Master's thesis

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

Fredrik Førde Lindhagen, and
Sigurd Marius Melsom

Visualizing Repository Data to Facilitate Feedback in Software Engineering courses

Master's thesis in Master in Informatics
Supervisor: George Adrian Stoica
June 2022

NTNU
Norwegian University of
Science and Technology

Fredrik Førde Lindhagen, and
Sigurd Marius Melsom

Visualizing Repository Data to Facilitate Feedback in Software Engineering courses

Master's thesis in Master in Informatics
Supervisor: George Adrian Stoica
June 2022

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

NTNU
Norwegian University of
Science and Technology

Abstract

Learning to plan and build software in collaboration is essential for Computer Science (CS) students to learn. In the industry Git has become the favored Version Control System (VCS), and an important part of CS education.

We explore how current research analyses data from Git, repository hosting services, and project management systems to improve the quality of Software Engineering courses and is used to evaluate and counsel students. Then a prototype is built using third-party visualization software to give student groups visualized feedback on projects in Software Engineering courses. Finally, usability tests are conducted in two rounds to discover and resolve usability problems and receive qualitative data on how students and Teaching Assistants (TAs) value the visualization tool.

This thesis has two main contributions. First, 1) Increased qualitative knowledge of how lower-level Computer Science students value insight into a course project through an interactive visualization tool. Important considerations when developing such tools are also covered. Lastly, 2) increased knowledge of the benefits and limitations of using third-party software to create visualizations for student groups.

We find that students and TAs react positively to a visualization tool that continuously gives them feedback on their projects, amplifying their feedback loop. Third-party visualization software is found to have limitations affecting students' usability. However, the effects are more promising for Teaching Assistants (TAs).

This knowledge, and related research, can be built upon in further research to create a more generalizable and applicable platform for students in Software Engineering courses.

Sammendrag

Planlegging og utvikling av programvare er fundamentale egenskaper å lære for datastudenter. I industrien har Git blitt det foretrukne verktøyet for versjonskontroll og blitt en viktig del av utdanningsløpet til datastudenter.

I denne oppgaven utforsker vi hvordan data fra Git, kodebrønner, og prosjekstyringssystemer anvendes i forskning for å bedre kvaliteten på programvareutviklingsfag, samt bruk i evaluering og oppfølging av studenter. Videre utvikles det en prototype i et tredjeparts visualiseringsverktøy som gir studenter visualiserte tilbakemeldinger på hvordan de jobber på prosjekter i programvaretuviklingsfag. Til slutt utføres brukertester fordelt på 2 runder for å utbedre brukskvalitet og få kvalitative data på hvordan studenter og studentassistenter verdsetter plattformen.

Denne avhandingen har i hovedsak to bidrag: 1) Økt kvalitativ kunnskap om hvordan studenter innen programvareutvikling verdsetter innsikt i prosjekter gjennom et interaktivt visualiseringsverktøy, og viktige poenger til utviklingen av slike systemer. 2) Økt kunnskap om fordelene og begrensningene ved å bruke tredjeparts visualiseringsverktøy for å utvikle visualiseringer for studentgrupper i fag.

Vi viser at både studenter og studentassistenter er positive til et visualiseringsverktøy som gir dem raskt tilbakemelding på prosjektene sine og forsterker tilbakemeldingssyklusen. I tillegg viser vi at tredjeparts visualiseringsverktøy har begrensninger som påvirker brukskvaliteten til studenter, men er mer lovende for studentassistenter.

Denne kunnskapen kan bygges videre på ved å utvikle en mer generaliserbar og anvendbar plattform for studenter i programvareutviklingsfag.

Preface

This thesis marks our final submission of our two year Master of Science in Informatics at the Norwegian University of Science and Technology (NTNU), and in total five years of education at NTNU.

Firstly, we would like to extend our gratitude to our supervisor George Adrian Stoica. His patience and guidance throughout the semester has been greatly appreciated.

Furthermore, we want to extend our gratefulness to all students and Teaching Assistants (TAs) who partook in interviews, providing invaluable insight and suggestion for improvements. Additionally, we wish to thank the Information Systems and Software Engineering (ISSE) group at Department of Computer Science for financial support of gift cards to the participants.

Lastly, we thank our friends and family for their support, and the staff at Café-sito Stripa (NTNU), who kept us caffeinated throughout the thesis.

Fredrik Lindhagen & Sigurd Melsom
Trondheim, 2022

Contents

Figures

Tables

Acronyms

CD Continuous Delivery. 7–11, 68, 74

CI Continuous Integration. 1, 7–11, 18, 68, 74

CS Computer Science. iii, 9, 10, 15, 19

CVCS Central Version Control System. 5, 6

DORA DevOps Research and Assessment. 11

DVCS Distributed Version Control System. 5, 6

GUI Graphical User Interface. 3, 4, 27, 28, 69

JS JavaScript. 71

JSON JavaScript Object Notation. 12, 70

LDAP Lightweight Directory Access Protocol. 39

LoC Lines of Code. 18, 20, 49–51, 54, 55, 65, 66

NSD Norwegian Centre for Research Data. 22, 24, 28, 72, 73

NTNU Norwegian University of Science and Technology. vii, 1, 2, 5, 7–10, 16, 19, 22, 26, 28, 73

RBAC Role-based access control. 71

SAML security assertion markup language. 38

SSO Single Sign-On. 32, 38, 39, 69, 71

TA Teaching Assistant. iii, vii, 2–4, 13, 16, 23, 25–27, 29, 40, 43, 47, 51, 53, 58, 59, 61, 64–66, 68, 72, 73, 75, 76

VCS Version Control System. iii, xiii, 1, 5–7, 15, 75

XML eXtensible Markup Language. 12

Glossary

DevOps "The combination of cultural philosophies, practices, and tools that increases an organization's ability to deliver applications and services at high velocity"[1]. 23, 66, 68

Git A Distributed Version Control System, used to handle code changes and simplify collaboration. iii, v, xv, 1–8, 10, 11, 13, 15–19, 24, 27, 28, 31, 35, 37, 41, 42, 44, 48, 51, 54, 59–61, 63–70, 72, 75, 76

GitLab Centralized Git repository for multiple projects, with additional project management features. 1–11, 13, 15, 18, 19, 23–25, 27, 28, 31–36, 39, 41, 42, 44, 49–51, 53, 54, 59–61, 63, 64, 69, 70, 72, 75, 76

IT1901 Course at Norwegian University of Science and Technology (NTNU), where students get knowledge and skills in agile application development in teams. [2]. 1, 2, 4, 8–10, 16, 21–24, 26–29, 31–33, 38, 41, 42, 47, 48, 55, 56, 61–63, 72–74

IT2810 Course at Norwegian University of Science and Technology (NTNU), where students are taught technologies and methods used in the development of web-based solutions. This course is taught in the fall semester.[3]. 19, 47

Linter ESLint defines linting as: "Code linting is a type of static analysis that is frequently used to find problematic patterns or code that doesn't adhere to certain style guidelines"[4]. 17, 55, 68

Software Engineering Software Engineering deals with the design, development, testing and maintenance of software applications. iii, 1–3, 5, 10–13, 15–19, 27, 63, 67–69, 71–73, 75, 76

TDT4140 Course at Norwegian University of Science and Technology (NTNU), where students are taught project management, agile and group processes,

[1] https://aws.amazon.com/devops/what-is-devops/
[2] https://www.ntnu.edu/studies/courses/IT1901
[3] https://www.ntnu.edu/studies/courses/IT2810
[4] https://eslint.org/docs/about/

and demonstrate the ability to plan and manage a small Software Engineering project. This course is taught in the spring semester.[5]. 4, 8–10, 16, 19, 47, 72, 73

Trello A web-based work management tool with kanban task boards.[6]. 53

[5] https://www.ntnu.edu/studies/courses/TDT4140

[6] https://trello.com/

Chapter 1

Introduction

1.1 Motivation

Collaboration with peers is crucial for computer scientists in academia and the industry [1–3]. The Version Control System (VCS) Git[1] is at the centre of collaborative development, enabling developers to work on the same code base asynchronously. To properly utilize Git, the Git repository should be hosted on repository hosting services like GitHub and GitLab. These hosting services have collaborative features such as task management, Continuous Integration (CI) pipelines, code reviews, and wiki services. Mastering these tools is a crucial skill for becoming a proficient developer [1, 4] and has become part of the curriculum for many computer science degrees. These Software Engineering skills are best learned through group projects [5]. In order to learn good practices and become skilled at collaborative software development [6], it is favorable for students to get continuous feedback [7] and insight into their projects.

In the Software Engineering course IT1901 at Norwegian University of Science and Technology (NTNU), the main objective is to build the basis for producing high-quality software in an agile development paradigm. The focus is on learning core techniques for managing source code, issues, build and deployment, testing, and the basic elements of agile development. For most students, this course is their first experience programming together in a team. The course is based on a group project with deliveries, which forms the basis for their grade. The functionality of the developed product is not as crucial as its code quality and testing. Their practices in collaborative tools like Git and GitLab are evaluated and part of their grade in the course.

We believe that a tool that highlights the mistakes and shortcomings of students' software development projects can give the students a better understanding of how to collaborate. Furthermore, the students can quickly correct bad practices by getting immediate feedback on their development process throughout the project, which helps improve the students' performance [7]. The outlined tool can

[1] `https://git-scm.com/`

also help course staff get an improved overview of how the groups collaborate, facilitating effective individual feedback to the different groups. We believe such a tool can be implemented by mining data from GitLab repositories and using established analytics and third-party visualization tools.

1.2 Research objective

The main objective of this thesis is to expand the knowledge on making students more skilled at collaborative programming. More specifically, we look at how visualization software can be used to provide continuous feedback on data from Git and repository hosting services.

We hypothesize that relevant insights will improve students' learning- and software development process and give course staff the ability to help students struggling by providing more targeted feedback. A prototype of a visualization tool is developed using third-party visualization software. It is made with the requirements of the a specific Software Engineering course at Norwegian University of Science and Technology (NTNU), IT1901. The prototype is tested for its usability and value through interviews, generating qualitative data. Literature on learning analytics in Software Engineering education is reviewed to determine the state of the art of collaborative insights for student development groups.

It is important to note that the goal of this insight is primarily to provide students with valuable insights during the course, so they can improve their process and not make quantitative evaluation metrics for course staff.

1.2.1 Stakeholders

In this project we focus on two primary stakeholders, *Students* and *Teaching Assistant (TA)* (Table 1.1). *Students* are most important since they are expected to get the most value from this system. TAs are included because they supervise one or multiple student groups, requiring access to the same insights.

Stakeholder	**Description**
Teaching Assistant (TA)	Students and PhDs hired to assist the Course staff and Student Groups.
Student	Students attending a Software Engineering course collaborating in student groups.

Table 1.1: Description of the primary project stakeholders.

1.2.2 Research Questions

The motivation and context can be broken down into the *Research Questions*, which specifies the objectives of our thesis.

RQ1 "What is the perceived value of a tool visualizing Git and GitLab usage, for students in Software Engineering courses?"

RQ2 "What are the benefits and limitations of using a third-party visualization software to visualize insight into students' Software Engineering projects?"

RQ1 is answered by building a prototype of a visualization tool, which is tested on students and Teaching Assistants (TAs) in a relevant Software Engineering course. The usability tests are conducted in two rounds, allowing us to address usability problems between each round (See Chapter 4 for details). Post-interviews collect qualitative data by asking open-ended questions to participants and having them reflect on the benefits and limitations of a visualization tool.

RQ2 is answered by looking at a selection of popular third-party visualization software (Section 5.4.1), comparing their benefits and limitations. Then one of those systems is selected for further prototyping (Section 5.5) and tested as part of the usability testing. Usability problems that participants explicitly noted, as well as other observations, were recorded. Finally, any problems that require considerable effort to resolve in the visualization software are discussed (Section 7.2.1) together with its benefits.

1.3 Findings

It is found that both students and TAs react positively to a visualization tool that continuously gives them feedback on their projects, amplifying their feedback loop. Additionally, some students appreciated how this data could enable them to discuss better how they cooperate.

