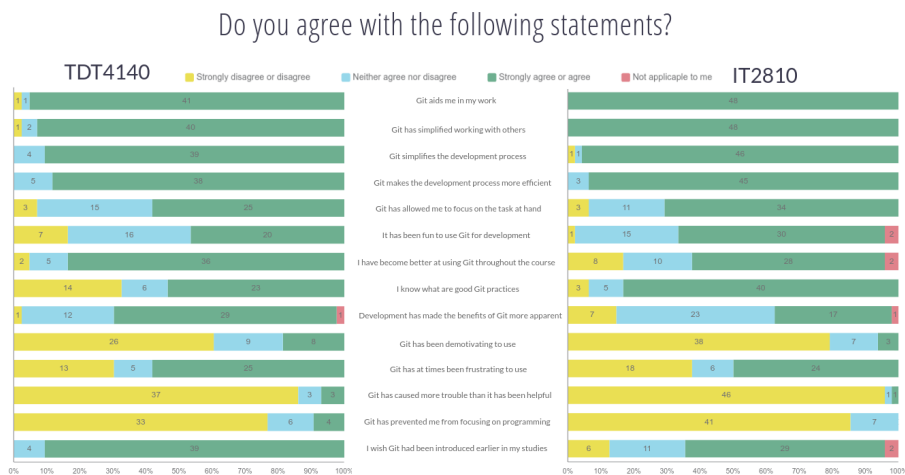
GitLab offers multiple features to improve collaboration in the development process. In Figure 5.3 students are asked what features they used. Course staff of IT2810 advised students to tag issues in their commit messages, and other students provided feedback on their use of Git. For the most part, both student populations were left to their own to decide on what GitLab features to use.

At first glance, the results appear similarly distributed in the two courses. Notably, TDT4140 students appear to have given feedback on merge requests more than IT2810 students, with 23.3%, N=10, of TDT4140 and 39.6%, N=19, of IT2810 report not using it. Code written in TDT4140 lived longer than code in IT2810. Thus, TDT4140 students may have been more careful to review code before it entered their master branch. On the other hand, IT2810 students were not graded on quality but whether they had functionality, reducing the need for quality assurance.

Most students of both courses report that they used issues to separate work tasks, solved merge conflicts, and worked on separate branches. The three features are often considered good practice, so the large amount of students using them are not surprising. Although students were not instructed to separate tasks into issues, *all*

students of IT2810 reported doing so, and 95.3%, N=42, of TDT4140 report doing so. This suggests that students find separating work tasks to be helpful, which correlates to the large amount of students who said GitLab makes it easier to track codebase progress.

The GitLab task board is an extension to its issue tracking as shown in Figure 2.2. Often used with scrum, it makes sense that more TDT4140 students (88.4%, N=38) report using it compared with IT2810 students (75%, N=36). Lastly, only 10.4%, N=5, of IT2810 and 16.3%, N=7, of TDT4140 did not tag issues in commit messages. Doing so increases the traceability of performed work and is considered good practice.



**Figure 5.4:** Student answers on using Git in the development process.

The statements in Figure 5.4 probe into how using Git affected students motivation and their development process. In both courses, close to all students report that Git aids them in their work, simplifies working with others, improves the development process, and makes the development process more efficient.

It is well known that Git can be difficult to use at times, but respondents seemingly tackle its complexities well. For IT2810, only 6.3%, N=3, students agree that Git has been demotivating to use, one student agreed it caused more trouble than it has been helpful, and 50%, N=24, agreed it has been frustrating to use at times. Results of TDT4140 are slightly worse, almost one out of five (18.6%, N=8) agree it has been demotivating to use, 7%, N=3, said it had caused more trouble than it has been helpful, and 58.1%, N=25, agree it has been frustrating to use at times. Furthermore, four students (9.3%) from TDT4140 agree that Git has prevented them from focusing on programming. No students of IT2810 reported the same. The

difference in student answers are likely direct results of how much experienced the student populations have with Git. The more experienced IT2810 students mainly have few problems with Git, compared to the less experienced TDT4140 students. The numbers for TDT4140 suggest that students do not have sufficient experience with or have not been sufficiently introduced to Git.

However, after a semester of using Git with GitLab for their development, 83.3%, N=40 and 53.5%, N=23, of IT2810 and TDT4140 students, respectively, agree they know what good Git practices are. Comparing the responses to what features Git and GitLab features students used, it appears that some students are following good practice without knowing it. A cause can be that a team member who has an idea of good practices instructs other team members on how to use the features.

Furthermore, 58.3%, N=28, of IT2810 and 83.7%, N=36, of TDT4140 report they have become better at using Git and 35.4%, N=17, to 67.4%, N=29, respectively agree Git's benefits have become more apparent. Considering that neither course teaches Git, the results suggest that leaving students to learn Git on their own sees some success. Although fewer students of IT2810 agreed, this may be correlated to the student population's previous experience with Git. For instance, 72.1%, N=1, of TDT4140 students report being little or not experienced with Git. Thus, any exposure to Git will likely provide learning.

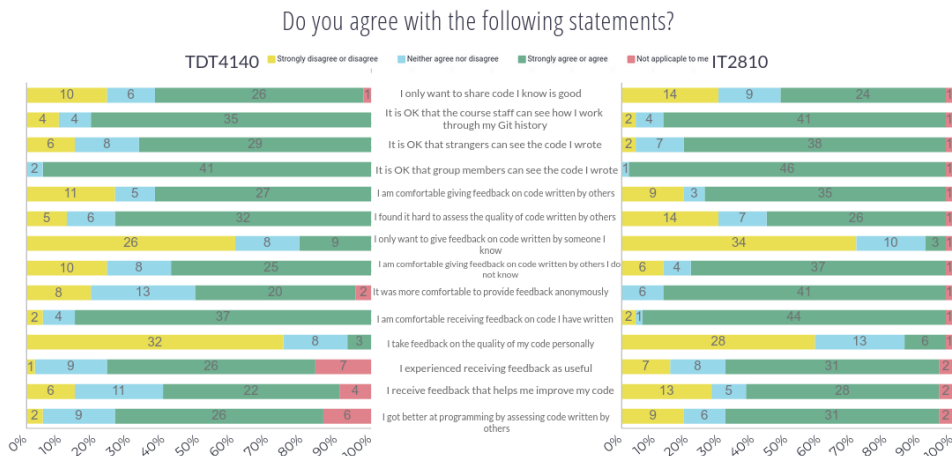
62.5%, N=30, of IT2810 students report that Git has been fun to use, compared to 46.5%, N=20, of TDT4140. 16.3%, N=7, of TDT4140 students disagree (N=5) or strongly disagree (N=2), that Git has been fun to use, possibly correlating to results on demotivation, frustration, and trouble associated with the use of Git. Viewing the results of the statement "*I wish Git had been introduced earlier in my studies*", we see 90.7%, N=39, of TDT4140 and 60.4%, N=29, of IT2810 students agreeing or strongly agreeing. No TDT4140 students disagreed, but 12.5%, N=6, of IT2810 disagreed, while 22.9%, N=11, were indifferent to the statement. The results do not provide insight into how students want Git introduced.

### **Git's transparency and feedback on code**

Working with Git on a project implies that all code will be available for all team members. The statements of Figure 5.5 aim to provide insight into how students experience this aspect of Git and also to assess how students feel about providing and receiving feedback. From the results, we observe that many students only want to share code they know is good, reading 50%, N=24, of IT2810 and 60.5% of TDT4140 students.

Regarding Git's transparency, the two courses are similar in their answers. Of IT2810 and TDT4140 respectively, 85.4%, N=41 and 81.4%, N=35, agree that it

is OK that staff can view their Git history, 79.2%, N=38 and 67.4%, N=29, are OK with strangers seeing their code, and 95.8%, N=46, to 95.3%, N=41, are OK with group members seeing their code. In a course context only educators and team members are likely to see students' code. Thus, the positive numbers are reassuring.



**Figure 5.5:** Student answers on Git's transparency and on giving and receiving feedback.

Five statements cover how students feel about providing feedback on code written by others. For the most part, students are comfortable providing feedback (79.29%, N=35 of IT2810 and 62.8%, N=27, of TDT4140), although they find it challenging to assess code quality (54.2%, N=25 and 74.4%, N=32). The two statements may be correlated, for example, discomfort when providing feedback may be due to insecurities about the feedback they provide.

Most students of IT2810 (70.8%, N=34) disagreed with the statement *I only want to give feedback on code written by someone I know* and only 6.3%, N=3, agreed, whereas for TDT4140 60.5%, N=26, disagreed and one out of five agreed (20.9%, N=9). The third-year students of IT2810 are more comfortable giving feedback on code written by students they do not know (77.1%, N=37) compared to the second-year students of TDT4140 (58.1%, N=25). Students of IT2810 likely have more experience with giving feedback on code written by strangers as a result of the peer grading they performed. This assessment was anonymous, and 85.4%, N=41, agreed it was more comfortable to provide feedback anonymously. In TDT4140, most feedback on code was not anonymous and on code by group members, yet 46.5% of students agreed. Although pinpointing what TDT4140 students agree with, it may be that some students would prefer the option to provide feedback anonymously.

On the topic of receiving feedback on code, the two student populations are again reasonably similar. Most students do not take the feedback received personally (only 12.5%, N=6, of IT2810 and 7%, N=3, of TDT4140 agree), and experienced receiving feedback as useful (64.6%, N=31 and 60.5%, N=26). Furthermore, the feedback helped students improve their code (58.3%, N=28 and 51.2%, N=22). Although peer reviews are part of the course grade in IT2810, a quarter of students (27.1%, N=13) disagree with the statement *I receive feedback that helps me improve my code*. In TDT4140, where feedback is provided mainly by group members, only 14%, N=6, disagree with the same statement. A possible explanation is that students perform peer reviews after a project's deadline, before starting their next project. Many students experience this as a tedious chore, possibly impacting the quality of the feedback they provide. However, most students of IT2810 (64.6%, N=31) and TDT4140 (60.5%, N=26) report that their programming improved by assessing code written by others.

### **Effects on social dynamics of group work**

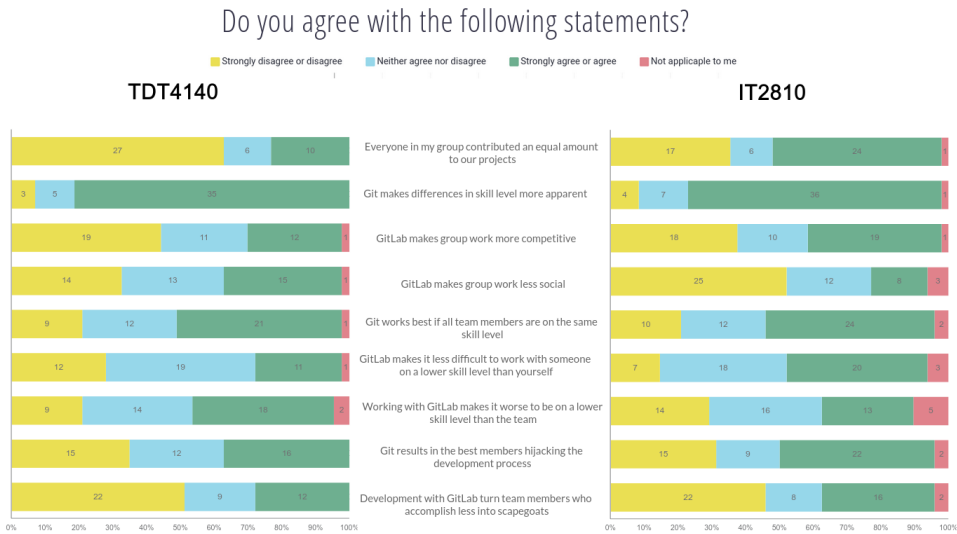
This thesis wants to answer how version control with Git affects the social dynamics of student groups. Groups of IT2810 mainly consisted of 3 students, whereas groups in TDT4140 mostly were seven students. Furthermore, IT2810 students could choose whether to be assigned a random group or pick whom to work with, and two-thirds (68.7%, N=33) chose their group. For TDT4140, groups were random, and no equivalent data was collected. Furthermore, 41.7%, N=20, of IT2810 students reported primarily working sitting together; the remaining students mainly reported working sitting distributed. The TDT4140 questionnaire asked the same question but included the option *We worked a lot both distributed and together*, to which 55.8%, N=24, answered the latter, and only 18.6%, N=8, of students reported that they mainly worked sitting together.

Perhaps as expected, large portions of both student groups report an uneven contribution to projects. 50%, N=24, of IT2810 agree everyone contributed an equal amount, and 35.4%, N=17, disagree. TDT4140 responses show 62.8%, N=27, of students thought contribution was uneven, and only 23.3%, N=10, agree it was equal. The size and composition of student groups likely affect these results. Students work in groups of seven in TDT4140; thus equal contribution is less likely. Results may also be affected by whether groups are randomly assigned or not, as IT2810 sees a larger amount of groups reporting equal contribution and students choosing their group.

Offering quantifiable metrics of contribution in terms of completing issues, amount of commits, and similar metrics, 39.6%, N=19, of IT2810 students agree that using GitLab makes group work more competitive, 37.5%, N=18, disagree, and 20.8%,



N=10, neither agree nor disagree. On the other hand, a smaller portion of TDT4140 agrees with the same statement, only 27.9%, N=12, whereas 44.2%, N=19, disagree and 25.6%, N=11, neither agree nor disagree.



**Figure 5.6:** Statements on working with GitLab and Git with an interpersonal topic.

Worryingly a third (34.9%, N=15) of TDT4140 students agree that GitLab makes group work less social, about a third disagree (32.6%, N=14) and a third neither agree nor disagree (30.2%, N=13). On the other hand, IT2810 sees more students disagreeing (52.1%, N=25) and fewer agreeing (16.7%, N=8) with the statement. Using GitLab for programming simplifies working distributed, which for some TDT4140 students may differ from how they previously have done programming in group projects. Thus, this may impact the percentage of students agreeing with the statement. In contrast, the more experienced programmers of IT2810 might not consider distributed working to affect the social aspect of group work.

Most students agree with *Git makes differences in skill level more apparent*, namely 75%, N=36, of IT280 and 81.4%, N=35, of TDT4140. Furthermore, 50%, N=24 and 48.8%, N=21, respectively agree with *Git works best if all team members are on the same skill level*. Git’s transparency makes it easier to track contributions by members, which likely affects the large amount of students who agree with the former statement. Responses to the latter statement likely refer to how working with Git to distribute work leaves developers to themselves, which works the best if students are autonomous.

Although close to one out of four (27%, N=13) of IT2810 students and 41.9%, N=18, of TDT4140 students agree with the statement *Working with GitLab makes it worse to be on a lower skill level than the team*, 50%, N=24, and 25.6%, N=11, respectively agree with *GitLab makes it less difficult to work with someone on a lower skill level than yourself*. On the other hand, for TDT4140, 27.9%, N=12, disagree and 44.2%, N=19, neither agree nor disagree; the latter students perhaps feel it does not have an impact. The large amount of TDT4140 students who agree with the former statement is likely due to a larger amount of the students being or *feeling* that they are on a lower skill level. Furthermore, few TDT4140 students agree that GitLab makes the situation less difficult, likely for the same reasons. The more experienced IT2810 students feel oppositely, but this may also be correlated to the group size of three. A smaller group makes it easier to follow up team members.

Supplementing the former statements, a third (33.3%, N=16) of IT2810 students and a fourth (27.9%, N=12) of TDT4140 students agree with *Development with GitLab turn team members who accomplish less into scapegoats*. Worth noting is that 45.8%, N=22, of IT2810 and 51.2%, N=22, disagree with the same statement. The statement responses may reflect how less experienced students feel about their contribution, but it can also be more experienced students who feel this way. The slightly larger amount of IT2810 students who agree may correlate to team sizes, because team members who contribute less will be more detrimental to the project.

Although omitted in Figure 5.6, the statement *Git results in the best members hijacking the development process* also stated "(Rewriting a lot of code, doing all the work themselves, etc.)" in the questionnaire. A concerning 45.8%, N=22, of IT2810 students and 37.2%, N=16, of TDT4140 students agree with the statement. The answers may be correlated to student answers on GitLab making group work more competitive, but also to how student groups work with issues in GitLab.

## 5.2 Mirroring tool results

This section will present results from the semi-structured interviews with students and educators. The results will be presented both as summative results from multiple interviewee responses and single answers from interviewees will be cited, providing nuance to the results. Furthermore, the results of making assumptions on student groups will be presented. An overview of all interviews can be seen in Table 5.2.

Overview of the interviews			
Interviewee	IT2810 student	TDT4140 student	Educator
No. of interviews	9	13	11
Average length	27 minutes	25 minutes	48 minutes
Course of action	Interview and interviewee tested the dashboard	Interview and interviewee tested the dashboard	Interview and demonstration of the dashboard

**Table 5.2:** An overview of the interviews held to demonstrate and test the dashboard. It shows the distribution of interviewees, the average interview length and the course of action for each interview.

### 5.2.1 Qualitative attribute assumptions

As mentioned in Section 3.3.2 and Section 3.3.2, the groups of all students who were interviewed were ascribed a set of qualitative attributes before their interview. This process took no longer than five minutes per group. Towards the end of each interview, the students themselves ascribed their group the same attributes.

This section will present the results and how the dashboard was viewed when deciding on an attribute. The qualitative attributes are listed in Table 5.3.

Attributes M4, M5, and M6 concern students' habits across all projects or sprints, but some inconsistencies in how groups were attributed occurred. For example, some groups for whom the attribute would be valid for one or two projects or sprints were attributed, whereas other similar groups may not have. The inconsistency has both been the case when attributing groups before interviews and when interviewees attributed their groups. Thus, the three attributes are possibly less accurate.

Furthermore, the group reviews followed only a mental model on how to attribute groups. A more guided decision-making process would likely give different results. E.g., attributing M2 to an IT2810 group, one could define groups where a student has fewer than 20% of commits and lines qualify for the attribute.

ID	Attributed using dashboard		Attributed by interviewee		Correctly attributed		Wrongly attributed		Difference attributions		Correctly not attributed		Wrongly not attributed		Difference not attributed		Assumption was correct		Accuracy	
	IT2810	TDT4140	IT2810	TDT4140	IT2810	TDT4140	IT2810	TDT4140	IT2810	TDT4140	IT2810	TDT4140	IT2810	TDT4140	IT2810	TDT4140	IT2810	TDT4140	IT2810	TDT4140
M1	3	13	6	12	3	12	0	4	3	11	3	0	3	0	0	0	6	12	66.7%	92.3%
M2	3	8	0	3	0	1	3	7	-3	-6	6	3	0	2	6	1	6	4	66.7%	30.8%
M3	8	10	5	4	5	4	3	6	2	-2	1	3	0	0	1	3	6	7	66.7%	53.8%
M4	5	3	5	11	3	3	2	0	1	3	2	2	2	8	0	-6	5	5	55.6%	38.5%
M5	5	12	5	11	3	10	2	2	1	8	2	0	2	1	0	-1	5	10	55.6%	76.9%
M6	8	13	9	12	8	12	0	1	8	11	0	0	1	0	-1	0	8	12	88.9%	92.3%
M7	3	7	3	5	0	2	3	5	-3	-3	3	3	3	3	0	0	3	5	33.3%	38.5%
M8	5	2	1	2	1	0	4	2	-3	-2	4	9	0	2	4	7	5	9	55.6%	69.2%

**Table 5.3:** Results of attributing student groups qualitative attributes. For the columns "Difference attributions" and "Difference not attributed", green cells means the attribute was correctly assumed more times than not. Red cells mean the opposite. Accuracies larger than 60% are marked in green, below 60% are marked in red.