Further, Grafana[2], a third-party visualization software, was chosen for developing a prototype. Grafana has several promises in terms of fast prototyping, flexibility in visualization options, and the ability to drill down in time. Ideal for TAs and course staff to get insight into one or multiple projects. However, for the primary use case, *RQ1*, the standardized Graphical User Interface (GUI) has several limitations in providing students with customized and interactive feedback. Most of the feedback received in the last round of tests was rooted in graphical limitations in Grafana. Accordingly, the thesis shows that third-party visualization software has limitations for stakeholders requiring a high degree of flexibility. Therefore, a custom-built visualization tool should be considered for student groups.

[2] `https://grafana.com/`

The findings from this thesis show future possibilities to explore in more detail what type of visualizations provide students with the most help, comparing different approaches. Additionally, a case study should explore how students use such tools during their courses. Finally, even though students and TAs from IT1901 are used as the basis for the developed prototype, the feedback should be generalizable to other similar courses. For example, most participants did, in parallel, attend a similar course, TDT4140 Software Engineering[3], and noted a similar value in this course as well.

1.4 Thesis Structure

This thesis starts with necessary background information on specific topics in Chapter 2. Chapter 3 summarizes related research on the use of analytical data from Git and GitLab in educational contexts. Next, Chapter 4 describes the overall design and creation process to build the prototype and details how the usability testing, with pre-interviews and post-interviews, is planned, conducted, and analyzed. Subsequent, Chapter 5 goes into detail about the creation of the prototype. That chapter discusses overall technical decisions, explains why Grafana was chosen as the third-party visualization system, potential pitfalls from crawling GitLab, and how the Graphical User Interface (GUI) evolved between each revision. Chapter 6 presents the findings from usability testing and interviews. Chapter 7 answers *RQ1* and *RQ2* by discussing the benefits and limitations of our visualization tool based on our findings, related work, and experience building the tool. Lastly, Chapter 8 summarizes the thesis and its contribution, including a section on future work.

[3] `https://www.ntnu.edu/studies/courses/TDT4140` - Web page describing the course "Software Engineering"

Chapter 2

Background

This chapter introduces select theoretical concepts and background knowledge needed to understand the contributions of this thesis. This includes what a Version Control System (VCS) is, best practices in Git, relevant functionality from GitLab, the most relevant Software Engineering courses at Norwegian University of Science and Technology (NTNU), and existing commercial and open-source analysis systems.

2.1 Version Control

A Version Control System (VCS) is a system for managing and keeping track of changes to files over time [8]. The most obvious trait of a VCS is the possibility of reverting files to previous versions and rolling back an entire project to a previous state. However, these systems have several other features, such as tracking the author of specific edits, monitoring the evolution of files, and working on different tasks by using branches. As a result, VCSs are considered a must by developers in the industry to successfully manage code changes in Software Engineering projects [9].

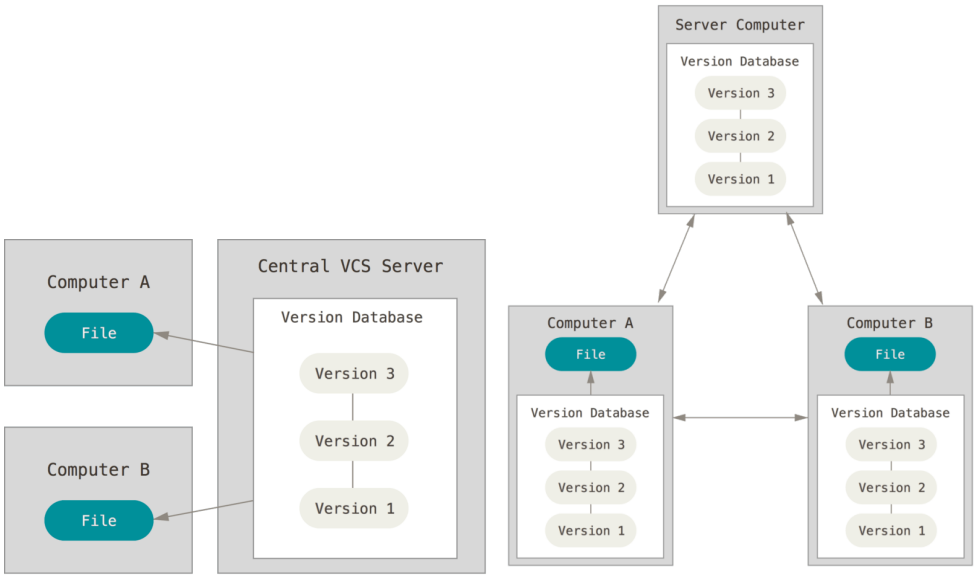
Typically, VCSs are classified as one of two architectures: *Central*, or *Distributed*. Their core differences are illustrated in Figure 2.1.

Central Version Control System (CVCS), is an architecture designed for cooperation, with a single centralized server for tracking file changes. However, this single server architecture is vulnerable as it becomes a single point of failure. If this server were corrupted or unavailable, the project files would be unavailable and, in the worst-case unrecoverable. This architecture is used by systems such as Subversion[1] and Concurrent Versions System[2] [8].

The architecture of Distributed Version Control System (DVCS) is similar to CVCS in the sense that it is suited for cooperation. However, there is no centralized server. Instead, the version database is mirrored on all the developers' com-

[1] `https://subversion.apache.org/`

[2] `https://cvs.nongnu.org/`



(a) Central Version Control System (CVCS) **(b)** Distributed Version Control System (DVCS)

Figure 2.1: Primary types of VCS architectures. Figures are from [8]. Retrieved February, 2022

puters, avoiding the single point of failure architecture of CVCS. One computer can go down without affecting the other developers by mirroring the entire version history to all the developers. The project can be recovered from one of the other mirrored instances.

2.1.1 Git

Git is an open-source DVCS which has become the most popular VCS tool for developers. The *Stack Overflow 2021 developer survey* reported that over 90% of 76-thousand respondents use Git [10]. Git has become the standard when collaborating with other developers on software projects and an essential skill for developers to master. The distributed architecture of Git allows all developers of a project to work on the same codebase and files simultaneously. A hosted Git repository, such as GitHub[3], GitLab[4], or BitBucket[5], are typically used as the centralized codebase, representing the code running in production.

In Git, file evolutions are tracked with snapshots called *commits*. A commit is a snapshot of all the files in a version of the project at the moment in time. Every time a *commit* is created, snapshots of all files changed since the previous commit are taken. In addition to the file snapshots, commits contain metadata such

[3] `https://github.com/`
[4] `https://about.gitlab.com/`
[5] `https://bitbucket.org/`

as timestamp, author name, email, commit message, a reference to the previous commit, and an identifying hash used to identify a specific commit. The commit message is written by the commit author when initiating the commit, which is useful for explaining what changes are done in the commit.

Git Commit Best practices

To successfully utilize Git most projects adhere to a set of conventions good enough for their project that makes tracking changes over time consistent and predictable. Best practices are mainly discussed in Grey Literature [11] such as web articles. Even though most sources agree on the primary conventions, there are different scales of strictness, such as how Git commits should be structured. For example, *Conventional Commits* follow a strict and semantic approach [12], with the benefit of having machine-parseable commits autogenerating changelogs. This, however, might be too strict for most developers who do not require the added features.

Table 2.1 is a summarized collection of general Git best practices, synthesized from collections of different Grey Literature. These are sourced through Google Scholar[6] and common search engines, by searching for "*Git Best practice*". Note that this is not an exhaustive list of references but includes enough to have sufficient confidence in what practices are recommended by multiple, instead of just a preference from a particular author or company.

Most notable commit practices are "*Write good commit messages*", "*Make small, atomic/single-purpose commits*", "*Branch frequently, using short-lived branches*", and "*Do not commit half-done work*", which are referenced by more than two sources. Additional practices are likely also relevant, but this depends more on how each project is structured. The practices "*Use branches*" can, for many projects, be a good practice. However, Trunk-based development [13] recommends that small teams may directly commit to trunk (main/master branch), with the claim that this increases Continuous Integration (CI) & Continuous Delivery (CD) performance.

Good commit messages can be distilled to a collection of guidelines, Table 2.2. These are typically more specific and agreed upon by many sources. Tools such as commitlint[7] have been created to validate the structure of a commit during development, preventing developers from introducing violating messages to a project. However, not all of the rules are easily validated by a tool. Validating the use of "*Imperative mood*" requires more complicated systems to understand the language.

2.2 GitLab

Norwegian University of Science and Technology (NTNU) use a self-hosted version of GitLab as the primary VCS. Similar to GitHub, Bitbucket and other hosting

[6] `https://scholar.google.com/`
[7] `https://commitlint.js.org/#/`

Best practice	Referenced in
Write good commit messages (details in Table 2.2)	[14–19]
Make small, atomic/single-purpose commits	[14, 15, 17–19]
Branch frequently, using short-lived branches	[13, 15, 19, 20]
Do not commit half-done work	[14–16, 18]
Commit often	[14, 15]
Test your code before you commit	[14, 15]
Use branches	[14, 19]
Obtain feedback through code reviews	[15, 19]
Agree on a workflow and branching strategy	[14, 19]
Use trunk (main/master branch) as the only shared branch for a team	[13]
Be concise and straight to the point. Explain *why* the changes were made, not *what* has changed	[18]

Table 2.1: Git best practices
Collection is synthesized from multiple Grey Literature articles and books, to give a confidence indicator on each best practice's relevance.

services, GitLab provide a centralized place where developers can store and manage their codebase. Additionally, these services provide project management, code review, and other Continuous Integration (CI) & Continuous Delivery (CD) functionalities to manage the whole life-cycle of a product. In several of the courses at NTNU that use GitLab, such as IT1901 (Section 2.3.1) and TDT4140 (Section 2.3.2), the ability to work efficiently with GitLab's features is an important learning goal. Most notable features are GitLab Issues, Milestones, and Merge Requests.

GitLab issue tracking provide developers with a central location to record, prioritize and manage project tasks. Developers can directly discuss an issue and reference them in Git commits or Merge Requests, which can automatically update or close the issue.

Merge Request is functionality in GitLab that allow other developers to peer review changes done to the code before it is merged into a branch, typically the `main` branch. This is similar to the Pull Request functionality in GitHub and Bitbucket. Peer-reviewed code has been empirically shown to provide higher quality software [21] while being a valuable tool to share knowledge with other peers [22, 23]. The Merge Request views typically present the developers with an overview

Commit message rules	**Referenced in**
Commit messages should be no longer than 50 characters	[8, 14–18]
Begin a commit message using capital letter	[16–18]
Do not end the message with a period	[16, 18]
Use the imperative mood	[8, 14, 16–19]
Commit message should be consistent	[18]
Explain *why* the changes were made, not *what*	[18]
Always leave the second line blank	[8, 14–16]
Wrap the body at 72 characters	[8, 16]
Use the body to explain *what* and *why* instead of *how*	[16]

Table 2.2: Specific rules for a good commit message

of the files and lines changed between two branches, including name and description on the Merge Request, relevant GitLab Issues, test coverage, and CI & CD pipeline status.

Different guidelines to how peer review of code should be conducted exist, such as Google's guidelines for code review[8]. A notable point from their guideline is *to have fast code reviews, that should take at most one business day to receive a response*.

2.3 Software Engineering at NTNU

Norwegian University of Science and Technology (NTNU) have multiple courses teaching Software Engineering. The most popular courses are IT1901[9] and TDT4140[10], teaching students software development and agile methodologies.

2.3.1 IT1901 - Informatics, Project I

The software engineering course IT1901 is a practical programming course at NTNU aimed at teaching "*knowledge and skills in agile application development in teams*". It is mainly taken by second-year Computer Science (CS) students. The course is graded on a group project with three deliveries, where the students work in groups of 5-6 members. Along with the projects, weekly lectures on tools and

[8] `https://google.github.io/eng-practices/review/`
[9] `https://www.ntnu.no/studier/emner/IT1901`
[10] `https://www.ntnu.edu/studies/courses/TDT4140`

methodology needed for the projects are held. The course objectives include learning to use methodologies and tools for collaborative programming, code structure and quality, and REST API architecture. The focus is to put a solid basis of skills and techniques for core software development practices.

In the projects, students are required to version the project with Git using NTNU's own GitLab server. For many students, this course is their first time in a Software Engineering and using Git. The lectures give a theoretical introduction to Git and its best practices, while the course project is a hands-on introduction to the tool.

Several parts of GitLab are investigated when the course staff evaluates groups' development processes. The course project is divided into 3 phases where the emphasis of the evaluation is on the last phase. The students are expected to have learned the practices through the first two phases. When evaluating groups, course staff explore multiple GitLab practices, such as:

- **Commit messages** are expected to be informative and of reasonable length (Table 2.1). Commit messages should also reference other GitLab features such as Issues.
- **Issue tracking and Milestones:** Students should structure their work into GitLab Issues which should have informative *descriptions*, *tags*, and *titles*. Each of the three project phases should have a designated GitLab Milestone, and the issues should belong to their respective Milestone. Each issue should be assigned to a group member and marked as closed when completed.
- **Merge Requests** should be used for merges to `main`/`master`. The requests should reference GitLab Issues, be reviewed by another team member, and merged by someone other than the author.
- **Presence of special files:** Files such as `.gitignore` and `README.md` are expected to be present in the repository. Depending on the project and technology used, other files, such as `pom.xml` for Maven[11] projects, be considered necessary, and absence may lead to point reduction.
- **CI & CD:** Use of GitLab CI & CD with a reasonable implementation of pipelines is desirable, but not a requirement.
- **Other details:** Some unexpected things have previously been found in the repositories, which the students should have avoided. An example is some groups committing build-folders and dependency-folders, causing repository storage to exceed $500MB$, failing to correct the mistake.

2.3.2 TDT4140 - Software Engineering

TDT4140 is a course taught in the spring semester to second-year Computer Science (CS) students at NTNU, where IT1901 is a recommended prerequisite course. Compared to IT1901, TDT4140 focus more purely on the Agile methodology Scrum, and the activities related to it [24]. The students learn throughout the

[11] `https://maven.apache.org/`

course how to plan and manage small Software Engineering projects by working in groups to develop a project and completing individual assignments. Even though a product is developed, the quality and robustness of the actual product are not in focus. Most importantly is how the group utilizes the Scrum activities during development.

2.4 Existing analysis systems

Several commercial systems to analyze Git and project management usage exist, either through the repository hosting themselves, such as GitLab Analytics or as external systems reading from the repositories. This section look at commercial and open-source, ready-to-use options.

2.4.1 GitLab Analytics

GitLab Analytics is a built-in feature on GitLab, providing insight into the behavior and usage of projects and groups. These insights cover *the amount of time spent waiting for Code review*, *time spent planning an issue*, *overall success ratio of pipelines*, *ratio of succeeded and all pipelines over time*, *pipeline duration per commit*, *code coverage statistics*, and *number of commits per day in month, weekday and hour*. The benefit of GitLab's analytics insight are that they are rooted in DevOps Research and Assessment (DORA) [20, 25]. However, to get more insightful and granular data, such as Contribution Analysis[12], a Premium or Ultimate subscription tier is required. These subscription tiers may be too costly for certain organizations even though education/academic institutions get free licenses to the higher tiers[13].

Furthermore, GitLab Analytics does not provide visualization or feedback of common Git best practices (Table 2.1) or custom recommendations on Issues and Merge Requests because these are highly dependent on project and organization policies.

2.4.2 GitHub Classroom

GitHub Classroom[14] is a collection of features designed to simplify the management and grading of assignments. Through GitHub Classroom, course staff can run code for automated assignment grading and create Pull Requests to answer questions and provide feedback. However, except for the ability to create standardized Continuous Integration (CI) & Continuous Delivery (CD) pipelines, it does not provide analytical functionality to receive insights into how students work. Each repository has access to basic insights into its project, such as the Number of Commits, Number of Lines Added or Deleted. However, in GitHub Classroom,

[12] `https://docs.gitlab.com/ee/user/group/contribution_analytics/`
[13] `https://about.gitlab.com/solutions/education/`
[14] `https://classroom.github.com/`

this requires a GitHub Team subscription, which may be too costly for particular courses.

2.4.3 Pluralsight Flow

Pluralsight Flow[15] claim to optimize software delivery with actionable insights, helping teams identify where their workflow bottlenecks are and what changes have the biggest impact on delivery. Flow supports different code repositories and can show how many lines of code are spent on new work, legacy refactoring, churn, or helping others. Regarding collaboration, managers can see who creates Merge Requests and who typically conducts Code Review on them. Although this is a commercial product, it claims many benefits but does not reflect on its limitations. Neither does it show any evidence supporting its claim to help improve productivity and collaboration versus being a tool to micromanage development. Although some insights in this product look promising for use in Software Engineering courses, the Flow software is designed for software teams working full time. Therefore, several key metrics may show inaccurate trends when student groups only have one or two days of work in a week.

2.4.4 Gitinspector

Gitinspector is an open-source static analysis tool showing general analysis per author or workload over time. This includes the Number of Commits, Number of Lines Added and Deleted and can show cumulative work done by each author. It was initially created to fetch repository statistics for university courses at Chalmers University of Technology and Gothenburg University and now claims it is used as a grading tool by many universities.

Although Gitinspector can output raw HTML with the analyzed results, it also supports machine-readable output formats, such as eXtensible Markup Language (XML) and JavaScript Object Notation (JSON). The machine-readable formats make it extensible for future analysis or presentation.

Gitinspector's main drawback is the lack of maintenance, with the last release in 2016. Currently, it supports only Python 2.7, which has been deprecated since 2020[16].

[15] `https://www.pluralsight.com/product/flow`

[16] `https://www.python.org/doc/sunset-python-2/`

Chapter 3

Related Work

This chapter reviews the state-of-the-art literature on how Learning Analytics is applied in Software Engineering courses. Furthermore, relevant metrics from Git and GitLab of student contribution and work are synthesized into Table 3.2. A subset of the synthesized metrics are selected to be included in the prototype (Chapter 5) and tested on students and Teaching Assistants (TAs).

3.1 Literature Review Process

Relevant research is discovered through literature review, using Snowballing [26] as the primary strategy. Inclusion and exclusion criteria for the search and filtration process are listed in Table 3.1.

Inclusion	**Exclusion**
Discuss Git or GitLab metrics and its use in Student Project	Is not written in English or Norwegian
Mention Learning analytics in the context of Software Engineering courses	Is older than 2005
	Is Bachelor thesis' or panel discussion

Table 3.1: General inclusion and exclusion criteria for our literature search
Some exceptions can be made for the exclusions, if we consider the source to be highly relevant

The snowballing process involved building a starting set of literature that formed the basis for further expansion and iterations (Figure 3.1). These are selected from the available literature that is considered relevant for this thesis and

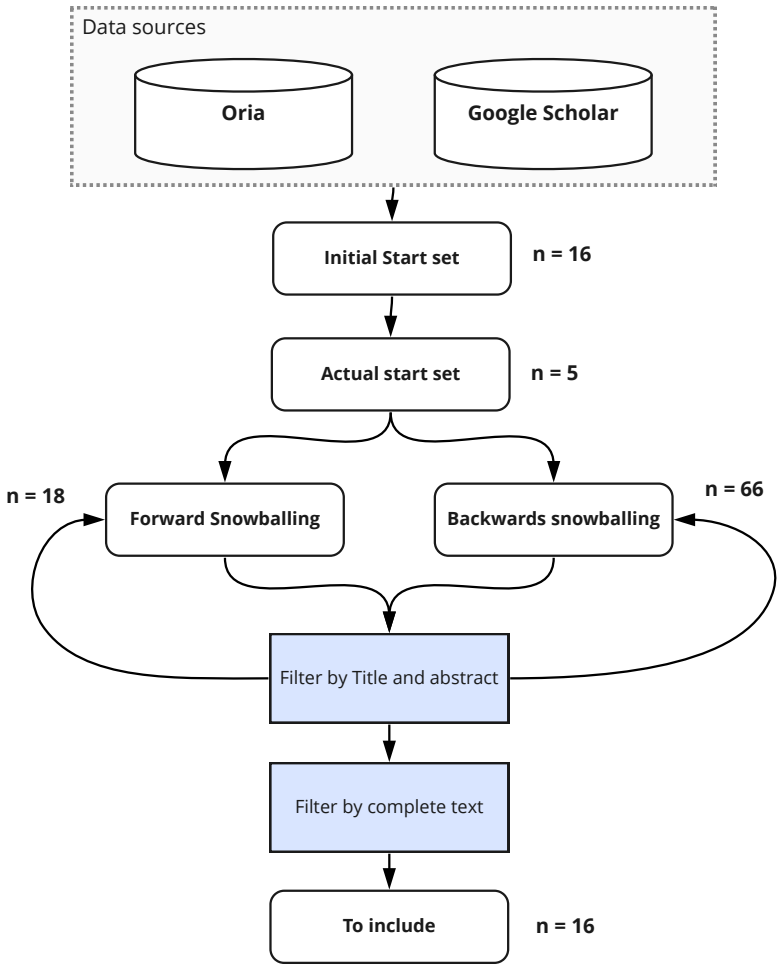


Figure 3.1: Literature review process
The literature review process using snowballing [26] as strategy. The blue boxes illustrate steps in the filtration process.

literature found on Oria[1] and Google Scholar[2]. In addition to a master thesis, 5 papers are chosen for the initial start set.

In the first iteration, Forward and Backward snowballing are applied to the start set, where Forward finds sources that reference the literature in the starter set. In Backward snowballing, the reference list in the starter set is crawled and filtered. All new literature that has been chosen from the first round is placed in a new starter set. Next, this process is repeated multiple times until we have sufficient literature or no new literature is found. In total, 66 documents from Backward snowballing and 18 from Forward snowballing matched the criteria in their Title or abstract. Each was read in more detail to consider their actual

[1] `https://ntnu.oria.no`
[2] `https://scholar.google.com/`

relevance. Finally, 16 papers were kept after the final filtering.

3.2 Use of Learning Analytics in Software Engineering courses

Using digital tools in Software Engineering courses generates a substantial amount of data that can be used for Learning Analytics. Such data has great potential for teachers to get an insight into students' group work and has become an important research direction [27]. Learning Analytics in software engineering courses has been researched from several approaches and with different intentions. For example, some research is aimed at applying Learning Analytics to courses to increase the understanding of how Software Engineering course can be improved [27, 28]. Other papers focus on how Learning Analytics can be applied in project assessment [29–31].

3.2.1 Analytics to better understand students' development processes

Learning Analytics is frequently used in Software Engineering to understand better how students work and make course improvements.

Macak *et al.* [32] applied process mining to Git logs from student repositories motivated to understand the students' development process for improving a Software Engineering course. In the pre-processing step for process mining, Macak *et al.* [32] classified each commit based on whether it added or deleted code, contained test code, and commit size (small, medium, or large), and whether a branch was used or not. The main problems Macak *et al.* found in the student's development process were: 1) Lack of systematic testing, 2) Inadequate use of branches, 3) Too much last-minute work before the deadline, and 4) Uneven workload between group members.

Baumstark and Orsega [33] researched the development process of introductory Computer Science (CS) students. Data from the students' development process was mined from their Mercurial VCS repositories. Baumstark and Orsega found that the students generally produced small and focused commits, but similarly to Macak *et al.*, varying testing practices were found. Baumstark and Orsega plan to implement a teaching tool giving the students feedback on their development process as part of future work.

Rein *et al.* [34] wrote a master thesis on improving course management and student supervision with data from Git and learning management systems (LMSs). They created a web application to support the management of software development courses, including visualizations of groups' GitLab projects. The visualizations were made to give course staff an overview of all the progress of the

groups with an ability to drill down to visualizing individual group members' contributions. It was made for facilitating course staff in supervision and guidance of students and not for facilitation of students learning. Rein *et al.* evaluated the application with expert interviews and focus groups on course coordinators and Teaching Assistants (TAs) from a selection of software development courses at NTNU, including IT1901 and TDT4140. Rein *et al.* found indications that their application gives students improved feedback from the course staff.

Hamer *et al.* [28] generated visualizations of Git metrics to offer insights to students' and teams' development practices and processes. The contribution metrics focused on inequality in contributions and distribution of work over time. Hamer *et al.* found that all the projects they monitored had inequalities and that students tend to work near deadlines. Hamer *et al.* argued that the insights from the visualizations could aid course staff in course improvement and grading processes. Further, it is argued that the objective feedback from the visualizations can be valuable to students and their development skills.