### M1 — *Some members have contributed more than others on the programming*

M1 was attributed fewer times by before interviews than by the interviewees. When attributing a group, a review of the dates of a group’s commits and the number of lines committed took place. The reviews attributed M1 to groups with some students appearing on many more dates and authoring many more commits and lines than their team members. For IT2810, we observe a 66.7% accuracy or two-thirds, whereas, for TDT4140, the accuracy was 92.3%. The group reviews failed to recognize that some students had contributed more in a third of the cases (N=3) of IT2810.

For TDT4140, the reviews attributed all groups with M1 before interviews, and all but one student attributed it to their group. Thus, the accuracy of TDT4140 assumptions may correlate to the size of student groups and not the accuracy of making assumptions before interviews. For example, when seven students work together, some students will likely contribute more than others.

### M2 — *Some members have been free-loaders on the programming*

The most negatively loaded attribute of the eight saw a low percentage of precision for TDT4140 (30.8%) and correct assumptions for two-thirds of IT2810. When reviewing student groups for M2, an attempt was made to identify groups where one or more members had only a tiny portion of the commits made and lines contributed.

In IT2810, no student groups attributed their group with M2, whereas three groups received the attribute before interviews. As a result, the remaining six groups were correctly not attributed M2. Students explained that differences in previous programming experience meant some students needed more time to produce code and that pair programming could explain the number of false attributions of M2.

Eight groups of TDT4140 were attributed M2, and although three students did too, only one of the groups overlapped. In other words, seven groups were wrongly attributed with M2. Furthermore, through interviews with TDT4140 students, it was

made apparent that many groups had done a significant amount of pair programming on one computer, effectively leaving no signs of contribution in commits.

### **M3 — *The group has worked evenly through each exercise/sprint***

M3 attributions saw higher accuracies for IT2810 groups (66.7%) than TDT4140 groups (30.8%). Factors for attributing groups were how student groups' graphs of merge requests developed and the consistency of commits made throughout the project period. Groups whose sum of closed merge requests steadily rose and who made commits often were attributed with M3.

Groups of TDT4140 had varying sprint lengths, and when a sprint ended was not known when reviewing groups. In other words, an assumption of both when sprints started and ended was made before assuming the consistency of work done. The accuracy of assumptions would likely increase if these dates were known beforehand.

Three IT2810 groups were misattributed M3, and one group was falsely not attributed, indicating an eagerness to attribute groups with M3. In addition, the mental model for what constitutes evenly working may have set the bar too low.

### **M4 — *The group has done skippertak every exercise/sprint***

Skippertak, defined as doing most of the work before the deadline, saw poor accuracy for both courses. The same shortcoming of when sprints started occurred as for M3, whereas for IT2810, it was more apparent when students worked on the project. Groups attributed with M4 had commit-graphs with large spikes on the last days of the project and totaled fewer days with commits.

In eight cases (61.5%), TDT4140 students attributed their group with M4, but review before interviews did not. This discrepancy suggests a poor job of identifying when groups had performed a skippertak in sprints. But, again, this is likely affected by the unknown sprint periods. Furthermore, 11 of 13 students attributed their group with M4, suggesting a trend in the course.

Compared to TDT4140, numbers for IT2810 were better, but two groups were wrongly attributed. The two students whose group was wrongly attributed specified that they had done some skippertak, but not every exercise. The absolute wording of M4 may have contributed to this inaccuracy. From student interviews, it was apparent that the approach to identifying skippertak often was accurate but that not all groups had consistently done skippertak. For example, some groups did skippertak in two exercises but not the third, yet review attributed them M4. However, this was not the case every time.

**M5 — *The group started working early in each exercise/sprint***

Graphs of commits, merge requests, and issues were consulted to decide on this attribute. IT2810 groups who only had a few commits in the first week or two did not receive M5. Again, the shortcoming of not knowing when sprints started applies. Based on assumptions on when sprints ended, the number of commits made in the days after a sprint had ended was inspected.

Results for TDT4140 were better than IT2810 with 76.9% to 55.6%. In addition, 11 of 13 students of TDT4140 attributed their group with M5, suggesting that the attribute was widely present in groups of interviewed students. Thus, a correlation to how the course and project are structured, requiring students to work more consistently, is suspected. Furthermore, because groups were assigned randomly, it may have been less acceptable to delay starting a sprint.

Review before interviews wrongly attributed and failed to attribute two groups, respectively, suggesting an imprecise approach to identifying groups. The low accuracy of IT2810 may also be affected by the inconsistency in how groups were attributed, similarly to M4.

**M6 — *The group worked in planned spaces of time (e.g., 8-13 Mondays)***

Attributed by all students of IT2810 and 12 of 13 TDT4140 students, this attribute was persistent in many student groups. In addition, the commit-graph was distributed by weekdays and hours of the day when attributing groups with M6.

Results for both courses showed high accuracies (88.9% for IT2810 and 92.3% for TDT4140), but this is likely mainly due to it being a common way of working. Its counterpart, sporadic working, is less commonly observed for development because members must know what remains of the project.

**M7 — *One group member was responsible for setting up the project(s)***

Groups whose commit charts, when distributed by members, saw only one student was present for the first day, in contrast to two or more students present were attributed with M7. In addition, students assigned this attribute to their groups if only one student set up the project, and if two or more students partook in the setup, they did not receive the attribute.

Accuracies were low for both courses (33.3% for IT2810 and 38.5% for TDT4140). Several groups had multiple students partake in the setup but using one computer, which would not show in the commit-graph.

**M8 — *Everyone in the group contributed an even amount in the programming***

Allowing for some variable amounts of commits, and when the graphs indicated the group working, groups attributed with M8 had no significant discrepancies between team members.

For TDT4140, review before interviews had an accuracy of 69.2%, but attributed no student groups with M8 successfully, instead correctly not attributing nine groups. Two students attributed their group with M8 when the review did not, and the review attributed two groups when the student did not. Results suggest not that the review correctly identified the attribute but that it was not present in student groups of TDT4140. Drawing a comparison to M1, this can be because all students seldom contribute an even amount in groups of seven students.

One would expect better results for IT2810, but the review before interviews only had an accuracy of 55.6%. In addition, reviews wrongly attributed four groups with M8, suggesting that the metrics used for evaluating M8 for IT2810 groups gave an inaccurate insight into contribution or that review consulted the data wrong.

### 5.2.2 Component feedback

This section will present feedback and comments on the components of the dashboard. During the interviews and demonstrations, some components received less attention than others, reflected in the feedback.

#### Advised group selection for educators

The advised group selection screen was of most interest to educators, providing quick insight into the composition of all student groups of an entire course. The current implementation by none perceived to be perfect, and most educators could think of one more change that would fit better to their needs. Despite its shortcomings, the general feedback was that the dashboard was helpful in its current state but could be much more helpful if tweaked.

The use of colored faces was well-received by all educators to quickly convey information visually. However, most educators were indifferent to smiley faces, instead emphasizing the use of colors. One educator suggested using even more colors to provide a more nuanced view of how the course as a whole was doing. His reasoning is quoted below:

*"If you could signal it using a continuous kind of scheme, that probably would be more helpful. Because from reality, that's how it is. Groups are not performing really good and then somewhat good, but actually performing slightly worse or slightly better on this metric or the other and so on."*

Each group displayed had four metrics: commits, lines, merge requests, and issues. These were perceived to be good choices of quantitative metrics to capture student groups, although not equally valuable. For example, one educator (who teaches Git) suggested that for an exercise where the curriculum is on issues, that metric would be the only interest. Moreover, he noted that for courses where students had yet to learn merge requests and issues, it would not make sense to use those metrics.

Educators suggested two functional changes to make the list more practical and less naive. Both were initially suggested by the same educator (although repeated by others in later interviews) and concerned how to convey group performance using the colored faces. First, educators should define how each metric is weighted to have the total score of groups be more tailored to a particular course.

The other suggestion entailed using colored smiley faces to partition student groups relative to each other—the dashboard listed student groups into quartiles for each metric and a total score. However, multiple educators expressed they would be more interested in seeing student groups who performed *even worse*, for instance,



the groups who were in the bottom 5% of the course were probably performing worse than the best of the 25% worst. Both functionalities were introduced to several other interviewees, who agreed it would be helpful to increase the list's ability to depict how student groups were performing accurately.

In line with Soller et al.'s [43] theoretical work on meta-cognitive tools, several educators suggested an alternative to sectioning students into colored faces based on how they performed compared to other students. Their suggestion was to define thresholds for each of the colored faces for a given date, e.g., two weeks into the project period, a green face on merge requests for more than 20 merge requests, whereas a red face would have fewer than four. This way, all groups who perform well will be shown as doing well, compared to the current implementation, which for 100 good groups would show 25 of them as red faces.

In other words, these educators want to define a model of interaction based on their expectations for their course. On this topic, multiple educators expressed concerns for students gaming the system. For example, if students knew that educators looked for many commits or issues, they could make more small data points to make it appear as if they were doing better. To remedy this, one educator suggested also showing the average amount of lines of commits.

Interviewees suggested new metrics for the page as well. One educator mentioned that an average of commits per day would be of interest. Others suggested a metric to give insight into how data points were distributed amongst members to identify groups who had free-loaders. An educator whose students work alone on projects suggested that the list also displayed a cake diagram (or a similar solution) displaying when students had the most commits during the project period.

### **Language graphs**

In interviews with students, this graph was of little focus. However, three students referenced the component, all using words along the line of *"This obviously makes a lot of sense and does not surprise me"*.

Before viewing the dashboard, one educator said he would be interested in seeing the programming language distribution. His students choose their technology stack themselves. Once shown, he said *"Viewing a group now, I'd look at project languages, from which I assume this group used Django"*.

### **Merge requests and issues**

The two graphs showing merge requests and issues over time are situated at the top of the view and show when clicking a project. Most students agreed that the two

graphs accurately depicted the progression of their project(s). For example, spikes in merge requests resulted from crunches or the group merging all the work done on separate branches in a given period. For TDT4140, spikes in issues often meant sprint planning had occurred, and plateaus indicated students had worked on other projects.

One student, viewing a spike in merge requests near the deadline of a project, said

*"That is probably because we worked in separate branches until we had to merge it at the end. And I think that would actually be quite useful to see, because this is an indication that the team needs to merge things a lot more often, that we probably work in individual branches more, which is not necessarily a good thing, because that means when you start merging stuff, you just get really stressed and things break".*

Although students noted that issues were accurate of how they worked, student groups had different habits when closing issues. For example, some groups reported closing issues consecutively as they worked; others reported only closing issues at the end of a sprint or when their group met.

Educators were also pleased to see how issues and merge requests developed over time; some admitted not considering the data points as indicators beforehand. As metrics to provide insight, an educator said *The curves (of the two diagrams) are very useful because they provide insight into how the students experience the scope of the projects and how they work towards it.* Furthermore, the same educator expressed that upon viewing issues visualized, he became aware of how he could restructure the use of issues in the course to allow for more effortless follow-up of student progress.

## **Commit graphs**

Commits are displayed per project and in a separate graph showing project commits separated by members (see Section 4.3.3). Furthermore, both graphs could display their data distributed by dates, day of the week, and hours of the day. This section will present results for both graphs and their distributions.

All students attempted to view the data using the different distributions. Although most students switched distributions unprompted, some students did not view change distribution until it was encouraged. Most students agreed that commits displayed per date indicated when they worked. Spikes on a date often meant they worked more, although in some cases, students recalled that they had many commits on some days because they were doing minor changes to their documentation (README-file) in GitLab.

One student, referencing the project period's commits graph displayed by dates, said *"Project period commits as well is kind of three periods, the three sprints. A lot of the commits happen at the end of each sprint, as you can kind of see here"*. The consensus of students was that the project period's commits did a good job conveying when the group worked on the project. Days with few commits often meant less work, and conversely, more commits often meant they worked more.

One student, whose graph, when distributed by hours of the day, showed a large portion of commits between 12-16 o'clock, said *"We always worked between 12 and 16, so it would be nice to see if we actually did work in that time period or if we wasted it by always just talking. But as you can see, it actually was the time when the group as a whole was the most effective"*. Several other students expressed the same sentiment, one student expressing *"I think it's clear that we've worked in our scheduled hours. And I think that it's clear all group members didn't work that much outside of working hours"*.

In the graphs showing commits distributed by members, although group members' names were anonymized, many students could make themselves out or at least one group member from anomalies or patterns in when during the day commits were made or the sheer amount of commits. Two educators expressed concern about working habits on display, one referring to it as surveillance. The other educator expressed concern for students who may experience having their behavior displayed as sensitive and uncomfortable.

Commits distributed by weekdays met much of the same feedback as hours. Student groups who scheduled weekly workdays mostly saw their workdays have more commits than other weekdays, yet some students of TDT4140 were made aware of group members who did more work during the weekends than they had thought. When viewing commits distributed by members and displayed by weekday, multiple students confirmed that columns with all members present were their scheduled workdays. However, this was not the case for all groups.

Students disagreed on whether the graph showing commits by team members accurately captured how group members did work. Although most students agreed that differences in commit habits (e.g., committing often or not, large or small commits) could skew the proportions of contribution, it was perceived to do a good job of roughly capturing when students contributed. Below is a quote from a student commenting on commits distributed by members:

*"I also like the team member commits where you can kind of see the distribution between people on the dates when there's been done a lot of commits. I think that's an interesting one to look at. You can also see how people have been working, like*

*one person will have a few commits every day. And then somebody has like a lot on the last day, that's a good metric to see like, okay, somebody wasn't, they weren't doing what they were supposed to be doing. And then they did everything at the last minute. Yeah. Because you can see, like in project three we were working a bit more throughout the project."*

## **Code lines graphs**

The graphs showing code lines added to the repository were the most poorly received by students. Because the dashboard counted changes from merge commits, numbers were often twice that of the actual codebase. As a result, the total code lines count of some student groups dropped below zero. On the other hand, students whose graphs had large commits (often from generated code) and educators alike appreciated the option to suppress large commits. One student said *"That's a really important and cool feature"* and another said *"I think this feature is nice, to suppress the large commits. Just to like, not corrupt all the data"*.

Before viewing the dashboard, several students noted that lines of code would be an interesting metric to see visualized. However, students whose groups had large commits that skewed the graph experienced the metric not representing progress in the project. Several of these students remarked that the importance of a gitignore-file was made more clear to them.

As a metric, lines of code was by most students, deemed the least useful and representative of work. One student accurately put this into words saying *"That's interesting to see, how commits don't necessarily represent code lines written. And similarly, how amount of code lines aren't equivalent with work done"*.

The code lines metric was explained when introducing the dashboard, but negative bars confused some students. Columns dropped below zero if students had more deletions than additions for the commits in a given period, often caused by refactoring or deletions of files or folders. Quoted below is a student succinctly summarizing this:

*"I have a hard time making sense of it. Because like you said, it doesn't really say anything about the work done since removing code lines is a negative. So, so it might be that on this day that says minus 77, maybe that's a day we did a lot of work, but we did a lot of work in both ends"*

Although viewing code lines in isolation yielded little new insight for most, multiple students observed the asymmetry of commits and lines of codes. Some members would produce fewer lines per commit and vice versa. One student noted *"This is pretty interesting. Like these two, between each other, added code lines and*

*commits. Because someone likes to commit a lot, which I think is nice. And then there's people that, might be me as well, who write a lot of code, like fix something and then don't commit until it's fixed. That's interesting to see distributed by members".*

Most educators only glanced at the two graphs, but some commented on their applicability. One educator said *"Viewing lines of code as a metric of quality is dangerous, but it can still be interesting. It was intriguing to view groups with negative millions code lines, because that too can give some thoughts into what the group has done".* On the topic of large amounts of code lines, another educator expressed *"It basically means that you likely have a red flag here [...], so don't care about the metrics. Just tell the group, clean up your repo".*

### **Team member comparison doughnuts**

The doughnut diagrams were placed at the bottom of the dashboard and were often the last graphs student saw. Although they did not introduce a new metric, one student remarked *"I think it shows a different type of information with the same information, or different types of information with the same data. Because, this shows much more which member pushed the most and did the most work".*

Interviewees perceived the graphs as more efficiently displaying team members' commits and code lines than the bar charts. One student, comparing the doughnuts to the bar charts, said *"I thought these charts at the bottom was easier to see the difference in the team members rather than the one here".* Several students noted that by viewing the doughnut graphs, they could probably make out their team members from the distribution of pieces.

Each piece of the doughnut showed the sum of that metric for a team member, e.g., the number of commits a team member had. Some students maintained the concern that differing habits and abnormally large commits would skew these graphs as well. Because GitLab already offered functionality to see the sum of team members' commits, one student expressed that the graphs provided little new insight and that he could manage without the doughnuts.

Several students and educators alike asked if they could suppress large commits in the doughnut graphs like they could the code lines graphs to accurately depict how many lines members added or removed. On a similar note, it was suggested not to count lines of code from merge commits. That way, 100 added lines in a commit on a branch later merged into master is not counted as 200 lines added.

## List of commits

Only some students used the list of commits, primarily to cross-reference a spike in the number of commits or lines of code at a given time. However, one student commented that the ease of access to individual commits was helpful to provide more insight into the trends of the graphs.

Some students suggested that the list should allow for toggling whether the commit should be included in the other graphs to remedy the large skewing commits.

One educator thought it was helpful to have the list of commits easily accessible to view how commit messages were formatted. Other educators also said they were interested in whether students had written good commit messages or not, but they did not mention the commit list during the interviews.

### 5.2.3 Accuracy of visualized GitLab data points

As mentioned earlier in this report, GitLab data points do not capture all that goes into software development. It is interesting to see how Git's lossy nature affects perception on the accuracy of the dashboard's visualizations. This section presents students' opinions and perceptions on the accuracy of the dashboard compared to their impression of their projects.