3.2.2 Analytics for evaluation of student contributions

Group work is a staple of Software Engineering education [5]. It is, however, inherently difficult for course staff to evaluate such group projects due to the difficulty of contributions from individuals versus from the team [28]. Applying Learning Analytics to the assessment process has been investigated in several papers.

Parizi *et al.* [30] proposed in a conference lecture, a model for evaluating team members contribution based on objective metrics from Git repositories. Their model used the following metrics: *Number of commits*, *Number of merges*, *Number of files*, *Total lines of code*, and *Time spent on the project per day*.

Time spent on project per day is estimated using the algorithm *git-hours*[3]. This algorithm estimates the number of working hours by summarizing working sessions in Git. In order to identify a working session, all commits in a repository are listed, comparing the time difference between them. Then, using a pre-defined threshold, split the sets of commits by this threshold and time differences. Parizi *et al.* used `2 hours` as the threshold because longer coding sessions would be unrealistic. Afterward, the authors should have an overview of all working sessions per author. These metrics were used individually and in combination to score each developer numerically. Their score was ranked by the grades: Excellent, Good, Satisfactory, Poor, and Unacceptable. When metrics are combined, these scores are averaged. A deficiency of ranking developers on these performances is its simplicity and does not consider the difficulty of work. The authors propose a difficulty gauge for difficulty and contribution.

[3] `https://github.com/kimmobrunfeldt/git-hours`

The metrics proposed by Parizi *et al.* can be considered intuitive and reusable for multiple use cases. In addition, their Git mining system could be reused as a data analysis platform. The ranking system is a good starting point. However, the ranking formula is too simplistic for most projects or provides precise grading guidelines. Even though the authors introduced a difficulty gauge, they do not provide sufficient evidence that this will adequately catch all nuances in building a project.

Buffardi [29] explored automated evaluation of individual contributions to collaborative Software Engineering projects. To do so, the groups' Kanban boards were investigated in addition to their Git logs. Buffardi introduced metrics that ranked the individual team members by comparing them to the rest of the group. This was done using quantitative metrics such as *Relative Contribution Share*, *Relative Commit Share*, and *Relative Story Share*. These metrics highlight how many code lines, commits, and kanban stories each group member has added/completed relative to the group's total.

It was found that there are inconsistencies between the quantitative measurements and the subjective assessments of the students. Buffardi argued that using performance metrics to assess individual contributions may provide a more objective assessment than a traditional subjective instructor assessment. However, the metrics have limitations. He argues that no metric which perfectly reflects contribution value exists. The paper further discussed a limitation to using metrics for assessment situations, referred to as *Campbell's law* [35]. This is an effect where such measurements can cause students to manipulate the measurements for their own gain when used in an assessment situation. Conclusively, Buffardi recommends that quantitative performance metrics be used to supplement and validate the traditional assessment process.

A limitation to Buffardi's research is that it was based on a single instructor's evaluations from a single course. As a response to this, Hundhausen *et al.* [36] replicated Buffardi's research to test the robustness by collecting data from three different instructors in three different courses. As a result, they could replicate most of Buffardi's key results. Additionally, Hundhausen *et al.* were able to find a stronger correlation between the objective and subjective evaluations, indicating that quantitative metrics might be more applicable than previously outlined by Buffardi. However, Hundhausen *et al.* also points out that there are limitations to the quantitative metrics. They argue that log data on GitHub activities, such as activities regarding GitHub issues and code reviews, could be valuable for future work.

Chen *et al.* [31] developed an automated programming assessment system. The system takes code quality into account, as well as code contributions. To measure code quality, they used static code analysis with Linters and SonarQube[4].

[4] `https://www.sonarqube.org/`

SonarQube is an advanced code quality analysis tool. However, the code quality assessment's efficacy has not been evaluated. To quantify contribution, Chen *et al.* introduced the metrics *Contribution Index* and *Contribution max-difference*. These are more complex metrics to measure individual student's contributions to a team.

Contribution Index is the ratio between a student's contributions and the team's average contributions, measured by a specified metric M. By using commits, LoC, or issues/stories as M, Contribution Index becomes very similar to *Relative Commit Share*, *Relative Change Share*, and *Relative Story Share* outlined by Buffardi [29].

Contribution max-difference measures the difference in contributions from a single student and the highest contributing member in the team for metric M.

Chen *et al.*'s system is aimed at course-staff of Software Engineering courses to help manage group-based projects. The system uses data from Git and a Continuous Integration (CI) server. When evaluating the developed system, Chen *et al.* found it especially beneficial for detecting free-riders in group projects.

Guerrero-Higueras *et al.* [37] attempt to predict whether students' passed or failed in three different computer science courses. To do this, they tested a wide range of machine learning models. The input data used for the predictions were metrics from Git and the students' Github Classroom repositories. The features included *Number of commits*, LoC, *Number of issues created*, and *Number of issues closed*. Guerrero-Higueras *et al.* also used *Number of days where at least one commit was done* as a feature to their model. This feature was the most discriminant feature meaning the most important feature to their predictive model in front of LoC and *Number of commits*.

3.2.3 Application of analytics to improve student learning

As suggested by Hamer *et al.* in Section 3.2.1, insights into students' Git repository data can be valuable for students in Software Engineering courses.

Eraslan *et al.* [39] researched the effect of using GitLab metrics in consultation sessions in a software engineering course. The consultation sessions were arranged twice during the semester, and the students were shown a report of GitLab metrics specific to their group. The metrics focused on assigned and completed GitLab issues. They found that almost all students were satisfied with the consultation sessions through interviews. Although some metrics, such as *number of commits*, were too simplistic. Code quality and amount of code were of higher value than the number of commits due to differences in commit behavior. A comparative analysis of the first year with such consultations and the preceding year found students much more likely to pass the course. However, this is not verifiable with the current data and hence only indicative.

Gustavsson and Brohede [40] described an assessment tool for continuously assessing students' performance in Software Engineering projects. The as-

sessment tool provided support for both students and course staff. The tool shows contribution metrics for each student and a ranking for each contribution. This conveys how the student compares to the rest of the class. The tool is implemented to get insight into individual students and has no functionality for groups. Gustavsson and Brohede argues that continuous feedback to students can help ensure students actively participate in the project. Further, they argue that weekly data collection is a suitable data collection frequency for their application in a large 10/week-long Software Engineering course.

Gary and Xavier [41] presented a monitoring tool, which provided students with continuous feedback. The tool gave students insight alongside a project by visualizing activities and progress. The data is pulled daily from GitHub for code activity and Scrumwise[5] for task management activity. From the tool, the students can see and compare their individual performance, team performance, and the average of the entire class. The tool was overall found helpful by the students surveyed.

Haugse and Aalberg [1] explored how to best visualize Git data and what the visualizations had to offer for both students and course staff. They implemented a custom-built prototype of a visualization tool and evaluated it on course staff and students from two courses at NTNU, TDT4140 Software Engineering (Section 2.3.2) and IT2810 Web Development[6]. The tool used GitLab data for visualizations, including insight into code contributions, commits, issues, and merge requests. Through interviews, Haugse and Aalberg found support for the relevance of a visualization tool, giving students and course staff insight into work and work patterns. The visualization tool did not support pair programming, which the interviews found to cause a wrongful impression of workload imbalance among several students.

Tarmazdi *et al.* [6] contributes with a dashboard to help educators monitor teams in real-time. It allows for visualization of team mood, role distribution, and emotional climate. The authors are motivated by the need to learn teamwork in Computer Science (CS) studies to prepare students for industry properly.

The dashboard is founded on learning theory, elements of learning analytics, and teamwork models. A case study was conducted on a third-year university course, where 2 to 4 students cooperated on group-based assignments. Using Piazza[7] as input data for each team discussion, sentiment analysis was conducted to identify each student's role and discover tensions. For example, the lecturer in the course could use the dashboard to identify which groups had not started on an assignment yet, and identify any students showing frustration or anger. The lecturer could use this information to provide more targeted guidance to students.

[5] `https://www.scrumwise.com/`

[6] `https://www.ntnu.edu/studies/courses/IT2810`

[7] `https://piazza.com/`

Tarmazdi *et al.* demonstrates the value of using analysis tools courses to better aid learning environments. However, in this case study, they only receive feedback from one lecturer's experience, not including the students' role. Therefore, assessing the value from the students' perspective or how Piazza as a discussion tool and the analysis of what they spoke affected them is difficult. Furthermore, even though this tool has the potential to help student groups resolve tensions before they manifest into more significant conflicts, it also requires that all discussions are done in writing.

3.3 Summary of metrics from the literature

Notable metrics from the literature in this chapter are synthesized into Table 3.2, giving an overview of what has been prioritized.

Metric	**Referenced by**
Number of Commits	[1, 28–30, 32, 34, 36, 37, 39–41]
Lines of Code (LoC)	[1, 29, 30, 32–34, 36, 37, 39–41]
Number of issues closed/done	[1, 29, 34, 36, 37, 39]
Number of issues created	[1, 34, 37, 40]
Contribution Index and Contribution max-difference	[31][29, 36]*
Time Spent on a Project per day	[28, 30]
Number of Review Request	[1, 30, 34]
Number of days where at least one commit was done	[37]
Number of issues assigned	[34, 39]
Number of issue comments	[40]
Group attendance	[39]
Issue state by type over time	[41]
Code quality measure	[31]
Commit Size (S/M/L)	[32]
Number of Files created/owned	[30]

Table 3.2: Metrics used in the literature. Each metric can be drilled down to student level. * very similar, but not identical metric.

Chapter 4

Methodology

This chapter elaborates on the process and methods used to answer the research questions (Section 1.2.2), describes what it is, and discuss the choices made. The project timeline of the thesis is illustrated in Figure 4.1, giving a general overview of the overall process.

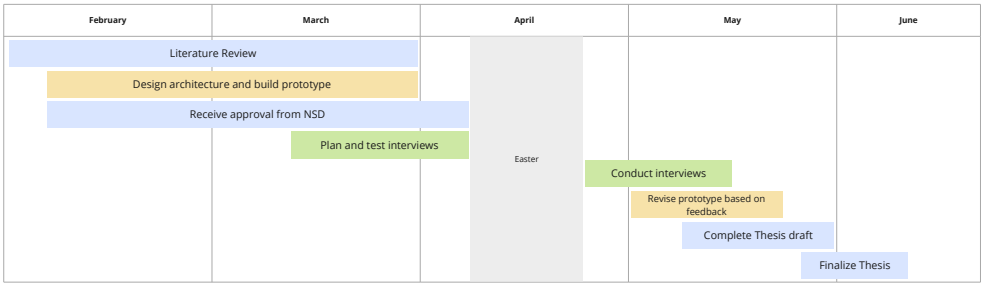


Figure 4.1: The project's planned roadmap through the semester. Tasks related to general research and documentation are colored blue, implementation focused work are colored yellow, and work related to interviews and feedback gathering are colored green.

4.1 Research process

The research process applied in this thesis, illustrated in Figure 4.2, shows from where research questions are derived, what strategy is used to answer them, including methods to analyze and validate these answers.

Motivation behind each Research Question (Chapter 1) is a combination of *Experience & motivation* and *Literature review*. Experience by course staff and students who have taken the course IT1901, serves as the primary motivation.

The *Literature review* was conducted using the strategy Snowballing [26], detailed in Section 3.1. Most literature primarily focuses on the evaluation of students and not tools to help guide them [2, 30, 42, 43]. Literature focused on better

guidance to students similarly has the lecturer as the target instead of the students [6].

Design and creation is used as a strategy to build a prototype tool to help answer *RQ1*. This process is explained in detailed in Section 4.3.

Data generation methods and *Data analysis* are used to generate necessary evidence of our contributions [44]. Section 4.4 describes which methods have been selected.

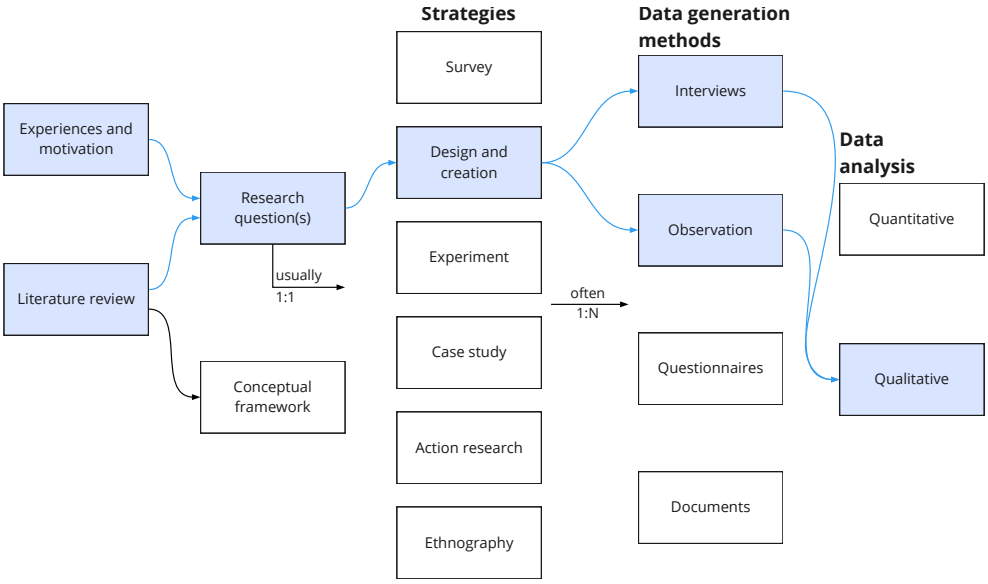


Figure 4.2: The research process used in this thesis. The figure is adapted from [44], where the colored boxes represent our process.

4.2 Research ethics

Given that personal information and behavior are processed in this thesis, ethics and privacy are key concerns. Unless there are strong reasons against it, research participants have a minimum set of rights that must be considered[1]. Norwegian Centre for Research Data (NSD) enables responsible use of personally identifiable information in research. All research in Norway must be approved by NSD beforehand if processed information is directly or indirectly able to identify a person.

Two types of personal information are processed: *1) Git and GitLab usage for student groups in IT1901*, and *2) Contact information and voice recording from interviews*. A single application to NSD was submitted at the beginning of the research process, with an expected processing time of around one month. In accordance with the NSD approval (Appendix C), all personally identifiable information is stored on NTNU approved software, servers, and devices.

[1] `https://www.nsd.no/personverntjenester/oppslagsverk-for-personvern-i-forskning/rettighetene-til-de-registrerte`

4.3 Design and Creation strategy

Design and Creation is described by Oates [44, p.108] as a research strategy focused on developing new IT products called artifacts. Different artifacts can be used depending on what the research is expected to contribute.

Constructs are concepts used in IT-related domains, such as the notion of entities or data flows.

Models combines constructs that represent a situation. It is used to aid in understanding problems and developing a solution.

Methods are stages in processes to solve problems with the help of IT. A method can be a strict formal algorithm or informal practice descriptions gathered from experience.

Instantiations is "*a working system used to demonstrate that construct, models, methods, ideas, genres or theories can be implemented in a computer-based system*" - Oates [44]

Parts of this thesis fall under the Instantiation artifact. In order to answer *RQ1* and *RQ2*, a functional prototype is constructed using a third-party visualization tool. Through Semi-structured Interviews and user tests, the prototype is expected to demonstrate how students would value such a platform qualitatively. Additionally, the experience from building the prototype, with feedback from students and TAs, should identify the benefits and limitations of using a third-party tool. Therefore, our prototype should be described as a *vehicle for something else*.

Although the prototype has the possibility of being used directly and immediately, the *contribution* mainly is the feedback received from the interviews on the demonstrated insights/graphs and how students find them valuable (discussed in Chapter 7). How a complete platform should be structured and developed highly depends on the organization, course, and end-users. For example, a particular organization may have an existing platform and wish to integrate these insights. Others may want to utilize third-party graph solutions, to reduce maintenance work and maximize user flexibility. The organization maintaining such a platform should also base its development on acknowledged Software Development practices, such as DevOps [20, 45] and Agile processes.

As previously mentioned, this thesis uses data from the course IT1901, which is available from GitLab. Other courses, however, may use Github or Bitbucket, which require the data to be extracted differently. Therefore, there is no one-size-fits-all solution, and each organization has to adapt the contributions from this thesis into its domain.

4.3.1 Conducting Design and Creation research

The Design and Creation research utilizes an iterative approach consisting of five general phases. Figure 4.3 give a general overview of each phase in this thesis and the iterations done after two rounds of interviews (*Evaluation* phase). A detailed roadmap is presented in Figure 4.1.

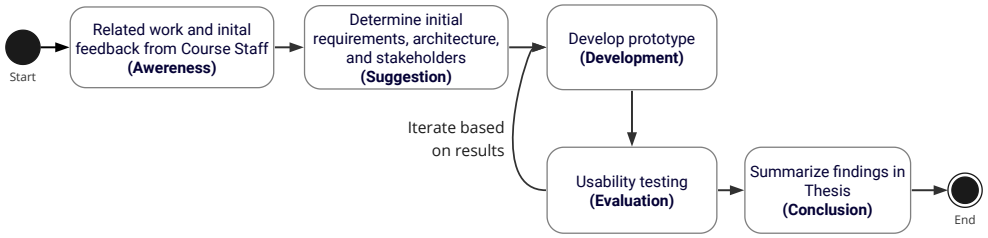


Figure 4.3: The Design and Creation research process structured in this thesis. Development and Evaluation are done in two rounds, allowing for refinements to the prototype

1) Awareness recognizes and articulates a problem based on existing literature, experience from practitioners, or the needs of clients.

In this phase our thesis explore best practices in Git and GitLab, and use cases from course staff in IT1901, see Chapter 2. Next, related work (Chapter 3) explore metrics from similar use case in other literature.

Exploration into the Git and GitLab behavior of students in IT1901 were included in the latter half of the thesis because access to the data required approval from Norwegian Centre for Research Data (NSD). A temporary solution was to use our GitLab repositories from earlier courses and create a couple of simulated GitLab repositories with known anti-patterns. However, the usefulness of repositories created by our projects is limited in providing insight, with the risk of being biased.

The data from these steps set the baseline for the initial prototype. This thesis mainly targets the student role while ensuring that course staff access the same information. However, in situations where the usefulness of a metric conflicts with these stakeholders, the students are generally favored. Distinguishing between metrics used to evaluate and guide the student is necessary because the former is rarely expected to be read directly by students and present them with actionable or understandable information on how to improve.

2) Suggestion "involves a creative leap from curiosity about the problem to offering a very tentative idea of how the problem might be addressed" [44, p. 112].

This phase focused on exploring the benefit and limitations of third-party visualization tools (Section 5.4.1). Next, these candidates were filtered down to one system based on the requirements, constraints, and needs identified in the *Awareness* phase. Given the similarities between each solution, we expect to get more

value by testing and improving the visualizations on one system instead of having to test the same layout and data on all third-party solutions.

3) Development is the phase where the design is implemented, based on insight from the *Suggestion* and *Awareness* phases. Chapter 5 go into the details of the prototype design.

This thesis plan to run two iterations of the Development phase, capturing any feedback received from interviews (*Evaluation*). The first iteration focus on the process of extracting data from GitLab. On the visualization tool, work involved iterations on different presentation strategies and discussing how to extract actionable insight from specific data. The second iteration focused on improving the visualization tool based on student feedback and Teaching Assistants (TAs). This allows us to resolve solvable problems and get higher-quality data from the final round of interviews.

4) Evaluation examines the artifact to assess and determine its value. Section 4.4 describes the evaluation strategy in detail.

5) Conclusion consolidates results from the process and what knowledge has been gained. Results from the interview are summarized in Chapter 6 and discussed in Chapter 7. Our thesis expects to have gained knowledge on *RQ1* and *RQ2* after completion of this research process.

4.4 Data generation methods

To learn what visualizations are perceived as valuable for students (*RQ1* and *RQ2*), we favor qualitative data generation that allows for reflection from the participants and asking open-ended questions. Therefore, *Semi-Structured Interviews* is selected as the primary Data generation method for its flexibility and efficiency in extracting data on participants' experience [44, 46]. However, Semi-Structured Interviews have some limitations, particularly regarding the time required to transcribe and analyze the results after each interview and the time required to plan which topic to talk about properly and in what order.

The complete interview is structured as a Usability test [47, 48], with a pre-interview and post-interview to get relevant feedback on what visualizations students value and how to improve the usability of the prototype/product [47]. It gives the participants first-hand insight into what a solution may look like, and the thesis with observable insight into how each participant uses the prototype. Considering usability testing is an exercise of observing target users use a product in a covert or non-covert situation, it can be argued to have similarities with the *Observation* Data generation method. Observations tell us *what* happened, not *why* [44]. Similarly, the user testing tasks mainly tell us *what* the participant did or did not do. Hence, the Data generation methods selected are a combination

of *Interviews* and *Observations* through usability testing. Subsequently, research results should provide insight into *how* the participant used the prototype and reflection on *why* the person did specific actions.

In this case, Observations have certain limitations that should be considered. First, IT1901 are taught during the fall semester, preventing this thesis from observing students in a natural setting. Each participant is presented with a situation and task similar to the project they worked with during IT1901. It is, however, still a simulated situation.

4.4.1 Participants and Recruitment

The target was to conduct around 12 interviews, separated into two iterations, where 80−85% of the candidates had the role of student. Students were selected from the fall semester of 2021 that had taken the course IT1901. In addition, TAs were included, even though students are the primary stakeholders because TAs are expected to provide another relevant point-of-view to the student projects.

This target group represents a fairly homogeneous group. All study the same or similar study and take similar courses. Given these similarities, the limited scope and functionality of the prototype, and the strategy to run interviews in two iterations, we expect around 12 respondents should provide a sufficient basis for our findings. Nielsen and Landauer [49] argue that 5 participants in usability testing should be sufficient to uncover 85% of the problems, even though Spool and Schroeder [50] argue that serious usability problems are typically found after 13 or 15 tests. However, these experiments tested systems with more functionality requiring each participant to do more steps to complete a task. The prototype presented to students and TAs in this thesis has less functionality and is focused on one dashboard. Lastly, by splitting the interviews into two rounds, any fixable usability problems could be resolved on the prototype before the last interviews, better utilizing the responses. Different participants were used in each round, ensuring no participants had any pre-existing knowledge or bias about the prototype.

Publishing was done through NTNUs Blackboard solution. Students that were interested in participating were encouraged to contact us by email. Similarly, TAs were contacted directly by email from a contact list given by the course staff. To further incentivize participation, every participant was given a gift card worth 200NOK. All the interview participants and the interview leader and referee spoke Norwegian as their primary language. The interviews were, therefore, all carried out in Norwegian.

4.4.2 Interview guide

Summarized, the usability testing is composed of three main parts, *1) Pre-interview, background information*, *2) Usability task to test the prototype*, and *3) Post-interview, gather reflections and feedback of the prototype*. Pre- and post-interviews are structured as *Semi-Structured Interviews* based on the guidelines from Oates [44] and

Adams [46]. The usability task follow the guidelines from [47, 48]. In total the usability testing lasts between 30 to 45 minutes.

Students and TAs follow the same general structure during the usability test. However, the TA usability tests are oriented around how the person counseled student groups in IT1901, what challenges students typically had during the project, and how the proposed tool could help simplify the guidance process. Appendix B (written in Norwegian) show the original interview guide for TAs. Students' usability tests focus on their own and group's experience from Software Engineering courses, with particular focus on IT1901, and how the proposed tool could improve their product and learning process. Appendix A (written in Norwegian) show the original interview guide for students.

Before the actual usability testing started, the interview guide was tested on similar target audiences to verify the overall structure and flow of the Interview guide. The flow should encourage natural conversation inside each part and avoid abrupt changes in theme. Also, questions around benefits and limitations should encourage the participant to explain its reasoning, be given in a natural order [46], and avoid pressuring the participant to give a specific answer.

Part 1) Pre-interview

This part aims to help us understand the student's pre-existing knowledge of Software Engineering, either through practical work or previous courses. Additionally, it presents insights into how they worked during the course IT1901, what features of GitLab they used, and other notable aspects. Finally, this insight is used to validate the general flow and structure of the prototype, which should resemble the expected flow of the course and what functionality is used.