An attribute all students of TDT4140 and many of IT2810 commented on was that the dashboard does not reflect pair programming. Because the interviewed students commonly used the development technique, this reduced the overall ability of the dashboard to capture the development process. One student group had used Git's co-authoring functionality to mark commits with two authors, but this did not improve its accuracy because the dashboard did not account for it. Furthermore, several students expressed that comparing students based on the number of commits they authored could be misleading. On pair programming and how commits can appear misleading, one student said:

*"And for example, team members commits, like I can tell some team members commit more than others which definitely bears root in some of us having more experience with writing code and, and committing more perhaps. But I don't think this is reflective of the method we used to code because we had pair programming. And so usually the most experienced person would, would sit on his computer or her computer and up to two people or at least one other would, would code alongside that person. And so usually it'd be the same person committing, so that could indicate why for example person two and person one have a lot of commits."*

Overall, most students expressed that the visualizations made sense to them

based on their impression of their group's work. Students whose commit-graphs saw an increase towards the end of a project or a sprint often reported that it reflected more work done. However, some students reported that their spikes were from many small changes to their documentation files. Furthermore, the data points make sense to the students because they view them in conjunction with their experiences of the group work.

When asked what wrong assumptions an outsider, e.g., an educator, could make about their group, all students shared the same concern — it does not capture all aspects of group work. All students of TDT4140 expressed that the dashboard gave the impression that their group had significant differences in how much each member contributed because it does not account for pair programming or other project-related work (e.g., report writing, planning). Almost all IT2810 students expressed similar concerns. Some mentioning that they spent the first week of each project learning the technologies before working directly on the project. Others mentioned that much of the work involved in development is not quantifiable in GitLab, such as meeting and discussing how to tackle the tasks at hand.

#### **5.2.4 As a mirroring tool promoting self-reflection**

As a mirroring tool, the dashboard aims to aid students in self-reflection and provide new insights into their group's performance. This section will present how the dashboard served its purpose and present some statements by students.

When asked if they were made aware of anything new, a handful of students replied that they had become more aware of Git as a tool and the correlation (or lack of) correlation between commits and lines. Others noted that they got a better understanding of why the gitignore-file is essential. Summarizing this perfectly, one student said:

*"I think that this graph right here with the removed code lines was surprising because I don't know if some of the large ones are node module files, but if they are not, it seems like we removed more than we wrote. And that seems a little bit off."*

Because the dashboard was anonymized and only tested on students using data from development in a group project, most of the new reflections and insights of students relate to their impression of how their group or team members have worked. Students reflected on data in light of when their group worked, how they divided work and more. As a tool to help their group tailor their sprints for their members, one student said:

*"Maybe a better understanding of how the group worked and a better understanding on how for the next sprint, how we could optimize to do better the group. The diagram*

*showing the projects periods commits sorted by days could be used to know when the group likes to work, and used to optimize the group."*

Awareness and confidence have strong ties to self-reflection. During development, one gets an impression of how individuals in a group and the group work. Often based on communication and observations, this impression potentially misrepresents reality. A student whose impression of their group changed during the interview reflected on it, saying:

*"It helps me connect the dots, seeing it all together. I definitely learned that maybe my perception of how and when, and how much other people work isn't, or usually probably isn't as accurate of how it actually is. So it's definitely fascinating to see it with your own eyes and not just guess based on assumptions."*

Other students expressed that viewing their data visualized changed their impression of their contribution to the project. One student claiming to be the least experienced member of their group expressed:

*"Now looking at project four, like I felt that I was contributing very little to the project sometimes, but looking at it now, it's like, I'm definitely had fewer commits or whatever than others, but it's not that uneven than I maybe felt like it was at sometimes, which is nice to see that it wasn't just yeah. And I think it's interesting to see how much more evenly distributed this commits distributed by members is in the last project compared to the first project. I think it was like, it's good to see that we'd gotten much better at distributing work."*

In contrast, several students expressed that the dashboard served to confirm notions they had of their group, one student quoted below:

*"I'm not sure if I was made aware of anything you per se, but I'm more confident in what I thought about the project. And also it was a new way to visualize things, and it made it more obvious. The things that I had a sense about when it came for workload and who did what and how the projects went, and so on. So it was a great tool for that I would say."*

All students were told to use the dashboard with their project experiences in mind. However, some students experienced the dashboard not to be useful. Although they agreed with the visualizations, they did not express any new insights, realizations or enjoyment, one student quoted below:

*"Åsmund: Were you made aware of anything new from the visualizations?"  
"Student: Not really, I think that I wasn't aware that someone used to like sit up to*



*two o'clock doesn't seem like it was often, but I wouldn't say I really learned anything new"*

The shortest interviews were with students who reflected little on the visualizations.

### **5.2.5 Anonymity of presented data**

The dashboard anonymized all data shown in graphs, swapping student names with non-identifying alternatives (e.g., "Person1", "Person2"). However, does this serve to hide student identities? Moreover, should the data be anonymized or tied to names? This section will present students' opinions on whether the data should be anonymized or not and whether they experienced it truly being anonymous or not. Name are only anonymized in the graphs. The dashboard listed members of a team, and the commit list showed author names.

Concerning whether the dashboard succeeded in anonymizing students, students' general opinion was that it did not. Of the 25 students interviewed, all but one said they could make out all or some of their team members from viewing the graphs. A more significant portion of IT2810 students said they could make themselves out and have an idea of the other team members (teams were 2-3 students), whereas most TDT4140 students said they could make out at least 1 or 2 students (teams were seven students).

Students were mainly of the impression that the dashboard should not hide students' names, although some also made arguments for anonymization. Some argued it should be anonymized expressed concern for grading and evaluation from educators based on the data. One student, when asked if the dashboard should be anonymous, expressed:

*"Yes. If you show it to a professor if it's affecting grade, like if he sees the graph and it's like, okay, this is really bad. I need to talk to your group, then, then maybe you can show the names, but I think it could just like affect the total impression of the people in the group, like person one here that has few commits. It doesn't mean that person didn't do anything they did a lot, it's just not shown here."*

Most students expressed themselves about anonymity in the context of using the dashboard as a student or a student group. One student argued that by showing the names, it would be easier to identify what times the groups should work; another that by removing names, it was hard to make sense of the data if they could not connect it to their impression of group members. A TDT4140 student expressed the following when asked if the dashboard would be more useful if showing names:

*"Yeah, for sure. I mean, it's just easier to relate the data to someone and then recall how they were working and then, you know, it makes more sense for them. And also just for the whole group too, if you weren't using it as like a 'you should contribute more sort of thing', but like maybe just to improve, it could be more useful."*

Several other students substantiated the argument, saying that including student names would reduce speculation on the data. Quoted below is a student who argued for including names based on improving group collaboration.

*"During the process, I would have the names here because we would easily be able to correct and make sufficient changes to our way of working. Like now after the project, and it is done and dusted, it is kind of interesting to see without the names and I see how, well I think adding the names would not remove anything to me, I would think other than removing like the speculation, but I would've done that in a way."*

Concerning the harmful effects of having students' contributions on display, some students were concerned about how students who contributed less would experience this transparency. Although the same data is available in GitLab, differences in data points are more apparent when aggregated. Quoted below is a student's suggestion to toggle whether names are shown or hidden:

*"Student: It would be actually a great option if you could have a tick box to choose if it's anonymous. Yeah. And then you can, let's say, then the project manager would go through, and people would discuss things. And I was thinking that I was the purple one or whatever. And then now looking at it, you're actually the tiny one, for example. So I guess it would be useful.*

*Åsmund: But you'd like the option to toggle the anonymity?*

*Student: Yes. But maybe not in a group like, or actually, yeah, it's fine. You have to be honest, but I mean, it depends on the group because if you are a group who gets along, and it feels like group chemistry is good. Yeah. I guess everyone would be fine with seeing their names, but I mean, it's not so nice being, even though I know who they are, I'm assuming it's not so nice to be the guy with two commits."*

Educators, on the other hand, were more divided on whether names should be available or not. Arguments were presented from the viewpoint of educators and students and varied accordingly. When assessing student groups, those for displaying names argued that data would be incomplete without names and would hinder their ability to provide feedback to students and student groups and assess how students have worked. One educator said the following:

*"As an educator who wants to keep track of how groups and students have worked, it would be half as interesting if names are missing. It is interesting to see who has done what."*

On the other hand, three educators expressed that the dashboard should hide names to ensure a fair assessment of students. They were concerned about how the dashboard could affect the fairness of their grading. An educator who taught a course of around thirty students expressed concern that seeing a student perform well or poorly compared to others might amplify their prejudices towards students.

One educator expressed that showing student names would improve their trust in the system and help validate the system's accuracy. Beyond this, the educator claimed that showing or hiding names would not change the value of the dashboard.

### **5.2.6 Would educators use the dashboard?**

When asked if the dashboard could be helpful to them, all 11 educators answered yes. Presented in this section are various use cases for the dashboard and concerns regarding use.

Seven of the educators expressed that they would be interested in using the dashboard's Advised Group Selection view to gain insight into how student groups perform throughout the semester. They expressed that they would use it to spot struggling student groups and decide what student groups to give attention to or contact. One educator, when asked to clarify, is quoted below:

*"It is easier to detect problems earlier. The worst is to detect that someone has not contributed just before a deadline. If this (the dashboard) can help me catch problems early and act on it, it would be very good. And it appears it can contribute to that."*

Another educator said the dashboard could provide a heartbeat into how students of their course were performing. Specifically, he could use the dashboard to quickly see how many student groups were doing poorly and act accordingly if the dashboard allowed for specifying what was regarded as doing good or bad (e.g., a *good* amount of merge requests and issues). Several instructors said they already consult GitLab to gain insight into student groups but that the same insight is available from using the dashboard. One educator is quoted below on the topic:

*"That is more useful than what GitLab offers out of the box. And I think it would pretty much cover the typical use cases that we currently use the data for. It's a bit easier having the dashboard because you don't have to do sort of a manual checking yourself. And the summary is also very useful for a quick overview of how*

*the different teams perform. So that would be useful to know during the semester, as well as at the end. [...] I would like to use dashboard like that in all my courses that use GitLab."*

In line with the theory on mirroring tools, most educators acknowledged the importance of knowing *what* information is displayed and using the dashboard accordingly, understanding that the locus of processing was with the user. All interviewees were informed that the data points are "dumb", to which one educator responded:

*"It is equally useful data, and I would think that in my topic, I could have approached some groups earlier and got to grips with the situation much earlier, so it did not escalate as much".*

Mirroring tools aim not to provide a conclusion but to help in decision-making and awareness in its user. For example, another educator made a reflection supporting the above quote and succinctly put into words how he would be careful and consult other data sources before deciding on the dashboard., quoted below:

*"If you focus on a thing from the system, you may miss something else. But it's not a concern because I'm aware of it. And in a classroom situation, I would use other measurements. But it is always dangerous that it can give us a skewed picture of the situation and not capture as much as we think we capture. [...] I'm not worried, but would be sober towards use."*

All educators answered what red flags they look for before contacting a student group, to which responses varied. For example, two educators whose courses had fewer than fifty students said they would contact groups if they had many red-faced indicators. In total, four educators said they would consider contacting groups that showed signs of not getting started with the project or showed signs of slow progression early on.

### **5.2.7 New features suggested**

This section presents new features suggested by students and instructors and the rationale behind them. Some were mentioned by interviewees when asked how they would visualize GitLab data points; others upon seeing the dashboard. All feature suggestions are listed in Table 5.4.

#### **Integration with GitHub**

Two educators said they used GitHub for their courses and queried what changes were needed to extend the dashboard to support visualizing repositories from GitHub

Where feature has impact	ID	Description of feature
Distributed Version Control Support	F1	Integration with GitHub and GitHub Classroom.
Advised Group Selection for educators	F2	Specify weights on metrics for aggregated performance indicator. E.g. commits are weighed less than issues and merge requests.
	F3	Specify discrete thresholds for colored faces to indicate performance. E.g. define red faces for the commit metric to be fewer than 10 commits one week into the project.
	F4	Specify percentiles used for colored faces to indicate performance. E.g. define red faces to be the bottom 5% and green faces to be top 10%.
	F5	A new metric to indicate equality of contribution among students. E.g. a numeric value or a pie-chart of commits among members.
	F6	A new metric to indicate consistency of work done. E.g. a numeric value or graph to show if commits are made consistently or sporadically.
	F7	More colored faces to show nuance between student groups. E.g. use a contiguous color space instead of only four colors.
	F8	Define red flags to be displayed for student groups. E.g. a red flag for student groups who have more than 200,000 lines or a negative amount of lines.
	F9	Add a slider for picking what dates of the project period to view groups. This would clarify when the project period begins and ends.
	F10	Functionality for educators to export data in numeric values in CSV or similar format. This would allow educators to perform their own analytics on the data.
	Changes to and new features for graphs	F11
F12		Development of commits and lines of code visualized with line graphs. This could provide easier insight into how individual students develop and contribute over the project period.
F13		Option to toggle whether names in graphs are shown or hidden.
F14		Option to specify date ranges for all graphs to view how the group has performed in a specific time period. This could be useful for viewing individual sprints.
F15		Option to view two student groups next to each other for comparison.
F16		Option to exclude certain commits and suppress large commits from doughnut graphs
F17		Support for co-authored commits. This could remedy the data loss of pair programming.

**Table 5.4:** Feature suggestions made by educators and students to improve the dashboard.

(Feature F1). The data points used in the dashboard have semantic equals in GitHub, although some variance in the specific attributes of each data structure is expected. Furthermore, the dashboard would need to support fetching data from GitHub and adapt to the differences in data structures. The rationale behind this extension is that the dashboard would be helpful to more institutions and educators and possibly integrated with GitHub Classroom.

### **Advised group selection for educators**

Several educators mentioned they would like more control over the Advised Group Selection page. The features mentioned by most educators were F2, F3, and F4, all somewhat overlapping in functionality. For example, F2 would allow educators to use the page more effectively for their courses, teaching version control with GitLab, monitoring an introductory course, or monitoring a more advanced software development course.

F3 and F4 are for educators who want more control over how the dashboard presents groups, specifically how it decides on groups who are doing well or poorly. The features could enable educators to allocate resources more efficiently to groups struggling and monitor the overall status of their courses more efficiently. F3 is closely related to F8 and would serve to notice educators of groups that require intervention or attention.

F5 and F6 introduce new metrics of interest to educators. For example, more educators expressed interest in picking up on groups that had free-loaders or under-performing students for both the purpose of individual grading and to help students who struggle. One educator said students complain every year that group members are not contributing, and using the dashboard could help him intervene earlier.

### **Changes to and new graph features**

A student suggested F12, explaining that viewing the development of commits in a line graph could more clearly display how students pick up their pace throughout the project period, e.g., students who are slow to start because they need to learn the tech stack. Furthermore, it could better illustrate if individual students or the group as a whole contribute evenly or sporadically.

F16 was suggested or requested by many students and educators alike who said the option to suppress large commits in the bar charts would be equally valuable for the doughnut graphs. The feature would prevent large commits from skewing individual students' pieces.

Although only one student reported having used Git's co-authoring functionality, almost all students expressed concern for the dashboard not accounting for pair programming. If the dashboard accounted for co-authoring and students used the dashboard, perhaps more students would use the co-authoring feature to log who has worked on a commit more accurately.

# Chapter 6

## Discussion

This section will discuss results in light of the research questions presented in Section 1. They will be viewed in correlation to the theory and previous work from Section 2.

### **6.1 What are students' perceptions on and experiences with using Git in an educational setting (RQ1)?**

The survey conducted to answer RQ1 serves both to verify previous research on using Git in software education and to probe into the relatively unexplored social implications of Git with GitLab in software development in teams. The survey results verify previous research that students enjoy using Git and that project work conveyed its benefits. Findings also support the decision to implement peer-assessment in programming courses. Finally, the results indicate that using Git has some implications on group dynamics and how students perceive their team members.

Looking at students' answers in Section 5.1 we quickly note the differences between the two student populations before discussing the survey. Students of IT2810, a fifth-semester course, are, for the most part, more experienced with both using Git and with web development compared to students of the fourth-semester course TDT4140. Not all students of TDT4140 choose to do web development, although a considerable amount do. The answers inform us that most respondents only have some previous experience. Thus, the questionnaire may provide insight into the learning process.

Section 5.1 present findings on students' perceptions on learning and using Git. These results suggest that many students experience Git to be both difficult to learn and to understand, supporting the claims of Lawrance, Jung, and Wiseman [8]. Furthermore, comparing the results from the less experienced students of TDT4140 to those of IT2810 students suggests that students' understanding of Git improves through using Git, a claim supported by the large number of students who reported improving at using Git throughout their course. A limitation is that while all students



have to enroll in TDT4140 in their fourth semester, they must enroll in IT2810 over other less programming intensive courses. Thus, in general, students of IT2810 may be more enthusiastic about programming and Git.

The results fit with the premonition that Git's benefits are made clear to students through software development in teams. Students answer in unison (>90%) that Git is necessary, simplifies working distributed, and simplifies tracking the progress of the codebase in project work. From both courses, more than 90% of students agree that Git aids in their work and that it simplifies both working with others and the development process. The results agree with those of Lawrance, Jung, and Wiseman [8] and support the decision to use Git for version control in software education.

Concerning when to introduce Git to students, the results suggest that students have not had an entirely smooth experience using it, or in the case of TDT4140, learning it. More than half of the students report that Git has been at times frustrating to use, and a worrying 18.6% (N=8) of TDT4140 students agreed that Git had been demotivating to use. Although results show that many students learn how Git works by using Git, many students wished that Git had been introduced earlier on in their studies. However, not all bad, 62.5% of IT2810 students and 45.6% of TDT4140 students agreed that Git had been fun to use. The results suggest that learning institutions must consider when to introduce Git, possibly introducing it earlier.

In contrast to introductory programming courses where students mainly program alone, both TDT4140 and IT2810 have students work in teams on the same code base. In other words, repositories are transparent, and all code is traceable back to its author. Viewing students' responses in Section 5.1 we observe that students are widely OK with their code being available to view by strangers, team members, and course staff. Furthermore, Git's transparency is not an aspect many students experience as a negative thing. Although this does not raise any new concerns or questions, it assures that it is not a problematic area.

Figure 5.1 also provides insight into the topic of receiving and providing feedback on code. We first look into students' experiences receiving feedback and see that the majority of students are comfortable receiving feedback (86.7% (N=37) of TDT4140 and 91.7% (N=44) of IT2810) and that few students take the feedback they receive personally. The statements aimed to inquire if students were perhaps offended by feedback on their code, but this does not appear to be the case for most students. 12.5% (N=6) of IT2810 students, however, agree that they do take the feedback personally (compared to only 7% (N=3) of TDT4140), perhaps because they received harsh feedback or because they feel pride in their code.

In IT2810, all students had to provide feedback to other students anonymously, and 85.4% (N=41) agreed that being anonymous made it more comfortable. The author suggests a correlation between this statement and the statement "*I found it hard to assess the quality of code written by others*", to which 54.1% (N=26) of students agreed. Thus, the large number of students who preferred anonymously giving feedback may be a direct result of students doubting the feedback they provide and their capabilities to assess code quality. However, although many students found it hard, 64.6% (N=31) of IT2810 students agreed that assessing code written by others helped them improve their programming. The results support the background theory of Section 2.4 which suggests that peer-assessment improved the quality of subsequent work.