Part 2) Usability testing

Usability testing is the main *Observation* phase, where the prototype is tested. Here we expect to discover problems with the Graphical User Interface (GUI), ambiguous formulations, and finally, which features the student mainly highlights when navigating. The design principles from Norman [51] are used as general guidelines for what to look for when observing the participant. Additionally, we expect a clearer idea of which metrics students perceived as valuable.

Respondents are presented with a fictional case, similar to an actual project in IT1901. Their task is to look around the prototype (see Section 5.5 for screenshots) and identify what should be improved and why it should be improved. The English-translated case presented to both students and TAs are:

> You are part of a group with *two* other students working on creating a site for hotel bookings as part of a university course. The project is separated by *three* deliveries, and you use GitLab and Git as the

> primary cooperation tools. The course assesses the code delivered and how you have cooperated.
>
> You have just completed your *second* delivery and are about to begin the *third* and last one. The lecturer has published a message on Blackboard encouraging you to use a new tool to get an overview of your project and identify where you can improve how you work and cooperate using Git and GitLab. You visit the tool from the browser, sign in to your NTNU account, and start looking.

After the case and task were presented, the participant was given a computer with the prototype ready (See Section 5.5), signed into a demo account, and encouraged to think aloud when interacting with the prototype. The demo account has equal access to what a student account would have to limit the risk of confusion and out-of-scope interactions.

Fictional data, constructed by real data from previous semesters of IT1901 and other relevant courses, is presented in the prototype. Examples include a difference in contribution per member, difference in commit sizes and lengths, varied practices of GitLab issues, Merge Request, and Milestones. The goal was to show realistic behavior similar to an average group in IT1901. Alternatively, real and anonymized data could give more realistic behavior. However, there would be a risk that a participant could identify its own or other groups based on specific data points or behavior. This could also become problematic in terms of privacy and the approved NSD application[2]. Therefore, it was safer to use the former approach.

The interview leader and secondary interviewer have two distinct roles during the usability task. The leader observes the participant's interactions directly and asks open-ended follow-up questions to facilitate reflection from the participant. The secondary interviewer has a more silent role, focused on observing nonverbal behavior that may hint at invisible elements, affordance mismatch, lack of feedback, unnatural flow in the Graphical User Interface (GUI), or other problems could be attributed to problematic design. In addition, through screen mirroring, the secondary interviewer has visual access to what the participant does without obstructing the participant and interview leader.

Part 3) Post-interview

When the usability task is done, the usability testing transitions to the post-interview. This part focus on getting the respondents to reflect on the *usefulness* and *limitations* of the information presented in the prototype and the prototype in general.

[2]NSD's criteria for de-identified information: `https://www.nsd.no/personverntjenester/oppslagsverk-for-personvern-i-forskning/hvordan-gjennomfore-et-prosjekt-uten-a-behandle-personopplysninger/` (only available in Norwegian)

Additionally, we gather feedback on how they expect to use such a system if it were available during their project in IT1901. Finally, questions are ordered in a way that is perceived as natural for the participants [46] and to avoid pressuring the participant to give a specific answer.

4.5 Interview transcription

Audio tapes were used to gather verbal data from each interview, giving a complete record of everything that was said and allowing the interview leader to concentrate on the interview process [44]. Even though notes were taken in parallel by the secondary interviewer, these were focused on capturing non-verbal notes and other notable insights discovered, which audio recording fails to capture. Video recording and eye-tracking systems were discarded because they were considered too intrusive in the interviews without providing sufficient benefits. In addition, as stated by Oates [44], many interviewees are reluctant to be filmed, which could result in even fewer participants and negatively affect the validity.

Each recording was passed to a transcription tool in Microsoft Word, giving a rough excerpt of the interview. Then the results were manually corrected, cleaned, aggregated, and anonymized. Student respondents were labeled as `Student<ID>`, such as `Student1` and `Student2`. Similarly, TAs were labeled as `TA<ID>`. The interview leader and referee were labeled by name.

4.6 Data analysis

To analyze the transcribed interviews, NVivo[3] was used for manual coding of the data and performing systematic analysis, following the guidelines from [44, 52]. NVivo is a software tool for qualitative data analysis, with advanced filters to query the codes. *Coding* is a process of assigning and grouping different parts of the data with codes(labels). The idea is to move from raw text to structured research concerns in small steps [52].

A code was created for every metric on the dashboard, making it possible to quickly get an overview of all remarks on a single panel on the Visualization tool. In addition, codes were created for any expressions of how often the interview objects reckoned they would use the Visualization tool during a project, general positive or negative remarks, and potential for improvements.

[3] `https://www.qsrinternational.com/nvivo-qualitative-data-analysis-software/home`

Chapter 5

Prototype design

This chapter dive into the technical specificity of designing and building the prototype used to answer *RQ1* and *RQ2*. Section 5.2 describes the overall architecture of a Git and GitLab analysis tool. Section 5.3 describes how metrics from data sources are aggregated and stored. Section 5.4 describes the design and evolution of the Visualization tool used in interviews, and Section 5.4.1 explores the benefits and limitations of different third-party visualization software.

It should be noted that the solutions selected in this chapter are customized to constraints specified in Table 5.1, relevant metrics from Chapter 2, and use cases of IT1901 and its data sources. Even though the solution uses general-purpose solutions with high customizability, other courses may benefit from using other systems or existing data sources.

5.1 Constraints

Conventionally, tools used in university courses are restricted by privacy concerns to protect students' data and identity. Additionally, it is expected that a visualization tool targeted toward students has to be easily accessible, requiring minimal setup and configuration. Table 5.1 lists key constraints a visualization platform should consider, using common Quality Attribute metrics as categorization.

5.2 Architecture

During the awareness phase, two distinct responsibilities were identified: *1) Loading and storing metrics from data sources, such as GitLab*, and *2) visualizing metrics aggregated as insights for students*.

Data loading and aggregating have entirely different sets of constraints and expectations, requiring a different architecture when compared to a *Visualization tool*. The latter is, in principle, a dashboard/statistics system, and the responsibility can mostly be resolved using existing third-party software (Section 5.4.1).

Constraint	Quality Attributes
The visualization tool should not require students to download any software	Usability
The visualization tool should support Single Sign-On (SSO) authentication, to utilize students university account	Usability, Security
Students' access should be limited to see data from their own group	Security
Course Staff should have access to all groups	Usability
Should allow for an automated way to manage users and access	Security, Maintainability

Table 5.1: Architectural constraints
Constraints that the visualization tool should consider

The benefit of splitting these systems into two components is the flexibility to replace one without drastically affecting the other components. It also prevents architectural limitations of one component from limiting the other.

Figure 5.1 illustrate the general distinction of responsibilities and how the overall system should interact. First, *Data sources* are any source of data that can be used to give *Consumers* valuable insights into their project. For example, in IT1901, GitLab is the primary data source, while another project may use Github and Trello. Second, *Producer* are components that produce valuable insights to *Consumers* based on data from one or multiple *Data sources*.

The primary interface between the two components was chosen as SQL using a dedicated database (Figure 5.2). An alternative would be a REST-API. However, it would limit which visualization tools to select. Coupling components at the database level are generally anti-pattern. However, in this setup, the flexibility of SQL outweighs the disadvantages. Furthermore, given that write access is exclusively restricted to the Metric aggregator, this should limit the worst consequences of database coupling.

5.3 Metric aggregator

Internally, the metrics aggregator consists of two systems: 1) The code sourcing data and aggregating it, and 2) a database where the aggregated metrics are stored.

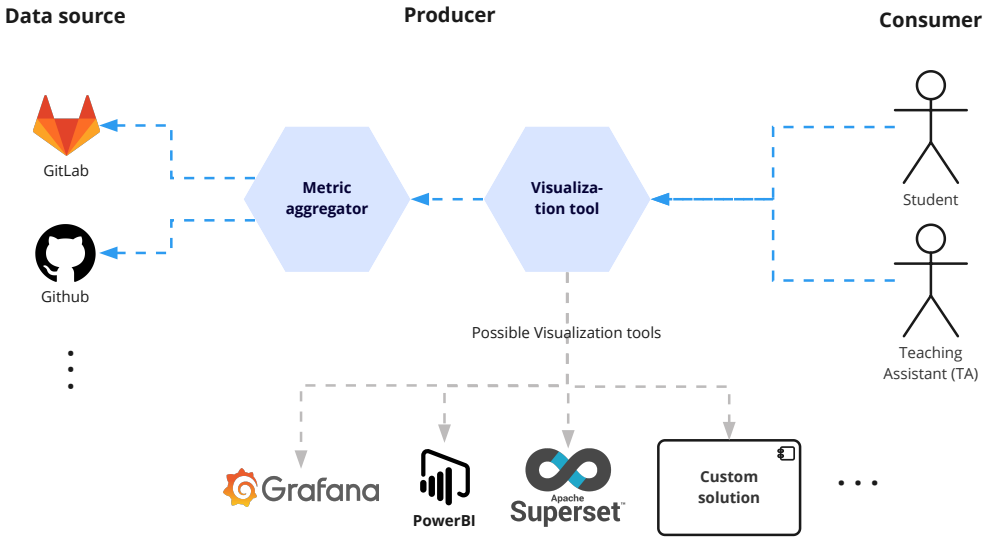


Figure 5.1: Architectural overview of the primary responsibilities and their dependencies

5.3.1 Database alternatives

This thesis selected PostgreSQL as a database solution for the prototype. It was selected because it provided a flexible SQL interface and was easy to set up and maintain while giving flexibility on how the data could be structured. Other alternatives were considered, such as Elasticsearch, AWS Redshift, Spark, InfluxDB, and Prometheus. However, all required more complicated infrastructure or were more costly.

5.3.2 Metric aggregation architecture

Students expect fast feedback on changes to their projects, were waiting up to 24 hours or more is likely, not acceptable. Therefore, a software architecture that can handle continuous updates of many student projects have to be designed. Simultaneously, the Metric aggregation component must consider rate limits from GitLab. Each project in a course requires hundreds (approximately 300) of requests to retrieve all information needed, resulting in a high risk of being rate-limited by GitLab. Multiplying the number of requests per project with the number of projects in a single course (around 80 for IT1901) Equation (5.1), we are guaranteed to be rate-limited when every project is crawled simultaneously.

$$ApproximateNumberOfRequests = 300 \cdot 80 = 24000 \tag{5.1}$$

To remedy rate limits, manage internal resource usage, and facilitate future use cases, an event-driven architecture[1] has been adopted. The internal logical

[1] `https://docs.microsoft.com/en-us/azure/architecture/guide/architecture-styles/`

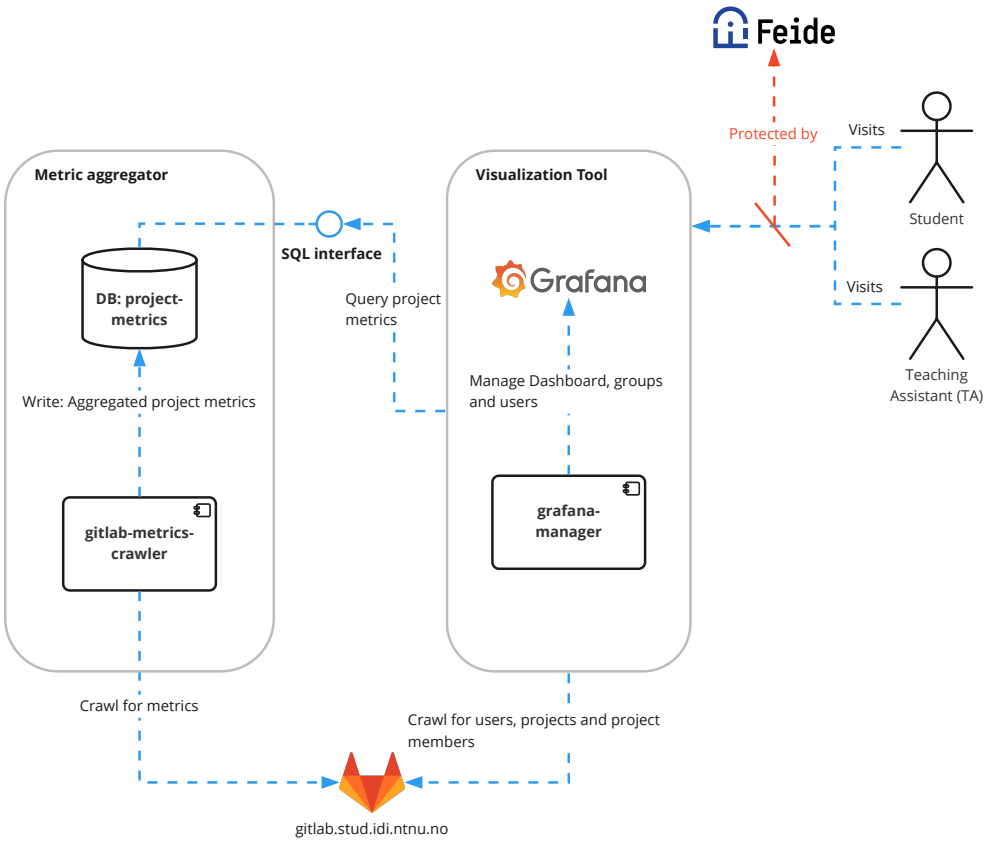


Figure 5.2: Diagram of architectural components. Drill down of Figure 5.1. This figure presents the components and interfaces this thesis has used to implement the prototype. GitLab is the Data source, a read-only PostgreSQL database provides an SQL interface to other components, and Grafana is used to provide statistics and insights to users.

structure of the Metric aggregator is illustrated in Figure 5.4. A combination of Publish-Subscribe (Pub-Sub) topics and message queues are constructed as a Fan-out pattern[2]. This pattern decouples the producer and Metric Aggregators (consumers) through a Pub-Sub topic. Additionally, the queues allow Metric Aggregators to "consume" messages in their tempo and retry failed jobs. If an aggregation job fails because of rate-limiting, it can retry later by polling the same message again. Figure 5.3 illustrates where rate-limiting may occur during aggregation of GitLab issues, and how it is handled.

Additional strategies are *caching of data*, and *retrieving only data that has changed*. Caching is excellent for reducing the number of requests when fetching the same information. The most straightforward approach is in-memory caching

event-driven

[2] https://learning.oreilly.com/library/view/serverless-design-patterns/9781788620642/3fa37b68-0490-4fa3-a749-13aa03eac59f.xhtml

to limit duplicated requests during a single processing run. However, long-term caching requires a third-party solution, such as Redis[3], or saving files on disk. Furthermore, additional mechanisms to invalidate the cached data are required to prevent stale data from being used. However, with a good cache invalidation strategy, the system can store unchanged data for a longer duration limiting the total number of requests to GitLab.

Retrieving changed data would ensure that only necessary requests are sent. However, it does require a more complex setup. GitLab's API has support for filters to receive only data that have been created or modified *after* a specified date and time[4]. Although reducing the total results returned, there is no clear indication from the documentation that this mechanism handles Git rebase[5] commands, potentially resulting in modified commits never being updated in the aggregated metrics.

An alternative strategy would be to listen for Webhook events[6], such as Push events. Then the metric aggregation can request details for only the changed information. This strategy can give students more immediate feedback on their changes, amplifying the feedback loop. Regarding load balancing, if updates are only triggered when changes are made and requesting details for the changed information may distribute the total load over a longer duration, given the data is received as a stream instead of in batches. However, the code must handle unexpected behaviors, such as Git rebase on old commits. In addition, some changes are not published to webhooks, such as modified comments[7] in an issue or Merge Request.

Webhooks and long-term caching were considered out of scope for this project, given that the data analyzed were not changing (the last code push was months ago). However, in a productionized setup, the developers should consider using a webhook to only trigger when changes are made.

The publisher's (`GitlabCrawler`) responsibility is to determine which project should be updated (re-crawled). This job is run regularly or can be triggered by events from GitLab. This thesis went for the former approach because of its simplicity. Projects that require an update are published to a Publish-Subscribe topic that every aggregator subscribes to.

Each Metric Aggregator are responsible for a specific metric, such as GitLab Issues or Commits, and are logically separated from each other. They own the whole metric pipeline, from the Data source down to the specific database table.

[3] `https://redis.io/`

[4] `https://docs.gitlab.com/ee/api/commits.html#list-repository-commits`

[5] `https://git-scm.com/docs/git-rebase`

[6] `https://docs.gitlab.com/ee/user/project/integrations/webhook_events.html`

[7] `https://docs.gitlab.com/ee/user/project/integrations/webhook_events.html#comment-events`

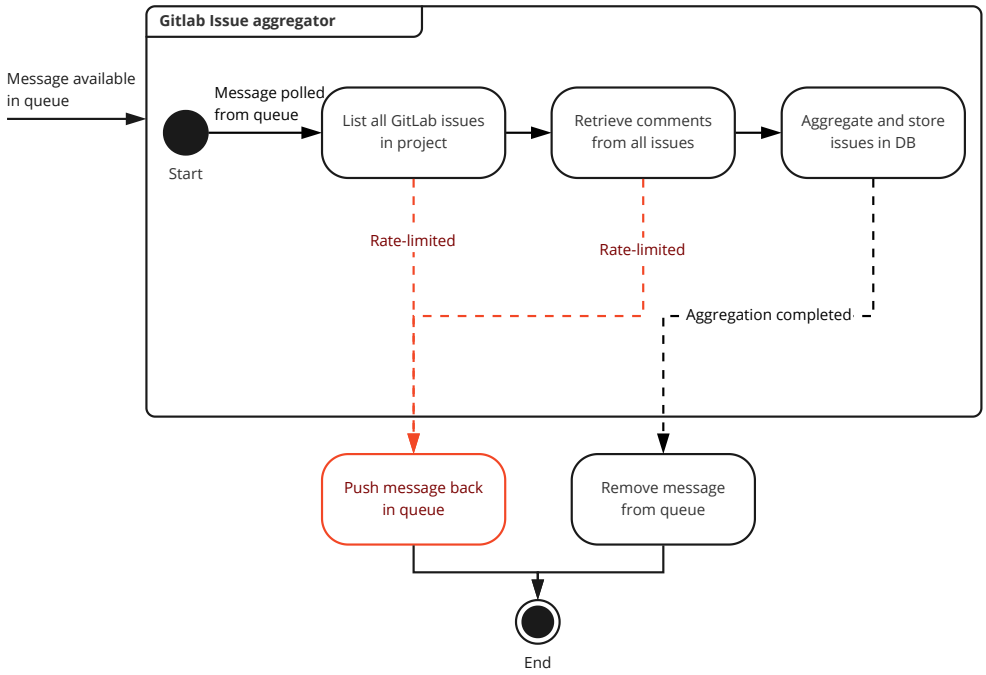


Figure 5.3: State diagram of rate-limiting
Simplified state diagram of how GitLab issues are aggregated, and how rate-limiting is handled.

In a Microservice oriented architecture, the publish-subscribe producers and consumers are often separate services [8]. For simplicity, our prototype has both the producer and consumers stored in the same system, using RabbitMQ[9] as a pub-sub broker. However, the proposed architecture is system and language agnostic, meaning that any pub-sub and queue systems can be used and implementation language and project structure.

5.3.3 Classifying code contribution for each student

The contribution classifier will, by best effort, look at which parts of the code each developer contributes to, down to each file changed in a commit. Conventional measurements are *number of commits to a file*, *lines added/removed to each file* (Table 3.2). Similarly, it should classify the number of commits or lines added/removed to tests, functional code, and documentation. However, as stated by Aivaloglou and Meulen [43] measuring contribution is not a precise measurement. Pair- and Mob programming will often only count contributions to one author, even though multiple people contributed. Similarly, many students commit with non-university emails, making it difficult to trace who the actual author is.

[8] `https://aws.amazon.com/pub-sub-messaging/`
[9] `https://www.rabbitmq.com/`

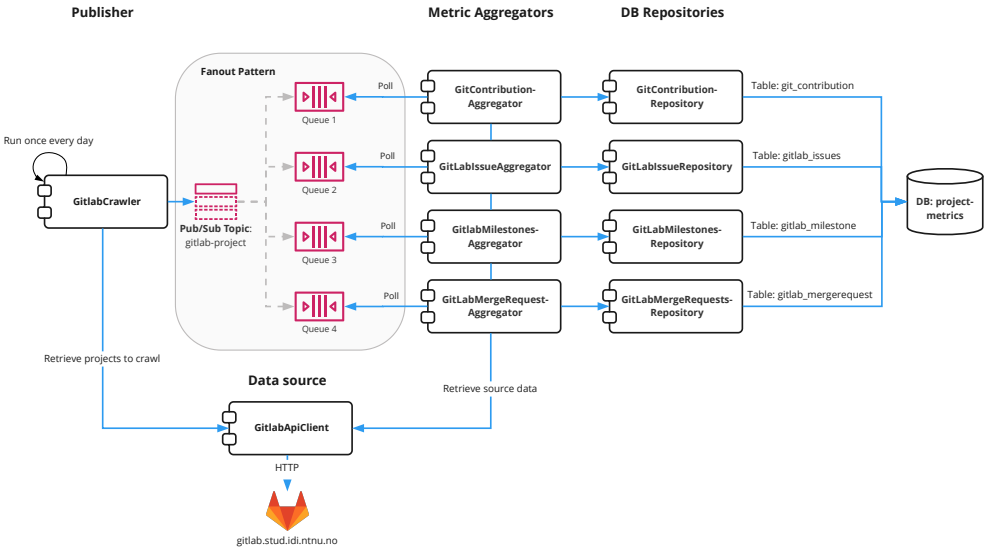


Figure 5.4: Diagram illustrating the internal flow of data inside the Metric aggregator. Drill down of Figure 5.2. Show how components communicate with Data sources and are written to the database. All metric aggregators are logically isolated from each other.

To reduce the likelihood of having more contributors than group members in a project, support for Git mailmap[10] was implemented. Students in the file `.mailmap` map their non-educational emails to the correct one, and the contribution classification will remap those.

Multiple contributors to the same commit are supported through Git's[11] co-authoring functionality. The system is designed to give co-authors the same amount of contributions. This is particularly valuable for pair programming, and a key concern for students [1].

Contribution to code types, such at tests, functional code, documentation and other, are done using knowledge about typical folder structures in the typical languages, and some educated guessing. Java based projects will store functional code inside `src/main/...` and tests inside `src/test/...`[12].

More educated guessing is required in JavaScript-based projects because no strict folder regime exists. However, a convention in Jest[13] is to store tests inside the folder `__tests__`, or suffixed with `.spec.js` or `.test.js`. These can be used as guidance to determine whether a file is part of a test or not.

[10] `https://git-scm.com/docs/gitmailmap`

[11] Example from GitHub: `https://docs.github.com/en/pull-requests/committing-changes-to-your-project/creating-and-editing-commits/creating-a-commit-with-multiple-authors`

[12] `https://cs.lmu.edu/~ray/notes/largejavaapps/`

[13] `https://jestjs.io/`

Documentation classification mainly look for markdown files, or files stored inside a folder named `doc` or `docs`. The latter is a convention recommended in IT1901.

5.4 Visualization tool

Part of this thesis (*RQ2*) is to consider how usable third-party visualization software is to present insight to students. This involves exploring what third-party software exists and comparing its pros and cons. Instead of prototyping and testing every software, one will be selected, allowing us to focus on incrementally improving the visualizations and metrics. Lastly, experience and feedback from using the developed prototype are discussed (Chapter 7), referencing the other third-party software.

5.4.1 Third-party Visualization software

Candidate software was found using a broad search on Google after statistics and metrics visualization software and included some systems that were known beforehand.

- Kibana[14]
- PowerBI[15]
- Tableau[16]
- Grafana[17]
- Apache SuperSet[18]
- Cyclotron.io[19]

Kibana is a dashboard solution purpose-built to communicate with Elasticsearch, with the ability to run ad-hoc queries, build and share the dashboard, and create alerts. To automate user management, Kibana exposes a REST API. SSO is configurable through security assertion markup language (SAML) or OpenID Connect. Kibana is limited to only Elasticsearch and no other data sources. In conjunction with Elasticsearch, a complete stack become complicated to maintain.