TDT4140 had no option to provide anonymous feedback on code, so we ignore their responses. More so than IT2810 students, 74.4% (N=32) agreed it was difficult to assess the quality of code written by others. The difference can be correlated to the larger number of students with little experience with web development, suggesting a correlation between comfort providing feedback and programming experience. For the statement "*I got better at programming by assessing code written by others*", six students answered that the statement did not apply to them. This makes sense when we consider 10 students said they did not provide feedback on merge requests (See Figure 5.3). Of the 37 students it applied to, 70.3% (N=26) agreed their programming skills improved by assessing the code of others.

Further, supporting the results on the benefits of peer-assessment, 58% (N=28) of IT2810 and 51.2% (N=22) of TDT4140 students received feedback that helped them improve their code, and 64.6% (N=28) and 60.4% (N=22) respectively experienced receiving feedback as helpful. Therefore, when considering whether to instruct students to do peer-assessment, the results should be weighed in. The results suggest that implementing peer assessment will have positive results on the learning outcome of many students.

In order to answer if any social implications follow from using Git and GitLab, we look to the results of Section 5.1. Because all answers collected are from individuals and not from entire student groups, we need to consider that any findings do not necessarily represent how entire groups feel; it could just be individuals of a group. We are possibly tracing back to our discussion on Git's transparency and observe that 81.4% (N=38) of TDT4140 and 75% (N=36) of IT2810 students agree on differences in skill level become more apparent when using Git. Although this means little by itself, it may be a factor for why we see the responses discussed next.

The statement "*Working with GitLab makes it worse to be on a lower skill level than the team*" saw 27.1% (N=13) of IT2810 students and 41.9% (N=18) of TDT4140

students agreeing, but for IT2810, we saw 29.2% (N=14) disagreeing, and 37.5% (N=18) neither agree nor disagree. Fewer TDT4140 students disagree (20.9% (N=9)) and 32.6% (N=14) neither agree nor disagree. These results do not give any clear indicators that GitLab affects collaboration between students of varying skills. The questionnaire also had the near inverse statement "*GitLab makes it less difficult to work with someone on a lower skill level than yourself*". For TDT4140, where students worked in groups of seven, only 25.6% (N=11) agreed, and 27.9% (N=12) disagreed. However, most students (44.2% (N=19)) neither agreed nor disagreed, suggesting that GitLab does not have a significant impact. A larger portion, 41.7% (N=20) of IT2810, agree with the statement, yet still 37.5% neither disagree nor agree and 14.6% disagree. The differences in answers may be associated with the differences in group size and the dynamics of the project work of the two courses. In groups of three (IT2810), it may be easier to pay attention to and provide help to the less skilled member than in groups of seven (TDT4140), but this is only speculation.

45.8% (N=22) of IT2810 students agree with the statement "*Git results in the best members hijacking the development process (Rewriting a lot of code, doing all the work themselves etc.)*". TDT4140 saw close to a third of students agreeing with 37.2% (N=16). IT2810 grades students on their end product, possibly contributing to this phenomenon, but we also see a notable amount in TDT4140. From Figure 5.3 we see a large amount of students from both courses used issues to separate work tasks. The case may also be that the better team members finish their issues more quickly and move on to the next one, completing more issues than other members.

Turning our attention to the less skilled team members, we look at results on the statement "*Development with GitLab turn team members who accomplish less into scapegoats*". We see that 51.2% (N=22) of TDT4140 students disagree with the statement to 45.8% of IT2810 students. However, we also see that every fourth TDT4140 students and every third IT2810 student agree with the statement. Thus, it is probably not the case of all student groups where members are on different skill levels, but it appears that many students do feel it is the case. It could be that more skilled students feel this way about their less skilled members, or it could be that the less skilled members *think* their team members feel that way. Either way, it suggests a harmful phenomenon is present in multiple student groups.

A consideration to be made is that the author has attempted to attribute GitLab with specific implications on group work. When using statements in a questionnaire, we risk putting words into the mouths of our respondents. Thus, we may not have seen the same results if students answered open-ended questions about using GitLab. There may be other aspects of programming that would more accurately be to blame than version control with Git, but for this survey, GitLab has been the focus.

Further research is needed to reliably say how working with Git and GitLab affects the social aspects of student groups. However, the above results can aid researchers in approaching the research topic.

## 6.2 How can GitLab's data points be visualized in a mirroring tool (RQ2)?

A dashboard solution to examine visualizing GitLab's data points has been implemented to meet the needs of both educators and students. It visualizes several data points of GitLab repositories to provide insight into work patterns and workflow. In addition, it offers educators functionality to order student groups based on quantitative GitLab data to make advised decisions on student groups.

The results of interviews and demonstrations of the implemented dashboard on students and educators support the visualizations developed for this thesis. Feedback from interviewees shows that the visualizations can be helpful to students and educators to provide insight into work and work patterns when viewed in retrospect. Data points displayed in a time series and allowing for comparing student contributions were well-received by both students and educators. Furthermore, the dashboard design was perceived to have high affordance and conveyed data to students and educators in a way that made sense.

### Visualizing data points

The background theory on monitoring in computer-supported collaborative learning and mirroring tools presented in Section 2.6 and Section 2.7 discusses how data points generated by students in a learning situation can be aggregated and reflected back to students or educators to provide new insights. This research has examined four data points: commits, issues, merge requests, and code lines extracted from commits. The four metrics were well-received by both students and educators and perceived to be the four most relevant data points from GitLab.

Some interviewees suggested or expected branches to be visualized similarly to how it is presented in Figure 2.4 and others suggested showing the number of branches. However, merge requests may more accurately depict progression in the codebase. Furthermore, graphs are not affected by *"dead"* branches, branches that remain unmerged. However, using branches in the dashboard can be helpful to provide other insights, perhaps to see if a student group abandons many branches. It can also provide insight into whether students are rebasing their commits before merging or not, an exciting aspect of evaluating the use of Git.

Line graphs visualized issues (see Figure 4.5), displaying the number of open issues on a given date, as well as the sum of closed issues on a given date. Viewing this graph proved to give insight into the amount of work student groups had done. However, varying habits regarding the use of issues mean two well-performing groups can appear differently. Furthermore, this also means a well-performing group can appear worse than a poorly performing group if naively viewing the dashboard. To substantiate this, we exemplify two groups, one performing well, but which opens fewer significant issues (e.g., "implement backend"), and a poorly performing group that opens many tiny issues (e.g., "fix typo in header"). The group with more minor issues will close more issues and appear to show more progress than the otherwise well-performing group.

An educator noted that having issues visualized could impact how he structured his course and allow for more rapid insight into how far students are in the course's project. More specifically, he suggested that all students open a set of issues at the start of the project, representing the programming goals of the project. Thus, as students close their issues, the graph would act as an up-to-date burn-down chart requiring little to no extra energy to keep it updated. Furthermore, the data structure for issues returned from the API contains a list of labels per issue. If filtering on these issues, the dashboard can group issues on topic and display them in a burn-down chart manner (e.g., labels could include "database", "backend", "frontend").

The dashboard ordered neither merge requests nor issues by author. Thus, reflections by students on the metrics concerned their group's work and offered no comparison of students. However, showing both metrics ordered by author can be valuable to students and educators to provide nuance to members' contributions. For example, quantifying the metrics in doughnut charts efficiently shows how many data points of each metric students' created. Displaying issues and merge requests in a line chart but ordered by students may convey when students finish their work, be it in bursts or evenly throughout the project period. In addition, results from other graphs ordering data by students show that both educators and students are interested in graphs that display student contributions per student.

Commits were visualized both on a project basis and separated by authors. The interviews with students proved that commits as a data point were a good decision for the dashboard. Visualized in a time series bar chart distributed by dates, days of the week, or hours of the day, they provided both students and instructors with insight into when groups worked. However, students were concerned that viewing commits distributed by team members failed to accurately capture how work was distributed by members, especially regarding pair programming. A potential remedy was to use Git's co-authoring functionality, but it is tedious to use, and convincing students to use this feature can be difficult.

Students perceived commits viewed on a project basis to capture the rough features of how they structured their programming with increases in the number of commits in a period, often correlating with increases in the work produced. However, other students reported that some of their spikes in commits were related to minor changes to documentation and spelling or attempting to fix bugs. In other words, commit numbers could run rampant and give a viewer the wrong impression. In addition, although students could consult the commit list to gain insight into the sizes of commits, only a few students used it. As a remedy, displaying both the average size of commits and the median size of commits, both per project and per member, can, to a degree, convey commit habits.

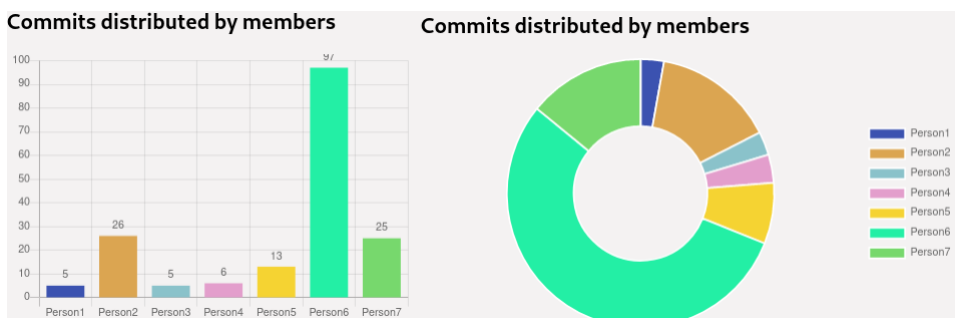
An interesting suggestion made by a student was to visualize commits per member in a line graph, showing how the amount of commits by a student develops over the project period. The graph could provide insight into how students' individual *pace* develop over the project period with a lower cognitive load than reading bar charts to extract the same information. An observed phenomenon for multiple TDT4140 students was that the bar graphs showed fewer commits early on in the project, and more as the member became confident and experienced with the technology. One student observed this when isolating commits of one member (by removing other members). They commented that they understood who the student was and confirmed that he had had to learn the technology for the project.

Feedback from interviews and demonstrations suggests that visualizing when students add code lines to the repository was less valuable than expected. Code lines were poor indicators of work done, were often skewed in repositories without gitignore-files, and merged commits worked to double numbers gave a false impression of the amount of work done. The option to suppress larger commits to prevent the graph from being skewed was well-received and helped give insight into when and the amount of work done by members. In addition, either excluding merge commits, or counting merge commits alone could prevent lines from being counted twice.

Although none of the interviewees commented on using the word code lines, future use of similar graphs should be unambiguous in its wording. All commits have added and removed lines, namely additions and deletions, but these are not restricted to code only. For example, SVG files have many lines, yet they are not *code* and would skew this metric. The metric should be used carefully and works the best if users of the dashboard should be aware of the amount of generated code in a repository to make use of this metric better.

The dashboard included doughnut graphs displaying the number of commits and lines (additions, deletions, and the difference between the two) per member to compare contributions by students. For smaller student groups like in IT2810, they

efficiently presented relative contributions. However, for TDT4140, where student groups were of seven and contribution amounts were more varied, it was challenging to view students with lower numbers. Therefore, replacing the graphs with bar charts, which make it easier to pin down similarly sized small contributions, can be worthwhile (See Figure 6.1).



**Figure 6.1:** Bar chart and doughnut graph both displaying team members' commits.

### Displaying student names

Data points can be presented either with students' names displayed or hidden, but students will identify some members even if names are not displayed. Furthermore, most students said that displaying names would allow students to better act on the visualized data and remove the need to guess who is who. On the other hand, some educators were more inclined to use the dashboard with anonymized names to prevent unfair evaluation and that previous experiences would students would color their impression. Adding the option to toggle anonymity will fit both needs, perhaps controlled by an educator. All interviews with students were one-on-one and did not capture how students who seemingly contributed less would react to seeing this with their group. Some educators were concerned that those students would not enjoy seeing themselves put on display, but further research is needed to establish how student groups experience this.

### 6.3 What does a mirroring tool on GitLab data offer (RQ3)?

The results of interviews and demonstrations of the implemented dashboard on students and educators indicate that the developed visualizations of data points generated when using Git can be helpful to students to improve self-reflection, planning in groups, and make aware of work patterns. However, results show that individuals viewing the dashboard in retrospect made few new realizations.

The study suggests that the dashboard is helpful for educators to rapidly gain insight into how students are performing in a course and spot underperforming students. More importantly, the interviews with students and educators have solidified the need for understanding its data points and critical thinking when viewing the dashboard. Although providing some insight into how students work, Git's data points on its own do not do accurately capture all aspects of software development in groups.

### **As a mirroring tool for students**

Section 2.7 presents theory and research on mirroring tools. A point of mirroring tools is enhancing self-awareness of actions and behavior to improve upon for future work. The results of Section 5.2.4 show that the dashboard promotes self-reflection in some students. Furthermore, the results indicate that group awareness can increase from viewing the dashboard. Group awareness is helpful when structuring workdays in a project. Results expectedly varied from student to student.

The interview results indicate that the dashboard is helpful to student groups working on multiple projects and sprints. Interviewees said the dashboard visualized times when members worked and that consulting it can optimize their work schedule. Furthermore, the dashboard can help students detect and help team members struggling. However, these results are from reflections of individual students, not student groups. Therefore, the dashboard has to be available for use in a course to get feedback from student groups. Alternatively, interviews with groups can provide more insight into its helpfulness for student groups.

In interviews, students mainly reflected on their group, not on their contributions. Although some students identified themselves and reflected on their data points, not all students did this because the dashboard hid names. Those who did make out themselves reflected on the data regarding how they remembered their project or projects. Interestingly, some students made new realizations or changed their perceptions of their contributions to the project. More students would likely reflect on their data if the dashboard displayed their names. However, to display only some student names would require either authentication (for example, through FEIDE) or manually choosing what names to display and hide.

A few students who made out themselves did not reflect on their data. Some focused more on their group, and others continued after acknowledging that their data *looked right*. Students would likely reflect more on their data if instructed to do so. However, when testing the dashboard, the interviewee was in the driver's seat. Unsurprisingly, students were more interested in viewing their group than reflecting on their data. A probable cause for this phenomenon is the availability of data that



was previously not easily accessible to them. However, students often compared their contributions to their team members.

Students reflected on the visualizations with their project work in mind, but student responses varied. However, several patterns emerged in students who got less out of viewing the dashboard. Some navigated through the system quickly, and other interviewees were confused or overwhelmed by the dashboard. For example, students who quickly navigated through the dashboard and those who explored fewer features reflected less. Overwhelmed or confused students also made fewer reflections, suggesting that using more time to explain the system can help some students. Interviewees who lacked an understanding of Git beyond knowing how to use it made fewer comparisons to their work. Mazza and Dimitrova [50] and Voyiatzaki and Avouris [52] made similar findings on mirroring tools for teachers.

The majority of students expressed that the dashboard was helpful or exciting; interviewees who took more time to examine and talk about their data got more out of the dashboard. Furthermore, students who understood Git had a more modest and sensible relationship to the visualizations. Finally, many students reported more confidence in their impression of their group and their work after viewing the dashboard. Although not particularly useful for the course they have completed, it can be helpful for future courses. Two valuable outcomes come to mind. Firstly, students who realized they had wrong impressions will know to be more careful. Secondly, confirmation on impressions can help students act sooner and more confidently if problematic behavior occurs.

Although multiple students said they became more confident from viewing the dashboard, some students were very eager to accept the visualizations and data points as authentic representations of their work. Nonetheless, this does not imply that students were wrong in their reflections, but expectations were that more students would mention the incompleteness of the data. In addition, it is terrible if the dashboard falsely changes a student's perception of themselves or their group. For example, some students showed little or no skepticism towards using the code lines metric. If code lines are in the millions, the validity of doughnut graphs decreases, even if they indicate a slight imbalance between members. Thus, educators should explain the dashboard to students before using it to prevent misconceptions.

It was expected that students would be skeptic towards using Git metrics. Less expectedly, the results indicate that students' understanding of Git improved from viewing the dashboard. Thus, the dashboard can be valuable in courses teaching Git. For example, the lack of consistency between code lines and commits is apparent in the graphs, a correlation some interviewees admitted not to know. In its current form, the dashboard does not help users understand its data. However, it can aid

students in developing a holistic mental model of good Git practices, for instance, displaying the significance of gitignore-files.

An alternative use case of the dashboard would be to provide students with a report of their data to promote self-reflection. For example, when students do sprint retrospectives like in TDT4140. Viewing a report visualizing their previous sprint, they can reflect on it, become aware of collaboration patterns, and more. Unfortunately, the dashboard in its current state cannot offer this report, but adding date ranges to select what data to show would suffice. Furthermore, comparing two or more sprints, it is easier to see how the group develops.

**To summarize the mirroring tool for students** we conclude that the dashboard can, to some degree, capture how much individual students contribute, when they contribute and how a group works over a period. It also captures Git habits, and provides some students with new insights and reflections on their work. Furthermore, it served to increase confidence in students' impression of their group work.

## **Educator's view**

Results from interviews and demonstrations with educators indicate that a mirroring tool visualizing GitLab data is valuable in multiple ways. For example, educators are interested in monitoring their courses' overall health, gaining insight into individual student groups, and making informed decisions on spending their resources.

Firstly we discuss how the dashboard helps educators monitor the health of their courses. The advised group selection page displays four metrics indicating student group performance, plus a fifth overall metric. The current implementation partitioned student groups into quartiles per metric based on how many of each data point is in their repositories. Educators were interested in specifying percentiles themselves to quickly view poorly performing student groups, e.g., the bottom five percent of overall performance or a specific metric. Extending the dashboard to support this would increase its value for little development effort. Git's incomplete metrics will lead to some groups appearing to do worse, but results cannot conclusively say if this is a problem. However, even in a course with 100 student groups, contacting the bottom five percent would only require contacting five student groups.

Nevertheless, the dashboard is only a mirroring tool, and acting solely on it is not suggested. Instead, educators should seek more information from students' repositories before making deciding whether to contact a group or not. Thus, the advised group selection serves to inform but not instruct.

Many educators suggested that the dashboard gave educators control of defining what amounts of each metric would show as what faces. E.g., more than ten commits

in the first week of the project would be a green face; fewer than two would be a red face. Implementing this, the advised group selection page extends into the domain of metacognitive tools, discussed in Section 2.6. The dashboard, if comparing students' interaction with the desired model, could notify of discrepancies. Educators would still have to process the data before acting, but it can help educators more efficiently distribute their resources.

Furthermore, this approach is less naive than viewing groups based only on percentiles. For example, if all 100 student groups do well, some would appear to be doing poorly if colored faces represent percentiles. Lastly, it gives the user more control, making the dashboard more valuable to a variety of courses.

Regarding the concern for students misinterpreting the dashboard, educators doing so would be even worse. Luckily, all interviewed educators maintained a sensible approach to the dashboard. They knew its data was imperfect and does not capture all aspects of project work. When introducing the dashboard to educators, the data was introduced as *dumb*, which may have influenced how educators view it. Nonetheless, educators must be aware of the limitation of Git data and the dashboard to prevent perceiving students wrong.

In its current state, the dashboard would simplify getting insight into student groups. Some educators said they consulted GitLab to get insight but said this was both tedious and slow. Furthermore, viewing individual data points like in GitLab makes it more challenging to see the whole picture.

**To summarize the mirroring tool for educators** we conclude that it captures student habits and work, which is helpful when teaching and grading Git use. Furthermore, it can roughly capture how a student group completes a project, whether they work consistently or do most of the work near the end. Finally, the advised group selection page helps educators identify student groups that need extra attention and gives a heartbeat indication of a course's health.

## **Making qualitative assumptions from the dashboard**

Assumptions on student groups were made before interviews to quantify what the dashboard conveys about student groups. Section 5.2.1 presents the results of this effort, which suggests that it is difficult to make accurate assumptions on student groups based only on how their visualizations appear. For example, for the attribute M6 — *The group worked in planned spaces of time (e.g., 8-13 Mondays)* the accuracy for both courses was >88%. However, attributing *all* groups with M6 would see a higher accuracy. Thus, the validity of drawing this assumption based on the dashboard is doubtful.

Results for *M7 — One group member was responsible for setting up the project(s)* shows that the dashboard fails to capture if more than one student participated in the setup of the project. In addition, results show that eight groups in total were falsely attributed with M7 and that five groups were falsely not attributed. The results suggest avoiding using the dashboard to draw assumptions regarding matters where two or more students can be *hidden* behind the activity of one student.

Similarly, *M1 — Some members have contributed more than others on the programming* and *M8 — Everyone in the group contributed an even amount in the programming*, saw high accuracies for TDT4140. However, the high occurrence of the two metrics may result from the group size of seven instead of the dashboard's ability to convey this information. To substantiate this claim, accuracies for IT2810 were only 66.7% and 55.6%, respectively.

*M2 — Some members have been free-loaders on the programming* saw inferior results for TDT4140 at only 30.8% accuracy, worse than a coin toss. With groups of seven in a course where students do a lot of pair programming, it is not easy to accurately assess whether students with fewer commits and lines have contributed less or just pair program on another computer. Results appear better for IT2810 with an accuracy of 66.7%, but no students attributed their group with this attribute. Thus, the three groups attributed with M2 were wrong assumptions. The latter results suggest that in student groups of three, free-loaders are more rarely present. However, students actively enrolled in IT2810, and many student groups knew each other beforehand, which both likely contributed to the low amount of free-loaders recorded.

When students attributed their groups at the end of their interviews, some students were hesitant to attribute their group or members with negative attributes, possibly resulting in more false negatives. Furthermore, the absolute wording of M3, M4, and M5, which regarded *every* exercise/sprint, in conjunction with inconsistencies in how groups were attributed and how students attributed their groups, may affect the results. However, in the interviews with students, many students pointed out both periods of consistent work and the occurrence of skippertak. Attribute *M5 — The group started working early in each exercise/sprint* was more challenging to assess because the dashboard does not capture work done outside of writing code, such as planning an exercise or sprint.

For attributes *M3 — The group has worked evenly through each exercise/sprint* and *M4 — The group has done skippertak every exercise/sprint*, assumptions made on individual projects for IT2810 were more often were correct, but the numbers do not show this. For similar experiments in the future, if attributes M3 and M4 are to be measured, they should instead concern specific exercises (or sprints) to give

more accurate results. If anything, the results on making qualitative assumptions should support the above claim that one must be aware of the dashboard's limitations when using it. It fails to capture all aspects of the development process and does not provide an outsider with a window into the composition or performance of a student group. Future research, perhaps using guided decision-making, is needed to determine if reliable assumptions are possible from viewing the dashboard alone and how to draw them.

## 6.4 Ethical concerns

The mirroring tool enables educators to monitor student activity which also raises ethical concerns. Data points are generated by students with no effort beyond using Git, and students do not actively consent to their data being mined, aggregated and interpreted. An essay is graded on its content, not how the student wrote it. Educators must decide if they grade an end product or a process, and if the insight provided crosses any boundaries.

The mirroring tool provides insight into when during the day students are active. Students who work during the day pose little concern, but students whose working hours extend beyond midnight may find it intrusive to be put on display. Furthermore, this insight might provide users with signs of larger problems that students do not want displayed. A consideration to be made is what data students experience as sensitive and adapt the dashboard accordingly.

The above concerns of monitoring single students arise because the mirroring tool allows for viewing contributions by individual students. Educators need to question how the value added from viewing individual students' contributions compares to privacy concerns of tracking individuals.

## 6.5 Validity of results

This study looked at two courses where students do software development. Although both courses are standard in their approaches to software development, results may differ for other courses. Furthermore, the two courses are from the same department and should not be considered two isolated cases. In addition, the students that participated in this study perform well both nationally and internationally. Thus, results may vary for student populations that perform less well.

All questionnaire responses and student interviews represent single students' impressions and experiences. It is not easy to assess which students from the student populations answered the questionnaire. There is always a possibility that a part of the student population is more represented, e.g., better-performing students, which

will impact how representative the aggregated answers are. Furthermore, not all students from the two courses answered the survey, which reduces the generalizability of the results.

Similarly, for student interviews, only one student per group was interviewed. Whether the interviewee was more or less skilled than their members is not collected, although some students did say so. Nonetheless, students only represent their impression of their group's work. Thus, a bias will persist when discussing what the dashboard says about their group.

The recruitment phase for student interviews may also affect the validity of results. For example, interviewees received food and a soft drink for participating, which may also impact what students sign up for participation. In addition, interviews took place when most students have exams, and for IT2810, interviews were a semester after they completed their course. Both of these factors likely affected how many students participated.

# Chapter 7

## Conclusion and future work

This thesis explored the implications on group dynamics and the social aspect of project work when using Git. In addition, a prototype mirroring tool using GitLab data was developed to investigate the data's potential when shown to students and educators. Ninety-one students answered an online questionnaire about using Git, and semi-structured interviews and demonstrations were conducted with 24 students and 11 educators to collect data on the mirroring tool.

This master thesis makes two contributions. Firstly, to research on using version control with Git in software development courses. This research aimed to support faculties' decision to use Git in software education and to explore its social and interpersonal impact on students and student groups. Based on the results of the conducted survey, it can be concluded that faculties should continue to use Git in software development education and that it is helpful to students. Furthermore, the results indicate that some students experience Git to have negative effects on some students' experience.

The survey results highlight potential trouble areas that no other researchers have investigated to the author's knowledge. However, they also support the current use of Git in software development courses. Quantitative studies fail to provide conclusive evidence of what causes a problem and how to fix a problem. Nevertheless, the survey clearly shows students want Git introduced earlier and that many students experience Git to be challenging to learn and, at times, frustrating to use. A theory is that students who struggle to learn Git independently will have a worse experience using the technology overall.

Future work on the results from the survey should further explore how students are affected by Git and GitLab in group projects — performing a more qualitative study to understand better what causes students to have worse experiences. More research will substantiate the claim that Git and GitLab affect student group dynamics and provide insight for educators to remedy the problem.

The thesis' second contribution is developing and testing a mirroring tool using data points from GitLab. In 24 student and 11 educator interviews with demonstrations, the research gathered feedback to assess if a mirroring tool on GitLab is helpful. Based on a quantitative and qualitative analysis of the collected data, we conclude that the mirroring tool can support both students and educators in a learning situation. The results show that GitLab data does not capture all aspects of development and its quantitative data requires human processing to make assumptions.

The dashboard results help future research select Git data points and choose how to visualize them. Many new features are described based on student and educator interviews which researchers can consult to design a dashboard to develop a similar or more extensive mirroring tool. The results describe what students and educators make of the different data points for self-reflection for students and monitoring and reflection for educators.

Impressions and results recorded from student interviews are from students who had no idea what to expect of the dashboard. In 15 minutes, students were introduced, had to understand, and then reflect on the dashboard. In other words, there was little time to learn to fly. The students who got less out of the dashboard might have gotten more out of it if given more time. Similarly, interviewees could provide even more insight into the mirroring tool's pros and cons if interviews lasted longer. However, the majority of students quickly understood the dashboard, speaking to its affordance.

Results from educator interviews show that the data points are helpful to monitor student activity and entire courses. However, educators gained retrospective insight into courses they had no relationship. Presenting educators with their courses can provide new insight. Future research should employ the dashboard in an active course to assess its helpfulness to educators. Topics of interest are, amongst others, resource management, learning outcome of students struggling, and course health. Researchers have to choose whether to inform students of the monitoring or not, which can affect results.

Future work on the mirroring tool should aim to make it available and practical to use. The current implementation manually populates the database and does not stay up to date in an active course. Containerizing the application using Docker and supporting authentication with GitLab will make the dashboard helpful with less setup required. With authentication in place, a logical step is to support automatically querying for new data, e.g., every twelve hours. These features will make the dashboard available to use for more educators. Furthermore, it will be a strong foundation for the development of a mirroring tool using Git data.

Due to the encouraging results on the mirroring tool from this study, continued



development to make it available and practical to use is planned. Furthermore, all code will become open source to encourage further development. It will be made available at <https://github.com/asmundh>.

# References

- [1] Joseph Feliciano, Margaret-Anne Storey, and Alexey Zagalsky. “Student Experiences Using GitHub in Software Engineering Courses: A Case Study”. In: *2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C)*. May 2016, pp. 422–431.
- [2] Courtney Hsing and Vanessa Gennarelli. “Using GitHub in the Classroom Predicts Student Learning Outcomes and Classroom Experiences: Findings from a Survey of Students and Teachers”. In: *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*. SIGCSE ’19. New York, NY, USA: Association for Computing Machinery, Feb. 2019, pp. 672–678. ISBN: 978-1-4503-5890-3. DOI: 10.1145/3287324.3287460.
- [3] Scott Chacon and Ben Straub. *Pro git*. Springer Nature, 2014.
- [4] 262588213843476. *Commit message guidelines*. en.
- [5] *Issue Boards | GitLab*.
- [6] Atlassian. *Git Feature Branch Workflow | Atlassian Git Tutorial*. en.
- [7] Atlassian. *Gitflow Workflow | Atlassian Git Tutorial*. en.
- [8] Joseph Lawrance, Seikyung Jung, and Charles Wiseman. “Git on the cloud in the classroom”. In: *Proceeding of the 44th ACM technical symposium on Computer science education*. SIGCSE ’13. New York, NY, USA: Association for Computing Machinery, Mar. 2013, pp. 639–644. ISBN: 978-1-4503-1868-6. DOI: 10.1145/2445196.2445386.
- [9] Lassi Haaranen and Teemu Lehtinen. “Teaching Git on the Side: Version Control System as a Course Platform”. en. In: *Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education - ITiCSE ’15*. Vilnius, Lithuania: ACM Press, 2015, pp. 87–92. ISBN: 978-1-4503-3440-2. DOI: 10.1145/2729094.2742608.
- [10] Joseph Feliciano. “Towards a Collaborative Learning Platform: The Use of GitHub in Computer Science and Software Engineering Courses”. en. Accepted: 2015-08-31T21:15:20Z ISSN: 1906-1927. Thesis. 2015.

- [11] J. Kelleher. “Employing git in the classroom”. In: *2014 World Congress on Computer Applications and Information Systems (WCCAIS)*. Jan. 2014, pp. 1–4. DOI: 10.1109/WCCAIS.2014.6916568.
- [12] Kimberly A. Freeman. “Attitudes toward Work in Project Groups as Predictors of Academic Performance”. en. In: *Small Group Research* 27.2 (May 1996). Publisher: SAGE Publications Inc, pp. 265–282. ISSN: 1046-4964. DOI: 10.1177/1046496496272004.
- [13] Guttorm Sindre et al. “Project-Based Learning in IT Education: Definitions and Qualities”. eng. In: *147-163* (2018). Accepted: 2018-06-13T13:17:13Z Publisher: Universitetsforlaget. ISSN: 1893-8981. DOI: 10.18261/ISSN.1893-8981-2018-02-06.
- [14] Filip Dochy et al. “Effects of problem-based learning: a meta-analysis”. en. In: *Learning and Instruction* 13.5 (Oct. 2003), pp. 533–568. ISSN: 0959-4752. DOI: 10.1016/S0959-4752(02)00025-7.
- [15] David W Johnson and Roger T Johnson. “Social Skills for Successful Group Work”. en. In: *EDUCATIONAL LEADERSHIP* (), p. 7.
- [16] L.J. Burnell, J.W. Priest, and J.B. Durrett. “Teaching distributed multidisciplinary software development”. In: *IEEE Software* 19.5 (Sept. 2002). Conference Name: IEEE Software, pp. 86–93. ISSN: 1937-4194. DOI: 10.1109/MS.2002.1032859.
- [17] Randall S. Hansen. “Benefits and Problems With Student Teams: Suggestions for Improving Team Projects”. In: *Journal of Education for Business* 82.1 (Sept. 2006). Publisher: Routledge \_eprint: <https://doi.org/10.3200/JOEB.82.1.11-19>, pp. 11–19. ISSN: 0883-2323. DOI: 10.3200/JOEB.82.1.11-19.
- [18] Petru L. Curşeu and Helen Pluut. “Student groups as learning entities: The effect of group diversity and teamwork quality on groups’ cognitive complexity”. In: *Studies in Higher Education* 38.1 (Feb. 2013). Publisher: Routledge \_eprint: <https://doi.org/10.1080/03075079.2011.565122>, pp. 87–103. ISSN: 0307-5079. DOI: 10.1080/03075079.2011.565122.
- [19] Charlie McDowell et al. “Pair programming improves student retention, confidence, and program quality”. In: *Communications of the ACM* 49.8 (Aug. 2006), pp. 90–95. ISSN: 0001-0782. DOI: 10.1145/1145287.1145293.
- [20] Daan van Knippenberg and Michaéla C. Schippers. “Work Group Diversity”. In: *Annual Review of Psychology* 58.1 (2007). \_eprint: <https://doi.org/10.1146/annurev.psych.58.110405.085546>, pp. 515–541. DOI: 10.1146/annurev.psych.58.110405.085546.
- [21] Carol L. Colbeck, Susan E. Campbell, and Stefani A. Bjorklund. “Grouping in the Dark”. In: *The Journal of Higher Education* 71.1 (Jan. 2000). Publisher: Routledge \_eprint: <https://doi.org/10.1080/00221546.2000.11780816>, pp. 60–83. ISSN: 0022-1546. DOI: 10.1080/00221546.2000.11780816.

- [22] Barbara Oakley et al. “Turning student groups into effective teams”. In: (2004). Publisher: Citeseer.
- [23] David Hall and Simone Buzwell. “The problem of free-riding in group projects: Looking beyond social loafing as reason for non-contribution”. en. In: *Active Learning in Higher Education* 14.1 (Mar. 2013). Publisher: SAGE Publications, pp. 37–49. ISSN: 1469-7874. DOI: 10.1177/1469787412467123.
- [24] Graham D. Hendry, Greg Ryan, and Jennifer Harris. “Group problems in problem-based learning”. In: *Medical Teacher* 25.6 (Nov. 2003). Publisher: Taylor & Francis Ltd, pp. 609–616. ISSN: 0142159X. DOI: 10.1080/0142159031000137427.
- [25] K. J. Topping et al. “Formative Peer Assessment of Academic Writing Between Postgraduate Students”. In: *Assessment & Evaluation in Higher Education* 25.2 (June 2000). Publisher: Routledge \_eprint: <https://doi.org/10.1080/713611428>, pp. 149–169. ISSN: 0260-2938. DOI: 10.1080/713611428.
- [26] D. Sluijsmans, F. Dochy, and G. Moerkerke. “Creating a Learning Environment by Using Self-, Peer- and Co-Assessment”. en. In: *Learning Environments Research* 1.3 (Oct. 1998), pp. 293–319. ISSN: 1573-1855. DOI: 10.1023/A:1009932704458.
- [27] Jirarat Sitthiworachart and Mike Joy. “Effective peer assessment for learning computer programming”. In: *ACM SIGCSE Bulletin* 36.3 (June 2004), pp. 122–126. ISSN: 0097-8418. DOI: 10.1145/1026487.1008030.
- [28] Mien Segers and Filip Dochy. “New Assessment Forms in Problem-based Learning: The value-added of the students’ perspective”. In: *Studies in Higher Education* 26.3 (Oct. 2001). Publisher: Routledge, pp. 327–343. ISSN: 0307-5079. DOI: 10.1080/03075070120076291.
- [29] Keith Topping. “Peer Assessment Between Students in Colleges and Universities”. en. In: *Review of Educational Research* 68.3 (Sept. 1998). Publisher: American Educational Research Association, pp. 249–276. ISSN: 0034-6543. DOI: 10.3102/00346543068003249.
- [30] Paul Orsmond, Stephen Merry, and Kevin Reiling. “The Use of Exemplars and Formative Feedback when Using Student Derived Marking Criteria in Peer and Self-assessment”. In: *Assessment & Evaluation in Higher Education* 27 (Aug. 2002), pp. 309–323. DOI: 10.1080/0260293022000001337.
- [31] Thu Thuy Vu and Gloria Dall’Alba. “Students’ experience of peer assessment in a professional course”. In: *Assessment & Evaluation in Higher Education* 32.5 (Oct. 2007). Publisher: Routledge \_eprint: <https://doi.org/10.1080/02602930601116896>, pp. 541–556. ISSN: 0260-2938. DOI: 10.1080/02602930601116896.
- [32] J. Sitthiworachart and M. Joy. “Web-based peer assessment in learning computer programming”. In: *Proceedings 3rd IEEE International Conference on Advanced Technologies*. July 2003, pp. 180–184. DOI: 10.1109/ICALT.2003.1215052.

- [33] George Siemens. “Learning Analytics: The Emergence of a Discipline”. en. In: *American Behavioral Scientist* 57.10 (Oct. 2013). Publisher: SAGE Publications Inc, pp. 1380–1400. ISSN: 0002-7642. DOI: 10.1177/0002764213498851.
- [34] *Big Data is a Big Deal*. en. Mar. 2012.
- [35] Olga Viberg et al. “The current landscape of learning analytics in higher education”. en. In: *Computers in Human Behavior* 89 (Dec. 2018), pp. 98–110. ISSN: 0747-5632. DOI: 10.1016/j.chb.2018.07.027.
- [36] Kimberly E. Arnold and Matthew D. Pistilli. “Course signals at Purdue: using learning analytics to increase student success”. In: *Proceedings of the 2nd International Conference on Learning Analytics and Knowledge*. LAK '12. New York, NY, USA: Association for Computing Machinery, Apr. 2012, pp. 267–270. ISBN: 978-1-4503-1111-3. DOI: 10.1145/2330601.2330666.
- [37] Katerina Mangaroska et al. “Gaze insights into debugging behavior using learner-centred analysis”. In: *Proceedings of the 8th International Conference on Learning Analytics and Knowledge*. LAK '18. New York, NY, USA: Association for Computing Machinery, Mar. 2018, pp. 350–359. ISBN: 978-1-4503-6400-3. DOI: 10.1145/3170358.3170386.
- [38] Maureen A. Guarcello et al. “Balancing Student Success: Assessing Supplemental Instruction Through Coarsened Exact Matching”. en. In: *Technology, Knowledge and Learning* 22.3 (Oct. 2017), pp. 335–352. ISSN: 2211-1670. DOI: 10.1007/s10758-017-9317-0.
- [39] Denise Whitelock et al. “OpenEssayist: a supply and demand learning analytics tool for drafting academic essays”. In: *Proceedings of the Fifth International Conference on Learning Analytics And Knowledge*. LAK '15. New York, NY, USA: Association for Computing Machinery, Mar. 2015, pp. 208–212. ISBN: 978-1-4503-3417-4. DOI: 10.1145/2723576.2723599.
- [40] Dirk T. Tempelaar et al. “Formative assessment and learning analytics”. In: *Proceedings of the Third International Conference on Learning Analytics and Knowledge*. LAK '13. New York, NY, USA: Association for Computing Machinery, Apr. 2013, pp. 205–209. ISBN: 978-1-4503-1785-6. DOI: 10.1145/2460296.2460337.
- [41] Katerina Mangaroska and Michail Giannakos. “Learning Analytics for Learning Design: A Systematic Literature Review of Analytics-Driven Design to Enhance Learning”. In: *IEEE Transactions on Learning Technologies* 12.4 (Oct. 2019). Conference Name: IEEE Transactions on Learning Technologies, pp. 516–534. ISSN: 1939-1382. DOI: 10.1109/TLT.2018.2868673.

- [42] Gregorio Robles and Jesus M. Gonzalez-Barahona. “Mining student repositories to gain learning analytics. An experience report”. In: *2013 IEEE Global Engineering Education Conference (EDUCON)*. ISSN: 2165-9567. Mar. 2013, pp. 1249–1254. DOI: 10.1109/EduCon.2013.6530267.
- [43] Amy Soller et al. “From Mirroring to Guiding: A Review of State of the Art Technology for Supporting Collaborative Learning”. In: *International Journal of Artificial Intelligence in Education (IOS Press)* 15.4 (Dec. 2005), pp. 261–290. ISSN: 15604292.
- [44] K. Gaßner et al. “Analysis Methods for Collaborative Models and Activities”. en. In: *Designing for Change in Networked Learning Environments: Proceedings of the International Conference on Computer Support for Collaborative Learning 2003*. Ed. by Barbara Wasson, Sten Ludvigsen, and Ulrich Hoppe. Computer-Supported Collaborative Learning. Dordrecht: Springer Netherlands, 2003, pp. 369–377. ISBN: 978-94-017-0195-2. DOI: 10.1007/978-94-017-0195-2\_45.
- [45] Daniel Dietsch et al. “Monitoring Student Activity in Collaborative Software Development”. In: *arXiv:1305.0787 [cs]* (June 2013). arXiv: 1305.0787.
- [46] Eun Kyoung Choe et al. “Understanding quantified-selfers’ practices in collecting and exploring personal data”. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI ’14. New York, NY, USA: Association for Computing Machinery, Apr. 2014, pp. 1143–1152. ISBN: 978-1-4503-2473-1. DOI: 10.1145/2556288.2557372.
- [47] Frank Bentley et al. “Health Mashups: Presenting Statistical Patterns between Wellbeing Data and Context in Natural Language to Promote Behavior Change”. In: *ACM Transactions on Computer-Human Interaction* 20.5 (Nov. 2013), 30:1–30:27. ISSN: 1073-0516. DOI: 10.1145/2503823.
- [48] Andre N. Meyer et al. “Design Recommendations for Self-Monitoring in the Workplace: Studies in Software Development”. In: *Proceedings of the ACM on Human-Computer Interaction* 1.CSCW (Dec. 2017), 79:1–79:24. DOI: 10.1145/3134714.
- [49] Anouschka van Leeuwen and Nikol Rummel. “Comparing teachers’ use of mirroring and advising dashboards”. In: *Proceedings of the Tenth International Conference on Learning Analytics & Knowledge*. LAK ’20. New York, NY, USA: Association for Computing Machinery, Mar. 2020, pp. 26–34. ISBN: 978-1-4503-7712-6. DOI: 10.1145/3375462.3375471.
- [50] Riccardo Mazza and Vania Dimitrova. “CourseVis: A graphical student monitoring tool for supporting instructors in web-based distance courses”. en. In: *International Journal of Human-Computer Studies* 65.2 (Feb. 2007), pp. 125–139. ISSN: 1071-5819. DOI: 10.1016/j.ijhcs.2006.08.008.

- [51] Essam Kosba, Vania Dimitrova, and Roger Boyle. “Using Student and Group Models to Support Teachers in Web-Based Distance Education”. en. In: *User Modeling 2005*. Ed. by Liliana Ardissono, Paul Brna, and Antonija Mitrovic. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2005, pp. 124–133. ISBN: 978-3-540-31878-1. DOI: 10.1007/11527886\_17.
- [52] Eleni Voyiatzaki and Nikolaos Avouris. “Support for the teacher in technology-enhanced collaborative classroom”. en. In: *Education and Information Technologies* 19.1 (Mar. 2014), pp. 129–154. ISSN: 1573-7608. DOI: 10.1007/s10639-012-9203-2.
- [53] Briony J. Oates. *Researching Information Systems and Computing*. en. Google-Books-ID: VyYmkaTtRKcC. SAGE, Nov. 2005. ISBN: 978-1-4462-3544-7.
- [54] Suphat Sukamolson. “Fundamentals of quantitative research”. In: (Jan. 2007).
- [55] Karina Kohl Silveira et al. “Confidence in Programming Skills: Gender Insights From StackOverflow Developers Survey”. In: *2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*. ISSN: 2574-1934. May 2019, pp. 234–235. DOI: 10.1109/ICSE-Companion.2019.00091.
- [56] *Audio to Text Automatic Transcription Service & App | temi.com.*
- [57] *oTranscribe.*
- [58] *Qualitative Data Analysis Software | NVivo.*
- [59] *API resources | GitLab.*
- [60] *Chart.js | Open source HTML5 Charts for your website.*

Chapter

**Appendices**





## A.1 NSD application and confirmation

Meldeskjema for behandling av personopplysninger

about:blank



### NSD sin vurdering

#### Prosjekttittel

Masteroppgave om datapunkter fra GitLab

#### Referansenummer

531119

#### Registrert

03.05.2021 av Åsmund Haugse - aasmuha@stud.ntnu.no

#### Behandlingsansvarlig institusjon

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

#### Prosjektansvarlig (vitenskapelig ansatt/veileder eller stipendiat)

Trond Aalberg, Trond.aalberg@ntnu.no, tlf: 73597952

#### Type prosjekt

Studentprosjekt, masterstudium

#### Kontaktinformasjon, student

Åsmund Haugse, aasmuha@stud.ntnu.no, tlf: 46887436

#### Prosjektperiode

03.05.2021 - 30.07.2021

#### Status

05.05.2021 - Vurdert

#### Vurdering (1)

---

##### 05.05.2021 - Vurdert

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 med vedlegg den 05.05.2021, samt i meldingsdialogen mellom innmelder og NSD. Behandlingen kan starte.

## DEL PROSJEKTET MED PROSJEKTANSVARLIG

For studenter er det obligatorisk å dele prosjektet med prosjektansvarlig (veileder). Del ved å trykke på knappen «Del prosjekt» i menylinjen øverst i meldeskjemaet. Prosjektansvarlig bes akseptere invitasjonen innen en uke. Om invitasjonen utløper, må han/hun inviteres på nytt.

## TYPE OPPLYSNINGER OG VARIGHET

Prosjektet vil behandle alminnelige kategorier av personopplysninger frem til 30.07.2021.

## 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 behandles 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: innsyn (art. 15), retting (art. 16), sletting (art. 17), begrensning (art. 18), og dataportabilitet (art. 20).

NSD vurderer at informasjonen om behandlingen 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).

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

## 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: <https://www.nsd.no/personverntjenester/fylle-ut-meldeskjema-for-personopplysninger/melde-endringer-i-meldeskjema>

Du må vente på svar fra NSD før endringen gjennomføres.

#### OPPFØLGING AV PROSJEKTET

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

Lykke til med prosjektet!



## A.2 Interview guide educators

### Interview Guide Instructors

#### Introduction:

I'll start by introducing my research. I've developed a proof-of-concept dashboard solution that mirrors data from GitLab in a more graphical way. This is done to examine if data from GitLab can be useful to students or educators, be it for self-reflection in students or for educators to better understand their groups and receive an indicator on their performances. I've held interviews with students, and before those interviews I've attempted to assign the groups attributes based on what they look like in this dashboard. Then, students themselves assigned their group the same attributes.

Everything said in this interview will be anonymized. Names and references to courses will be replaced with pseudonyms and IDs.

The interview consists of three sections. Firstly, I'll ask you some questions, then we'll move on to the demonstration, and finally I'll ask some more questions.

#### Before demonstration:

1. Do you teach any courses where students use Git and GitLab? If so, can you give a brief summary of how they are used?
  - a. Are they part of the curriculum and are students graded on Git/GitLab use?
2. How do you think data from GitLab can be used to give insight into student group performances?
  - a. What GitLab data do you think can indicate student group performance?
  - b. What kind of visualizations would you use to present GitLab data efficiently?
3. Does your course have student assistants? If so, how do they engage with students?

#### Demonstration:

Introduction to the software solution.

#### Afterwards:

- 1) How did the visualizations of GitLab data match your expectations?
- 2) Do you see any use for a dashboard like this in your courses? If so, for students, instructors or both?
- 3) Could a dashboard like this one impact how Git and GitLab is used and taught in your course?
- 4) Do you have any concerns on using a dashboard like this?
  - a) What do you think of the use of coloured smiley faces?
- 5) Do you think the dashboard would be more useful if names were not anonymized?
- 6) Does your course have student assistants? Would this be of use to them?



## A.3 Interview guide TDT4140

### Interview Guide TDT4140

I'll start by introducing my research. I've developed a proof-of-concept dashboard solution that mirrors data from GitLab in a more graphical way. This is done to examine if data from GitLab can be useful to students or educators, be it for self-reflection in students or for educators to better understand their groups and receive an indicator on their performances. I've held interviews with students, and before those interviews I've attempted to assign the groups attributes based on what they look like in this dashboard. Then, students themselves assigned their group the same attributes.

Everything said in this interview will be anonymized. Names and references to courses will be replaced with pseudonyms and IDs.

#### **Spørsmål:**

Før webbløsningen vises:

1. Can you give a short summary of your project?
  - a. Technology choice, ambition and scope
2. Can you summarize how your group has worked together?
  - a. How was each sprint structured?
  - b. Did you work sitting together or separately?
  - c. How was an average week of work?
3. How has your group distributed their work?
  - a. Coding, writing report, hours spent
  - b. Did everyone contribute an equal amount to the report and programming?
4. What kind of visualizations can help you understand your progress as a group?
  - a. Would you like to view your group's progress compared to other groups?
5. How would you visualize your data?

#### **Etter webbløsningen har blitt vist:**

1. How are the visualizations you are seeing aligned with your expectations of visualizations of your progress/data?
2. Were you surprised to see any of the visualizations? Which ones?
3. Were you made aware of anything new from the visualizations ?
4. What conclusions do you think one can erroneously make from the visualizations?
5. Do you think you can make out which person is which student by viewing the visualizations?





## A.4 Interview guide IT2810

### Interview Guide IT2810

I'll start by introducing my research. I've developed a proof-of-concept dashboard solution that mirrors data from GitLab in a more graphical way. This is done to examine if data from GitLab can be useful to students or educators, be it for self-reflection in students or for educators to better understand their groups and receive an indicator on their performances. I've held interviews with students, and before those interviews I've attempted to assign the groups attributes based on what they look like in this dashboard. Then, students themselves assigned their group the same attributes.

Everything said in this interview will be anonymized. Names and references to courses will be replaced with pseudonyms and IDs.

The interview consists of three sections. Firstly, I'll ask you some questions, then we'll move on to the demonstration, and finally I'll ask some more questions.

#### Questions:

##### Before showing the web solution:

1. Can you give a short summary of your project?
  - a. Who was your group?
2. Can you summarize how your group has worked together?
  - a. How was each exercise structured?
  - b. Did you work sitting together or separately?
3. How has your group distributed their work?
  - a. Coding and hours spent
  - b. Did everyone contribute an equal amount to the programming?
4. What kind of visualizations can help you understand your progress as a group?
  - a. Would you like to view your group's progress compared to other groups?
5. How would you visualize your data?

##### After showing the web solution:

1. How are the visualizations you are seeing aligned with your expectations of visualizations of your progress/data?
2. Were you surprised to see any of the visualizations? Which ones?
3. Were you made aware of anything new from the visualizations ?
4. What conclusions do you think one can erroneously make from the visualizations?
5. Do you think you can make out which person is which student by viewing the visualizations?



## A.5 Results questionnaire IT2810

Spørreundersøkelse om bruk av Git i IT2810 – Rapport - Nettskjema

<https://nettskjema.no/user/form/submission/report.html?id=170303>

### Rapport fra «Spørreundersøkelse om bruk av Git i IT2810»

Innhentede svar pr. 2. juli 2021 14:58

- Leverte svar: **48**
- Påbegynte svar: **0**
- Antall invitasjoner sendt: **0**

#### Med fritekstsvar

### Hvorfor bruker du Git?


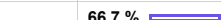

Svar fordelt på antall

	Påstanden er ikke relevant for meg	Veldig uenig	Litt uenig	Verken eller	Litt enig	Veldig enig
Det kreves i forbindelse med skoleprosjekter *	0	2	2	6	12	26
Det er relevant for jobb etter studiet *	0	0	0	1	6	41
Jeg bruker det for egne prosjekter *	1	0	0	4	9	34
Det forenkler arbeid med andre *	0	0	0	0	8	40



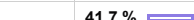

Svar fordelt på prosent

	Påstanden er ikke relevant for meg	Veldig uenig	Litt uenig	Verken eller	Litt enig	Veldig enig
Det kreves i forbindelse med skoleprosjekter *	0 %	4,2 %	4,2 %	12,5 %	25 %	54,2 %
Det er relevant for jobb etter studiet *	0 %	0 %	0 %	2,1 %	12,5 %	85,4 %
Jeg bruker det for egne prosjekter *	2,1 %	0 %	0 %	8,3 %	18,8 %	70,8 %
Det forenkler arbeid med andre *	0 %	0 %	0 %	0 %	16,7 %	83,3 %

### Hvor erfaren vil du si at du er med bruk av Git? \*

Svar	Antall	Prosent
Ikke erfaren	0	<b>0 %</b>
Litt erfaren	9	<b>18,8 %</b> 
Ganske erfaren	32	<b>66,7 %</b> 
Veldig erfaren	7	<b>14,6 %</b> 

### Hvor mye erfaring hadde du med webutvikling før du tok emnet? \*

Svar	Antall	Prosent
Null erfaring	1	<b>2,1 %</b> 
Litt erfaring	19	<b>39,6 %</b> 
En del erfaring	20	<b>41,7 %</b> 
Mye erfaring	8	<b>16,7 %</b> 

### Hvem arbeidet du på gruppe med? \*


Svar	Antall	Prosent
Studenter jeg kjente fra før	33	<b>68,8 %</b> 
Studenter jeg ikke kjente fra før	15	<b>31,2 %</b> 

### Hvordan arbeidet dere sammen? \*

Svar	Antall	Prosent
Satt oftest hver for oss og jobbet	28	<b>58,3 %</b> 

Svar	Antall	Prosent
Satt oftest sammen og jobbet	20	41,7 % 

**Huk av de påstandene om å arbeide hver for seg som du kjenner deg igjen i \***

Svar	Antall	Prosent
Jeg kunne enkelt kontakte gruppen min hvis jeg slet	23	47,9 % 
Det var frustrerende å løse bugs alene	10	20,8 % 
Det ble hver enkelt sitt ansvar å gjøre jobben sin	19	39,6 % 
Det fungerte bra å jobbe alene	19	39,6 % 
Det var demotiverende å jobbe alene	4	8,3 % 
Gruppearbeidet ble mindre sosialt	12	25 % 
Det gjorde arbeidet mitt mer effektivt	17	35,4 % 
Det gjorde arbeidet mitt mindre effektivt	4	8,3 % 
Det gjorde arbeidet mindre gøy å holde på med	4	8,3 % 

**Huk av de påstandene om å arbeide sammen som du kjenner deg igjen i \***

Svar	Antall	Prosent
Det gjorde gruppearbeidet mer sosialt	19	39,6 % 
Webutvikling var vanskelig å gjøre alene	7	14,6 % 
Det gjorde det enklere å hjelpe hverandre med utviklingen	16	33,3 % 
Jeg lærer bedre av å forklare og få forklart hvordan koden fungerer	10	20,8 % 
Det gjorde det enklere å få alle til å jobbe like mye	9	18,8 % 
Det ga en bedre oversikt over prosjektet	16	33,3 % 
Det var bedre å være flere til å løse bugs som oppstod	13	27,1 % 

**Hvor enig er du i følgende påstander om måten gruppen din kommuniserte?****Svar fordelt på antall**

	Ikke relevant for meg	Veldig uenig	Litt uenig	Verken eller	Litt enig	Veldig enig
Gruppen min kommuniserte godt i løpet av hvert prosjekt *	0	3	1	3	12	9
Gruppen min kommuniserte mest i starten av hvert prosjekt *	0	6	6	5	7	4
Gruppen min fordelte oppgaver, deretter arbeidet vi hver for oss *	0	1	1	1	19	6
Jeg hadde god oversikt over prosjektets fremgang til enhver tid *	0	1	4	4	9	10
Hele gruppen deltok i valg av teknologi og hva vi skulle implementere *	0	1	1	2	9	15

**Svar fordelt på prosent**

	Ikke relevant for meg	Veldig uenig	Litt uenig	Verken eller	Litt enig	Veldig enig
Gruppen min kommuniserte godt i løpet av hvert prosjekt *	0 %	10,7 %	3,6 %	10,7 %	42,9 %	32,1 %
Gruppen min kommuniserte mest i starten av hvert prosjekt *	0 %	21,4 %	21,4 %	17,9 %	25 %	14,3 %
Gruppen min fordelte oppgaver, deretter arbeidet vi hver for oss *	0 %	3,6 %	3,6 %	3,6 %	67,9 %	21,4 %
Jeg hadde god oversikt over prosjektets fremgang til enhver tid *	0 %	3,6 %	14,3 %	14,3 %	32,1 %	35,7 %
Hele gruppen deltok i valg av teknologi og hva vi skulle implementere *	0 %	3,6 %	3,6 %	7,1 %	32,1 %	53,6 %

**Hvor enig er du i følgende påstander?**

**Svar fordelt på antall**

	Ikke relevant for meg	Veldig uenig	Litt uenig	Verken eller	Litt enig	Veldig enig
Jeg måtte lære meg Git på egenhånd før jeg kunne starte med prosjektet/utviklingen *	10	21	6	3	2	6
Git er lett å lære *	1	1	10	12	18	6
Git er lett å bruke *	1	1	4	7	25	10
Git er lett å forstå *	1	2	9	11	20	5
Jeg forstår hvordan Git fungerer *	0	1	2	3	16	26
Git er nødvendig for prosjektet jeg arbeider på *	0	0	0	3	10	35
Git hjelper meg i arbeidet mitt *	0	0	0	0	4	44
Git har gjort arbeid med andre enklere *	0	0	0	0	5	43
Git har vært demotiverende å bruke *	0	35	3	7	2	1
Git har vært tidvis frustrerende å bruke *	0	9	9	6	19	5
Git har bydd på mer trøbbel enn det har gitt belønning *	0	39	7	1	1	0
Git gjør utviklingsprosessen enklere *	0	1	0	1	10	36
Git gjør utviklingsprosessen mer effektiv *	0	0	0	3	8	37
Git har tillatt meg å fokusere på oppgaven jeg skal løse *	0	1	2	11	12	22
Git forhindret meg i å fokusere på webutvikling *	0	32	9	7	0	0
Jeg vet hva som er god Git praksis/Jeg har en mening om hva som er god Git praksis *	0	0	3	5	20	20
Jeg har blitt flinkere til å bruke Git i løpet av Webutvikling *	2	3	5	10	15	13
Webutvikling har gjort fordelene med Git mer tydelige *	1	2	5	23	11	6
Det har vært gøy å bruke Git i utviklingsprosessen *	2	0	1	15	14	16
Jeg skulle ønske Git ble bedre forklart tidligere i studieløpet *	2	2	4	11	5	24

**Svar fordelt på prosent**

	Ikke relevant for meg	Veldig uenig	Litt uenig	Verken eller	Litt enig	Veldig enig
Jeg måtte lære meg Git på egenhånd før jeg kunne starte med prosjektet/utviklingen *	20,8 %	43,8 %	12,5 %	6,2 %	4,2 %	12,5 %
Git er lett å lære *	2,1 %	2,1 %	20,8 %	25 %	37,5 %	12,5 %
Git er lett å bruke *	2,1 %	2,1 %	8,3 %	14,6 %	52,1 %	20,8 %
Git er lett å forstå *	2,1 %	4,2 %	18,8 %	22,9 %	41,7 %	10,4 %
Jeg forstår hvordan Git fungerer *	0 %	2,1 %	4,2 %	6,2 %	33,3 %	54,2 %
Git er nødvendig for prosjektet jeg arbeider på *	0 %	0 %	0 %	6,2 %	20,8 %	72,9 %
Git hjelper meg i arbeidet mitt *	0 %	0 %	0 %	0 %	8,3 %	91,7 %
Git har gjort arbeid med andre enklere *	0 %	0 %	0 %	0 %	10,4 %	89,6 %
Git har vært demotiverende å bruke *	0 %	72,9 %	6,2 %	14,6 %	4,2 %	2,1 %
Git har vært tidvis frustrerende å bruke *	0 %	18,8 %	18,8 %	12,5 %	39,6 %	10,4 %
Git har bydd på mer trøbbel enn det har gitt belønning *	0 %	81,2 %	14,6 %	2,1 %	2,1 %	0 %
Git gjør utviklingsprosessen enklere *	0 %	2,1 %	0 %	2,1 %	20,8 %	75 %
Git gjør utviklingsprosessen mer effektiv *	0 %	0 %	0 %	6,2 %	16,7 %	77,1 %
Git har tillatt meg å fokusere på oppgaven jeg skal løse *	0 %	2,1 %	4,2 %	22,9 %	25 %	45,8 %
Git forhindret meg i å fokusere på webutvikling *	0 %	66,7 %	18,8 %	14,6 %	0 %	0 %
Jeg vet hva som er god Git praksis/Jeg har en mening om hva som er god Git praksis *	0 %	0 %	6,2 %	10,4 %	41,7 %	41,7 %
Jeg har blitt flinkere til å bruke Git i løpet av Webutvikling *	4,2 %	6,2 %	10,4 %	20,8 %	31,2 %	27,1 %
Webutvikling har gjort fordelene med Git mer tydelige *	2,1 %	4,2 %	10,4 %	47,9 %	22,9 %	12,5 %
Det har vært gøy å bruke Git i utviklingsprosessen *	4,2 %	0 %	2,1 %	31,2 %	29,2 %	33,3 %
Jeg skulle ønske Git ble bedre forklart tidligere i studieløpet *	4,2 %	4,2 %	8,3 %	22,9 %	10,4 %	50 %

**Hvordan brukte du/gruppen din Git/GitLab?****Svar fordelt på antall**

	Ikke brukt	Brukte noen ganger	Brukte ofte	Brukte hele tiden
Tilbakemeldinger på merge requests *	19	10	10	9
Issues for å dele opp oppgaver *	0	3	12	33
Task board for oversikt av issues *	12	8	6	22
Løse merge conflicts *	2	15	11	20
Issues ble tagget i commit-meldinger *	5	6	16	21
Arbeid på separate branches *	0	1	7	40

**Svar fordelt på prosent**

	Ikke brukt	Brukte noen ganger	Brukte ofte	Brukte hele tiden
Tilbakemeldinger på merge requests *	39,6 %	20,8 %	20,8 %	18,8 %
Issues for å dele opp oppgaver *	0 %	6,2 %	25 %	68,8 %
Task board for oversikt av issues *	25 %	16,7 %	12,5 %	45,8 %
Løse merge conflicts *	4,2 %	31,2 %	22,9 %	41,7 %
Issues ble tagget i commit-meldinger *	10,4 %	12,5 %	33,3 %	43,8 %
Arbeid på separate branches *	0 %	2,1 %	14,6 %	83,3 %

**Hvor enig er du i følgende påstander?****Svar fordelt på antall**

	Ikke relevant for meg	Veldig uenig	Litt uenig	Verken eller	Litt enig	Veldig enig
Jeg vil bare dele kode jeg er trygg på at er god *	1	3	11	9	16	8
Jeg synes det er greit at gruppelemmer ser koden jeg har skrevet *	1	0	0	1	7	39
Jeg synes det er greit at faglærer får se hvordan jeg jobber gjennom Git-historikken min *	1	0	2	4	8	33
Jeg synes det er greit at fremmede ser koden jeg har skrevet *	1	0	2	7	15	23
Jeg er komfortabel med å gi tilbakemeldinger på andres kode *	1	0	9	3	15	20
Jeg synes det var vanskelig å vurdere kvaliteten på andres kode *	1	3	11	7	20	6
Jeg er komfortabel med å gi tilbakemeldinger på kode skrevet av noen jeg ikke kjenner *	1	0	6	4	21	16
Jeg vil bare gi tilbakemeldinger på kode skrevet av noen jeg kjenner *	1	26	8	10	3	0
Jeg er komfortabel med å få kritikk mot egen kode *	1	0	2	1	20	24
Jeg tar meg nær av tilbakemeldinger som går på kvaliteten av koden min *	1	13	15	13	4	2
Jeg ble bedre i webutvikling av å vurdere andres kode *	2	4	5	6	22	9
Jeg får tilbakemeldinger som hjelper meg med å forbedre koden min *	2	2	11	5	21	7
Jeg opplevde tilbakemeldingene jeg fikk som nyttige *	2	1	6	8	24	7
Jeg var enig i tilbakemeldingene jeg fikk på innleveringene mine *	1	1	6	9	28	3
Det er mer komfortabelt å gi tilbakemeldinger som anonym *	1	0	0	6	12	29

**Svar fordelt på prosent**

	Ikke relevant for meg	Veldig uenig	Litt uenig	Verken eller	Litt enig	Veldig enig
Jeg vil bare dele kode jeg er trygg på at er god *	2,1 %	6,2 %	22,9 %	18,8 %	33,3 %	16,7 %
Jeg synes det er greit at gruppelemmer ser koden jeg har skrevet *	2,1 %	0 %	0 %	2,1 %	14,6 %	81,2 %

Jeg synes det er greit at faglærere får se hvordan jeg jobber gjennom Git-historikken min *	2,1 %	0 %	4,2 %	8,3 %	16,7 %	68,8 %
Jeg synes det er greit at fremmede ser koden jeg har skrevet *	2,1 %	0 %	4,2 %	14,6 %	31,2 %	47,9 %
Jeg er komfortabel med å gi tilbakemeldinger på andres kode *	2,1 %	0 %	18,8 %	6,2 %	31,2 %	41,7 %
Jeg synes det var vanskelig å vurdere kvaliteten på andres kode *	2,1 %	6,2 %	22,9 %	14,6 %	41,7 %	12,5 %
Jeg er komfortabel med å gi tilbakemeldinger på kode skrevet av noen jeg ikke kjenner *	2,1 %	0 %	12,5 %	8,3 %	43,8 %	33,3 %
Jeg vil bare gi tilbakemeldinger på kode skrevet av noen jeg kjenner *	2,1 %	54,2 %	16,7 %	20,8 %	6,2 %	0 %
Jeg er komfortabel med å få kritikk mot egen kode *	2,1 %	0 %	4,2 %	2,1 %	41,7 %	50 %
Jeg tar meg nær av tilbakemeldinger som går på kvaliteten av koden min *	2,1 %	27,1 %	31,2 %	27,1 %	8,3 %	4,2 %
Jeg ble bedre i webutvikling av å vurdere andres kode *	4,2 %	8,3 %	10,4 %	12,5 %	45,8 %	18,8 %
Jeg får tilbakemeldinger som hjelper meg med å forbedre koden min *	4,2 %	4,2 %	22,9 %	10,4 %	43,8 %	14,6 %
Jeg opplevde tilbakemeldingene jeg fikk som nyttige *	4,2 %	2,1 %	12,5 %	16,7 %	50 %	14,6 %
Jeg var enig i tilbakemeldingene jeg fikk på innleveringene mine *	2,1 %	2,1 %	12,5 %	18,8 %	58,3 %	6,2 %
Det er mer komfortabelt å gi tilbakemeldinger som anonym *	2,1 %	0 %	0 %	12,5 %	25 %	60,4 %

### Hvor enig er du i følgende påstander?

#### Svar fordelt på antall

	Ikke relevant for meg	Veldig uenig	Litt uenig	Verken eller	Litt enig	Veldig enig
GitLab gjør det enklere å følge opp fremgangen på kodebasen/prosjektet *	1	0	0	3	14	30
GitLab har gjort det enklere å dele opp utviklingsoppgaver i mindre oppgaver *	1	0	2	0	15	30
GitLab gjør det enklere å fordele arbeidsoppgaver *	1	0	0	6	14	27
Gruppen min brukte funksjonaliteten til GitLab til arbeidsfordeling *	1	0	0	3	20	24
GitLab gjør gruppearbeid med fremmede enklere *	10	0	1	2	6	29
GitLab gjør samarbeidet mer konkurrerende(Antall commits, antall issues fullført, linjer kode osv.) *	1	9	9	10	15	4
Git fungerer best hvis alle gruppelemmer ligger på samme ferdighetsnivå *	2	1	9	12	16	8
GitLab gjør det enklere å arbeide sammen med noen som er på et lavere ferdighetsnivå enn deg selv *	3	1	6	18	13	7
GitLab gjør det vanskeligere å arbeide sammen med noen som er på et lavere ferdighetsnivå enn deg selv *	3	8	12	19	4	2
Utvikling med GitLab gjør gruppelemmer som får til mindre til syndebukker *	2	5	17	8	14	2
Git gjør forskjeller i ferdighetsnivå mer synlig *	1	2	2	7	27	9
Arbeid med GitLab gjør det verre å være på et lavere ferdighetsnivå enn sine gruppelemmer *	5	7	7	16	9	4
Git gjør det enklere å jobbe distribuert(Ikke på fysisk samme plass) *	1	1	0	1	6	39
GitLab gjør prosjektarbeidet mindre sosialt(Gjennom å jobbe adskilt og mindre behov for å møtes) *	3	10	15	12	7	1
Git fører til at de flinkeste overkjører de dårligste i utviklingsprosessen(Skrive om mye kode, gjør mye av arbeidet selv osv.) *	2	8	7	9	18	4
Alle på gruppen min bidro omtrent like mye til prosjektene våre *	1	7	10	6	17	7
Gruppen min hjalp hverandre med å lære *	1	1	3	4	18	21
Alle på gruppen min var på omtrent samme nivå *	1	6	11	9	15	6

#### Svar fordelt på prosent

	Ikke relevant for meg	Veldig uenig	Litt uenig	Verken eller	Litt enig	Veldig enig

GitLab gjør det enklere å følge opp fremgangen på kodebasen/prosjektet *	2,1 %	0 %	0 %	6,2 %	29,2 %	62,5 %
GitLab har gjort det enklere å dele opp utviklingsoppgaver i mindre oppgaver *	2,1 %	0 %	4,2 %	0 %	31,2 %	62,5 %
GitLab gjør det enklere å fordele arbeidsoppgaver *	2,1 %	0 %	0 %	12,5 %	29,2 %	56,2 %
Gruppen min brukte funksjonaliteten til GitLab til arbeidsfordeling *	2,1 %	0 %	0 %	6,2 %	41,7 %	50 %
GitLab gjør gruppearbeid med fremmede enklere *	20,8 %	0 %	2,1 %	4,2 %	12,5 %	60,4 %
GitLab gjør samarbeidet mer konkurrerende(Antall commits, antall issues fullført, linjer kode osv.) *	2,1 %	18,8 %	18,8 %	20,8 %	31,2 %	8,3 %
Git fungerer best hvis alle gruppede medlemmer ligger på samme ferdighetsnivå *	4,2 %	2,1 %	18,8 %	25 %	33,3 %	16,7 %
GitLab gjør det enklere å arbeide sammen med noen som er på et lavere ferdighetsnivå enn deg selv *	6,2 %	2,1 %	12,5 %	37,5 %	27,1 %	14,6 %
GitLab gjør det vanskeligere å arbeide sammen med noen som er på et lavere ferdighetsnivå enn deg selv *	6,2 %	16,7 %	25 %	39,6 %	8,3 %	4,2 %
Utvikling med GitLab gjør gruppede medlemmer som får til mindre til syndebukker *	4,2 %	10,4 %	35,4 %	16,7 %	29,2 %	4,2 %
Git gjør forskjeller i ferdighetsnivå mer synlig *	2,1 %	4,2 %	4,2 %	14,6 %	56,2 %	18,8 %
Arbeid med GitLab gjør det verre å være på et lavere ferdighetsnivå enn sine gruppede medlemmer *	10,4 %	14,6 %	14,6 %	33,3 %	18,8 %	8,3 %
Git gjør det enklere å jobbe distribuert(Ikke på fysisk samme plass) *	2,1 %	2,1 %	0 %	2,1 %	12,5 %	81,2 %
GitLab gjør prosjektarbeidet mindre sosialt(Gjennom å jobbe adskilt og mindre behov for å møtes) *	6,2 %	20,8 %	31,2 %	25 %	14,6 %	2,1 %
Git fører til at de flinkeste overkjører de dårligste i utviklingsprosessen(Skrive om mye kode, gjør mye av arbeidet selv osv.) *	4,2 %	16,7 %	14,6 %	18,8 %	37,5 %	8,3 %
Alle på gruppen min bidro omtrent like mye til prosjektene våre *	2,1 %	14,6 %	20,8 %	12,5 %	35,4 %	14,6 %
Gruppen min hjalp hverandre med å lære *	2,1 %	2,1 %	6,2 %	8,3 %	37,5 %	43,8 %
Alle på gruppen min var på omtrent samme nivå *	2,1 %	12,5 %	22,9 %	18,8 %	31,2 %	12,5 %

For å bli med i trekning av gavekort, bruker du lenken du får tilsendt i epost-kvitteringen du mottar når du har sendt inn svaret ditt.

[Se nylige endringer i Nettskjema](#)





## A.6 Results questionnaire TDT4140

Spørreundersøkelse om bruk av Git i TDT4140 – Rapport - Nettskjema

<https://nettskjema.no/user/form/submission/report.html?id=203071>

### Rapport fra «Spørreundersøkelse om bruk av Git i TDT4140»

Innhentede svar pr. 2. juli 2021 14:58

- Leverte svar: **43**
- Påbegynte svar: **0**
- Antall invitasjoner sendt: **0**

#### Med fritekstsva

### Hvorfor bruker du Git?

#### Svar fordelt på antall

	Påstanden er ikke relevant for meg	Veldig uenig	Litt uenig	Verken eller	Litt enig	Veldig enig
Det kreves i forbindelse med skoleprosjekter *	0	2	0	1	13	27
Det er relevant for jobb etter studiet *	0	1	2	3	18	19
Jeg bruker det for egne prosjekter *	4	7	6	7	11	8
Det forenkler arbeid med andre *	0	0	0	0	13	30

#### Svar fordelt på prosent

	Påstanden er ikke relevant for meg	Veldig uenig	Litt uenig	Verken eller	Litt enig	Veldig enig
Det kreves i forbindelse med skoleprosjekter *	0 %	4,7 %	0 %	2,3 %	30,2 %	62,8 %
Det er relevant for jobb etter studiet *	0 %	2,3 %	4,7 %	7 %	41,9 %	44,2 %
Jeg bruker det for egne prosjekter *	9,3 %	16,3 %	14 %	16,3 %	25,6 %	18,6 %
Det forenkler arbeid med andre *	0 %	0 %	0 %	0 %	30,2 %	69,8 %

### Hvor erfaren vil du si at du er med bruk av Git? \*

Svar	Antall	Prosent
Ikke erfaren	6	14 %
Litt erfaren	25	58,1 %
Ganske erfaren	9	20,9 %
Veldig erfaren	3	7 %

### Hvor mye erfaring hadde du med webutvikling før du tok emnet? \*

Svar	Antall	Prosent
Null erfaring	17	39,5 %
Litt erfaring	19	44,2 %
En del erfaring	6	14 %
Mye erfaring	1	2,3 %

### Hvordan arbeidet dere sammen? \*

Svar	Antall	Prosent
Satt oftest hver for oss og jobbet	11	25,6 %
Satt oftest sammen og jobbet	8	18,6 %
Satt både mye sammen og mye adskilt	24	55,8 %

### Huk av de påstandene om å arbeide hver for seg som du kjenner deg igjen i \*

Svar	Antall	Prosent
------	--------	---------

Svar	Antall	Prosent
Jeg kunne enkelt kontakte gruppen min hvis jeg slet	9	20,9 %
Det var frustrerende å løse bugs alene	4	9,3 %
Det ble hver enkelt sitt ansvar å gjøre jobben sin	6	14 %
Det fungerte bra å jobbe alene	5	11,6 %
Det var demotiverende å jobbe alene	1	2,3 %
Gruppearbeidet ble mindre sosialt når vi satt adskilt	10	23,3 %
Git/GitLab gjorde arbeidet mitt mer effektivt	9	20,9 %
Git/GitLab gjorde arbeidet mitt mindre effektivt	0	0 %
Git/GitLab gjorde arbeidet mindre gøy å holde på med	1	2,3 %

**Huk av de påstandene om å arbeide sammen som du kjenner deg igjen i \***

Svar	Antall	Prosent
Git/GitLab gjorde gruppearbeidet mer sosialt	3	7 %
Programmering var vanskelig å gjøre alene	4	9,3 %
Git/GitLab gjorde det enklere å hjelpe hverandre med utviklingen	9	20,9 %
Jeg lærer bedre av å forklare og få forklart hvordan koden fungerer	5	11,6 %
Git/GitLab gjorde det enklere å få alle til å jobbe like mye	4	9,3 %
Git/GitLab ga en bedre oversikt over prosjektet	11	25,6 %
Det var bedre å være flere til å løse bugs som oppstod	5	11,6 %

**Hvor enig er du i følgende påstander om måten gruppen din kommuniserte?****Svar fordelt på antall**

	Ikke relevant for meg	Veldig uenig	Litt uenig	Verken eller	Litt enig	Veldig enig
Gruppen min kommuniserte godt i løpet av hver sprint *	0	1	0	1	7	2
Gruppen min kommuniserte mest i starten av hver sprint *	0	1	2	1	5	2
Gruppen min fordelte oppgaver, deretter arbeidet vi hver for oss *	0	0	1	2	6	2
Jeg hadde god oversikt over prosjektets fremgang til enhver tid *	0	1	2	2	6	0
Hele gruppen deltok i valg av teknologi *	0	1	2	2	3	3

**Svar fordelt på prosent**

	Ikke relevant for meg	Veldig uenig	Litt uenig	Verken eller	Litt enig	Veldig enig
Gruppen min kommuniserte godt i løpet av hver sprint *	0 %	9,1 %	0 %	9,1 %	63,6 %	18,2 %
Gruppen min kommuniserte mest i starten av hver sprint *	0 %	9,1 %	18,2 %	9,1 %	45,5 %	18,2 %
Gruppen min fordelte oppgaver, deretter arbeidet vi hver for oss *	0 %	0 %	9,1 %	18,2 %	54,5 %	18,2 %
Jeg hadde god oversikt over prosjektets fremgang til enhver tid *	0 %	9,1 %	18,2 %	18,2 %	54,5 %	0 %
Hele gruppen deltok i valg av teknologi *	0 %	9,1 %	18,2 %	18,2 %	27,3 %	27,3 %

**Hvor enig er du i følgende påstander?****Svar fordelt på antall**

	Ikke relevant for meg	Veldig uenig	Litt uenig	Verken eller	Litt enig	Veldig enig
Jeg måtte lære meg Git på egenhånd før jeg kunne starte med prosjektet/utviklingen *	4	8	8	10	8	5

Git er lett å lære *	0	2	13	14	14	0
Git er lett å bruke *	0	2	6	12	21	2
Git er lett å forstå *	0	5	13	9	15	1
Jeg forstår hvordan Git fungerer *	0	2	6	7	22	6
Git er nødvendig for prosjektet jeg arbeidet på *	0	0	2	2	9	30
Git hjelper meg i arbeidet mitt *	0	0	1	1	17	24
Git har gjort arbeid med andre enklere *	0	0	1	2	12	28
Git har vært demotiverende å bruke *	0	21	5	9	7	1
Git har vært tidvis frustrerende å bruke *	0	4	9	5	16	9
Git har bydd på mer trøbbel enn det har gitt belønning *	0	24	13	3	2	1
Git gjør utviklingsprosessen enklere *	0	0	0	4	17	22
Git gjør utviklingsprosessen mer effektiv *	0	0	0	5	17	21
Git har tillatt meg å fokusere på oppgaven jeg skal løse *	0	0	3	15	19	6
Git forhindre meg i å fokusere på programmeringen *	0	15	18	6	4	0
Jeg vet hva som er god Git praksis/Jeg har en mening om hva som er god Git praksis *	0	1	13	6	21	2
Jeg har blitt flinkere til å bruke Git i løpet av Programvareutvikling *	0	1	1	5	17	19
Programmering har gjort fordelene med Git mer tydelige *	1	0	1	12	21	8
Det har vært gøy å bruke Git i utviklingsprosessen *	0	2	5	16	15	5
Jeg skulle ønske Git ble bedre forklart tidligere i studieløpet *	0	0	0	4	9	30

## Svar fordelt på prosent

	Ikke relevant for meg	Veldig uenig	Litt uenig	Verken eller	Litt enig	Veldig enig
Jeg måtte lære meg Git på egenhånd før jeg kunne starte med prosjektet/utviklingen *	9,3 %	18,6 %	18,6 %	23,3 %	18,6 %	11,6 %
Git er lett å lære *	0 %	4,7 %	30,2 %	32,6 %	32,6 %	0 %
Git er lett å bruke *	0 %	4,7 %	14 %	27,9 %	48,8 %	4,7 %
Git er lett å forstå *	0 %	11,6 %	30,2 %	20,9 %	34,9 %	2,3 %
Jeg forstår hvordan Git fungerer *	0 %	4,7 %	14 %	16,3 %	51,2 %	14 %
Git er nødvendig for prosjektet jeg arbeidet på *	0 %	0 %	4,7 %	4,7 %	20,9 %	69,8 %
Git hjelper meg i arbeidet mitt *	0 %	0 %	2,3 %	2,3 %	39,5 %	55,8 %
Git har gjort arbeid med andre enklere *	0 %	0 %	2,3 %	4,7 %	27,9 %	65,1 %
Git har vært demotiverende å bruke *	0 %	48,8 %	11,6 %	20,9 %	16,3 %	2,3 %
Git har vært tidvis frustrerende å bruke *	0 %	9,3 %	20,9 %	11,6 %	37,2 %	20,9 %
Git har bydd på mer trøbbel enn det har gitt belønning *	0 %	55,8 %	30,2 %	7 %	4,7 %	2,3 %
Git gjør utviklingsprosessen enklere *	0 %	0 %	0 %	9,3 %	39,5 %	51,2 %
Git gjør utviklingsprosessen mer effektiv *	0 %	0 %	0 %	11,6 %	39,5 %	48,8 %
Git har tillatt meg å fokusere på oppgaven jeg skal løse *	0 %	0 %	7 %	34,9 %	44,2 %	14 %
Git forhindre meg i å fokusere på programmeringen *	0 %	34,9 %	41,9 %	14 %	9,3 %	0 %
Jeg vet hva som er god Git praksis/Jeg har en mening om hva som er god Git praksis *	0 %	2,3 %	30,2 %	14 %	48,8 %	4,7 %
Jeg har blitt flinkere til å bruke Git i løpet av Programvareutvikling *	0 %	2,3 %	2,3 %	11,6 %	39,5 %	44,2 %
Programmering har gjort fordelene med Git mer tydelige *	2,3 %	0 %	2,3 %	27,9 %	48,8 %	18,6 %
Det har vært gøy å bruke Git i utviklingsprosessen *	0 %	4,7 %	11,6 %	37,2 %	34,9 %	11,6 %
Jeg skulle ønske Git ble bedre forklart tidligere i studieløpet *	0 %	0 %	0 %	9,3 %	20,9 %	69,8 %

## Hvordan brukte du/gruppen din Git/GitLab?

## Svar fordelt på antall

	Ikke brukt	Brukte noen ganger	Brukte ofte	Brukte hele tiden

Tilbakemeldinger på merge requests *	10	20	7	6
Issues for å dele opp oppgaver *	2	6	6	29
Task board for oversikt av issues *	5	4	9	25
Løse merge conflicts *	2	12	14	15
Issues ble tagget i commit-meldinger *	7	11	9	16
Arbeid på separate branches *	0	0	5	38

**Svar fordelt på prosent**

	Ikke brukt	Brukte noen ganger	Brukte ofte	Brukte hele tiden
Tilbakemeldinger på merge requests *	23,3 %	46,5 %	16,3 %	14 %
Issues for å dele opp oppgaver *	4,7 %	14 %	14 %	67,4 %
Task board for oversikt av issues *	11,6 %	9,3 %	20,9 %	58,1 %
Løse merge conflicts *	4,7 %	27,9 %	32,6 %	34,9 %
Issues ble tagget i commit-meldinger *	16,3 %	25,6 %	20,9 %	37,2 %
Arbeid på separate branches *	0 %	0 %	11,6 %	88,4 %

**Hvor enig er du i følgende påstander?****Svar fordelt på antall**

	Ikke relevant for meg	Veldig uenig	Litt uenig	Verken eller	Litt enig	Veldig enig
Jeg vil bare dele kode jeg er trygg på at er god *	1	3	7	6	21	5
Jeg synes det er greit at gruppelemmer ser koden jeg har skrevet *	0	0	0	2	11	30
Jeg synes det er greit at faglærer får se hvordan jeg jobber gjennom Git-historikken min *	0	0	4	4	16	19
Jeg synes det er greit at fremmede ser koden jeg har skrevet *	0	1	5	8	18	11
Jeg er komfortabel med å gi tilbakemeldinger på andres kode *	0	2	9	5	17	10
Jeg synes det var vanskelig å vurdere kvaliteten på andres kode *	0	1	4	6	25	7
Jeg er komfortabel med å gi tilbakemeldinger på kode skrevet av noen jeg ikke kjenner *	0	3	7	8	18	7
Jeg vil bare gi tilbakemeldinger på kode skrevet av noen jeg kjenner *	0	9	17	8	7	2
Jeg er komfortabel med å få kritikk mot egen kode *	0	0	2	4	19	18
Jeg tar meg nær av tilbakemeldinger som går på kvaliteten av koden min *	0	18	14	8	3	0
Jeg ble bedre i programmering av å vurdere andres kode *	6	0	2	9	20	6
Jeg får tilbakemeldinger som hjelper meg med å forbedre koden min *	4	0	6	11	16	6
Jeg opplevde tilbakemeldingene jeg fikk som nyttige *	7	0	1	9	18	8
Jeg var enig i tilbakemeldingene jeg fikk på innleveringene mine *	6	0	6	11	12	8
Det er mer komfortabelt å gi tilbakemeldinger som anonym *	2	0	8	13	12	8

**Svar fordelt på prosent**

	Ikke relevant for meg	Veldig uenig	Litt uenig	Verken eller	Litt enig	Veldig enig
Jeg vil bare dele kode jeg er trygg på at er god *	2,3 %	7 %	16,3 %	14 %	48,8 %	11,6 %
Jeg synes det er greit at gruppelemmer ser koden jeg har skrevet *	0 %	0 %	0 %	4,7 %	25,6 %	69,8 %
Jeg synes det er greit at faglærer får se hvordan jeg jobber gjennom Git-historikken min *	0 %	0 %	9,3 %	9,3 %	37,2 %	44,2 %
Jeg synes det er greit at fremmede ser koden jeg har skrevet *	0 %	2,3 %	11,6 %	18,6 %	41,9 %	25,6 %
Jeg er komfortabel med å gi tilbakemeldinger på andres kode *	0 %	4,7 %	20,9 %	11,6 %	39,5 %	23,3 %
Jeg synes det var vanskelig å vurdere kvaliteten på andres kode *	0 %	2,3 %	9,3 %	14 %	58,1 %	16,3 %

Jeg er komfortabel med å gi tilbakemeldinger på kode skrevet av noen jeg ikke kjenner *	0 %	7 %	16,3 %	18,6 %	41,9 %	16,3 %
Jeg vil bare gi tilbakemeldinger på kode skrevet av noen jeg kjenner *	0 %	20,9 %	39,5 %	18,6 %	16,3 %	4,7 %
Jeg er komfortabel med å få kritikk mot egen kode *	0 %	0 %	4,7 %	9,3 %	44,2 %	41,9 %
Jeg tar meg nær av tilbakemeldinger som går på kvaliteten av koden min *	0 %	41,9 %	32,6 %	18,6 %	7 %	0 %
Jeg ble bedre i programmering av å vurdere andres kode *	14 %	0 %	4,7 %	20,9 %	46,5 %	14 %
Jeg får tilbakemeldinger som hjelper meg med å forbedre koden min *	9,3 %	0 %	14 %	25,6 %	37,2 %	14 %
Jeg opplevde tilbakemeldingene jeg fikk som nyttige *	16,3 %	0 %	2,3 %	20,9 %	41,9 %	18,6 %
Jeg var enig i tilbakemeldingene jeg fikk på innleveringene mine *	14 %	0 %	14 %	25,6 %	27,9 %	18,6 %
Det er mer komfortabelt å gi tilbakemeldinger som anonym *	4,7 %	0 %	18,6 %	30,2 %	27,9 %	18,6 %

### Hvor enig er du i følgende påstander?

#### Svar fordelt på antall

	Ikke relevant for meg	Veldig uenig	Litt uenig	Verken eller	Litt enig	Veldig enig
GitLab gjør det enklere å følge opp fremgangen på kodebasen/prosjektet *	0	0	1	1	20	21
GitLab har gjort det enklere å dele opp utviklingsoppgaver i mindre oppgaver *	0	1	3	7	15	17
GitLab gjør det enklere å fordele arbeidsoppgaver *	0	1	2	3	18	19
Gruppen min brukte funksjonaliteten til GitLab til arbeidsfordeling *	0	2	8	1	15	17
GitLab gjør gruppearbeid med fremmede enklere *	1	0	1	5	18	18
GitLab gjør samarbeidet mer konkurrerende(Antall commits, antall issues fullført, linjer kode osv.) *	1	9	10	11	9	3
Git fungerer best hvis alle gruppede medlemmer ligger på samme ferdighetsnivå *	1	1	8	12	13	8
GitLab gjør det enklere å arbeide sammen med noen som er på et lavere ferdighetsnivå enn deg selv *	1	0	12	19	11	0
GitLab gjør det vanskeligere å arbeide sammen med noen som er på et lavere ferdighetsnivå enn deg selv *	1	0	12	17	12	1
Utvikling med GitLab gjør gruppede medlemmer som får til mindre til synde bukker *	0	8	14	9	12	0
Git gjør forskjeller i ferdighetsnivå mer synlig *	0	1	2	5	27	8
Arbeid med GitLab gjør det verre å være på et lavere ferdighetsnivå enn sine gruppede medlemmer *	2	3	6	14	17	1
Git gjør det enklere å jobbe distribuert(Ikke på fysisk samme plass) *	0	0	0	3	14	26
GitLab gjør prosjektarbeidet mindre sosialt(Gjennom å jobbe adskilt og mindre behov for å møtes) *	1	6	8	13	15	0
Git fører til at de flinkeste overkjører de dårligste i utviklingsprosessen(Skrive om mye kode, gjør mye av arbeidet selv osv.) *	0	2	13	12	12	4
Alle på gruppen min bidro omtrent like mye til prosjektene våre *	0	11	16	6	9	1
Gruppen min hjalp hverandre med å lære *	0	0	1	1	24	17
Alle på gruppen min var på omtrent samme nivå *	0	21	12	3	6	1

#### Svar fordelt på prosent

	Ikke relevant for meg	Veldig uenig	Litt uenig	Verken eller	Litt enig	Veldig enig
GitLab gjør det enklere å følge opp fremgangen på kodebasen/prosjektet *	0 %	0 %	2,3 %	2,3 %	46,5 %	48,8 %
GitLab har gjort det enklere å dele opp utviklingsoppgaver i mindre oppgaver *	0 %	2,3 %	7 %	16,3 %	34,9 %	39,5 %
GitLab gjør det enklere å fordele arbeidsoppgaver *	0 %	2,3 %	4,7 %	7 %	41,9 %	44,2 %
Gruppen min brukte funksjonaliteten til GitLab til arbeidsfordeling *	0 %	4,7 %	18,6 %	2,3 %	34,9 %	39,5 %

GitLab gjør gruppearbeid med fremmede enklere *	2,3 %	0 %	2,3 %	11,6 %	41,9 %	41,9 %
GitLab gjør samarbeidet mer konkurrerende(Antall commits, antall issues fullført, linjer kode osv.) *	2,3 %	20,9 %	23,3 %	25,6 %	20,9 %	7 %
Git fungerer best hvis alle gruppede medlemmer ligger på samme ferdighetsnivå *	2,3 %	2,3 %	18,6 %	27,9 %	30,2 %	18,6 %
GitLab gjør det enklere å arbeide sammen med noen som er på et lavere ferdighetsnivå enn deg selv *	2,3 %	0 %	27,9 %	44,2 %	25,6 %	0 %
GitLab gjør det vanskeligere å arbeide sammen med noen som er på et lavere ferdighetsnivå enn deg selv *	2,3 %	0 %	27,9 %	39,5 %	27,9 %	2,3 %
Utvikling med GitLab gjør gruppede medlemmer som får til mindre til sydebukker *	0 %	18,6 %	32,6 %	20,9 %	27,9 %	0 %
Git gjør forskjeller i ferdighetsnivå mer synlig *	0 %	2,3 %	4,7 %	11,6 %	62,8 %	18,6 %
Arbeid med GitLab gjør det verre å være på et lavere ferdighetsnivå enn sine gruppede medlemmer *	4,7 %	7 %	14 %	32,6 %	39,5 %	2,3 %
Git gjør det enklere å jobbe distribuert(Ikke på fysisk samme plass) *	0 %	0 %	0 %	7 %	32,6 %	60,5 %
GitLab gjør prosjektarbeidet mindre sosialt(Gjennom å jobbe adskilt og mindre behov for å møtes) *	2,3 %	14 %	18,6 %	30,2 %	34,9 %	0 %
Git fører til at de flinkeste overkjører de dårligste i utviklingsprosessen(Skrive om mye kode, gjør mye av arbeidet selv osv.) *	0 %	4,7 %	30,2 %	27,9 %	27,9 %	9,3 %
Alle på gruppen min bidro omtrent like mye til prosjektene våre *	0 %	25,6 %	37,2 %	14 %	20,9 %	2,3 %
Gruppen min hjalp hverandre med å lære *	0 %	0 %	2,3 %	2,3 %	55,8 %	39,5 %
Alle på gruppen min var på omtrent samme nivå *	0 %	48,8 %	27,9 %	7 %	14 %	2,3 %

For å bli med i trekning av gavekort, bruker du lenken du får tilsendt i epost-kvitteringen du mottar når du har sendt inn svaret ditt.

[Se nylige endringer i Nettskjema](#)