PowerBI is Microsoft's interactive visualization software. Customers can connect to different data sources and create interactive dashboards. The features a customer has access to are determined by which subscription they pay for. Customers must have a Windows computer and the Power BI desktop program to create dashboards. Dashboards are viewable from the browser, given that the data

[14] `https://www.elastic.co/kibana/`
[15] `https://powerbi.microsoft.com/`
[16] `https://www.tableau.com/`
[17] `https://grafana.com/`
[18] `https://superset.apache.org/`
[19] `https://www.cyclotron.io/`

source is available. Data sets can be pre-processed and uploaded. However, this requires the data to be regularly recalculated.

Tableau is proprietary software with similar capabilities as Microsoft PowerBI. However, the desktop client is available on Windows, Mac OS, and Linux.

Grafana is an open-source analytics and visualization platform. Like Kibana, the solution runs in the browser and supports SSO, giving the dashboard authors and users flexibility. However, Grafana has the added benefit of supporting multiple data sources. Administrators can chose to self-host Grafana, pay for managed hosting, or utilize the hosted version inside GitLab[20].

Apache SuperSet is an advanced open-source analysis tool comparable to Tableau and PowerBI. It claims ease of use, support for many data sources, and rich visualizations. Administrators can choose self-hosting or pay for a managed setup[21]. Compared to Grafana, SuperSet provides a higher degree of flexibility at the cost of complexity. Administrators need knowledge of Python and Flask[22] to configure a complete setup.

Cyclotron.io is an open-source dashboard solution with the ability to be extended with custom JavaScript and CSS. They claim this can be used to create dashboards that do not look like regular dashboards. However, Cyclotron does have some limitations: 1) requires a valid Highcharts license[23], 2) It is not designed to be exposed to the public internet as-is[24], 3) supports only Lightweight Directory Access Protocol (LDAP) and Active Directory as authentication solution, 4) does not support SQL databases as a data source, 5) Have little available documentation, and 6) the project have had no new releases since 2018.

5.4.2 Selecting a third-party visualization software

There is no definitive solution that is the most suitable for everyone. All listed solutions have benefits and limitations, making them suitable for different situations. For example, if a course already has licenses and knowledge on PowerBI or Tableau, these might be a better option. Similarly, if a course already uses Elasticsearch, then Kibana may be better.

In this thesis, low cost, easy maintenance, and fast prototyping were favorable, in addition to the constraints in Table 5.1. Their cost and need to download a program excluded PowerBI and Tableau. Kibana would require the setup of an Elasticsearch cluster, therefore increasing the maintenance cost. Similarly, Apache

[20] `https://docs.gitlab.com/omnibus/settings/grafana.html`
[21] `https://preset.io/`
[22] `https://flask.palletsprojects.com/en/2.1.x/`
[23] `https://shop.highcharts.com/`
[24] `https://www.cyclotron.io/faq.html#internet-use`

SuperSet is designed for more advanced data analytics situations, which would require a more complicated setup. Finally, Cyclotron.io is excluded because of its limitations. Concluding, based on exclusion, Grafana was the most suitable solution for further prototyping.

5.5 Prototype development

This section goes through the prototype development and how its structure changed between the initial prototype and the second round of interviews (Chapter 6).

5.5.1 Initial prototype

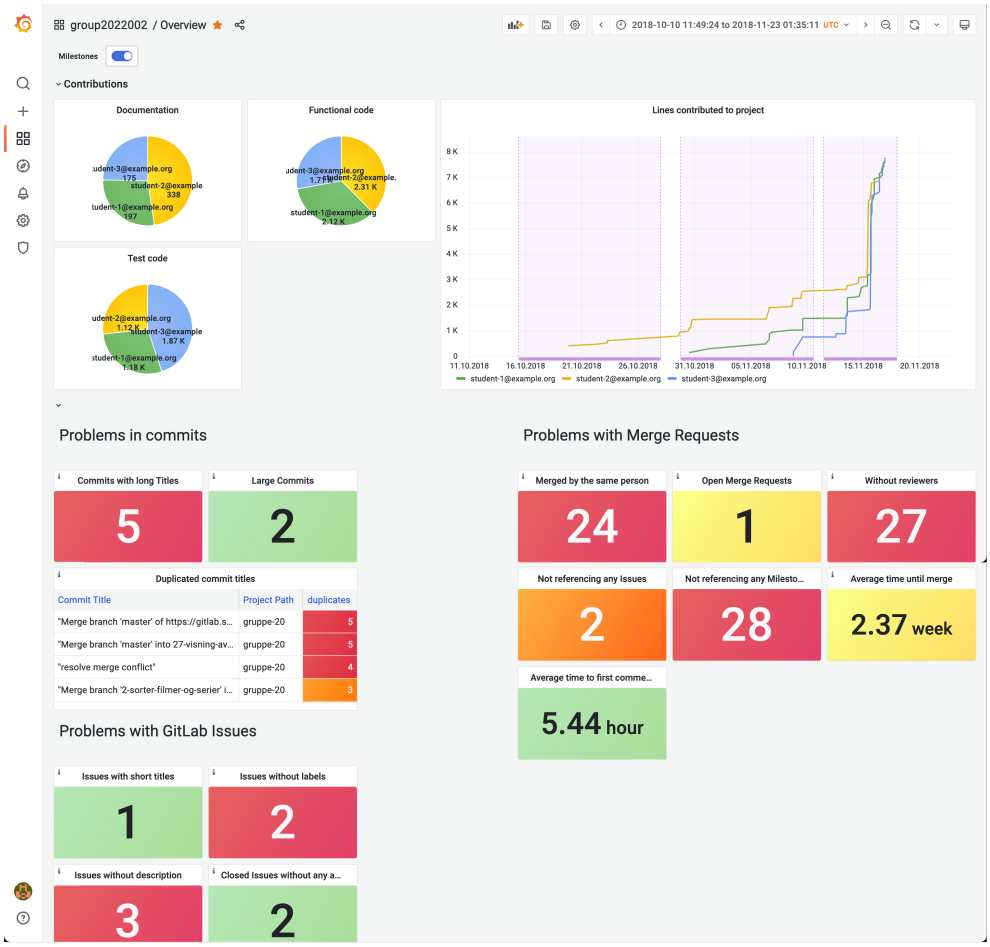


Figure 5.5: Full page screenshot of the initial prototype design. This was presented to students and TAs in the first interview round

The primary focus of the initial prototype was determining which metrics to include in the visualization. Screenshots are shown in Figure 5.5. Metrics from

the literature review (Table 3.2) were used as inspiration, even though most of the literature was used to evaluate students and was not necessarily directly valuable to them. In addition to metrics from literature, IT1901-specific metrics (Section 2.3.1) and best practices regarding Git commits (Table 2.1), GitLab Issues and Merge Request were included. Four categories of information were created, named *indicators*: 1) Contributions, 2) Git commits, 3) Merge Requests, and 4) GitLab Issues. Each should give an overall health indication for their respective categories and visualize how something should be improved.

Contributions indicates how each student has contributed to a project. The course staff wanted to see how each student has contributed over time and which parts of the code. A Cumulative Line Graph, counting *lines added*, is used to visualize the total contribution for each student (Figure 5.5). GitLab Milestones were overlaid as purple boxes, providing context to how contribution changed over time. In IT1901 students were encouraged to use Milestones to denote each delivery. This makes it easier to see whether certain students only contributed to one delivery or whether most of the work was done close to the deadline.

The Pie charts show how many lines each student have contributed to *1) Documentation*, *2) Functional code*, or *3) Test code*. These should help students discover potential imbalances and what parts of the code each student contributes.

The colors used for contributions were neutral[25] because the information was strictly informational. Whether imbalance in the contribution is problematic depends on the course and the goal of the student group. For example, in IT1901, every student should contribute equally to all parts. However, having more dedicated roles in other courses is more natural. There is a risk of measuring the wrong metric [20, p. 45], resulting in students focusing more on writing lines of code instead of contributing to improving the project.

"**Problems in commits**" focus on easy-to-measure metrics that give students actionable feedback. From commit best practices (Table 2.1), developers should prefer small single-purpose commits and keep the commit message length under 50 characters.

In Figure 5.5 commits that are likely too large or generic is visualized in the panel "*Large Commits*". We have defined a *commit* as large when it either touches many files or has added/removed many lines of code. The thresholds are 800 lines of code or 26 files touched. Merge commits generated by GitLab will often be classified as large and are therefore ignored. However, these thresholds are imprecise and often dependent on the project. More explicit language may, for example, yield more lines added/removed compared to smaller scripting languages.

"*Commits with long Titles*" counts the number of Git commits messages containing more than $50 + 10$ characters (10 characters added as an error margin),

[25] Grafana's Classical palette were used when denoting neutral colors. `https://grafana.com/docs/grafana/latest/visualizations/time-series/graph-color-scheme/`

inspired by the commit message guidelines (Table 2.2). Merge commits are ignored because GitLab's auto-generated message often breaks this rule.

"*Duplicated commit titles*" attempts to indicate how many Git commit messages that have exact duplicates. It can be argued that if a commit message is duplicated multiple times, the commit's purpose is not clearly defined, or the message is too vague, violating two of the Git best practices (Table 2.1). Furthermore, a large number of merge commits and "merge conflict" commits, as seen in Figure 5.5, can indicate that the students use long-lived branches and do not push to trunk frequently enough. This violates "Branch frequently, using short-lived branches" (Table 2.1).

"**Problems with Merge Requests**" visualize usage of Merge Requests that has the potential to become problematic. Such as Merge requests that have not received any code review by team members, violating Table 2.1 and the requirements set by IT1901 (Section 2.3.1). Furthermore, IT1901 also expected Merge Requests to reference either GitLab Issues or Milestones.

The panel "*Without reviewers*" indicates the lack of code review. "*Merged by the same person*" is adopted for IT1901 that wanted to encourage the code reviewer to merge the branch and not the author. In conjunction with these panels, "*Average time to first comment*" and "*Average time until merge*" give an averaged indication of how long the code review process takes in a group.

"*Not referencing any Issues*" and "*Not referencing any Milestones*" check the title and description of Merge Requests for whether they contain references to issues or milestones.

"**Problems with GitLab Issues**" look at practices that can become problematic over time, typically in how a team manages their tasks. Issues with at most two words in their title are assumed to be problematic because only two words will rarely convey sufficient context and motivation for a task. "*Issues with short titles*" counts the number of issues violating this assumption. Similarly, issues that do not have a description likely lack enough context and information about what a task attempts to solve. Such issues can be complex for someone on the team to work on because they lack the necessary information to be autonomous. The presence of description in issues is also expected in IT1901.

In GitLab, labels[26] is the preferred way to manage prioritization, categorization, and progress of issues, also allowing for the use of Workflow boards in GitLab. Therefore, issues will, in practice, always contain at least one label unless they have been forgotten. The indicator "*Issues without labels*" visualizes the number of issues that lack labels.

Lastly, "*Closed issues without any assignees*" look at the number of completed issues that have not been assigned to a team member. Note that issues may have

[26] `https://about.gitlab.com/handbook/marketing/project-management-guidelines/boards/`

been closed because they were redundant, unnecessary, or out of scope, so it is natural not to assign a person to them. Unfortunately, given the varying naming convention and usages of labels, we have no way to discern between a completed or closed issue properly.

A description is added to each panel as information boxes, helping students understand the motivation behind each metric, with examples, shown in Figure 5.6. These information boxes are ideal for short and to-the-point descriptions, which were sufficient for the current use cases. However, the button to show an information box is small and blends into the panel itself, making it hidden from people who do not already know they exist. We expect this to become a problem during the interviews (Chapter 6), even though these boxes are the most suited functionality in Grafana for this use case.

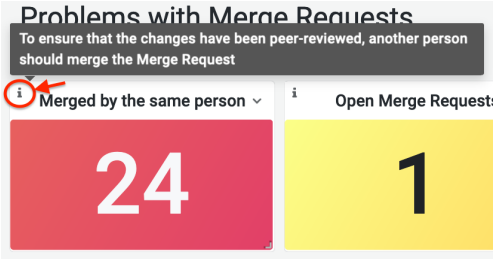


Figure 5.6: Grafana description box
Used to provide additional description to a Grafana panel. Becomes visible when hovering over the `"i"` using their mouse pointer.

5.5.2 Second revision

The second revision was conducted between the first and second rounds of usability testing. Screenshots of the revised prototype are shown in Figures 5.7 and 5.8. Structural changes, removing redundant panels, updating the information of panels, or formulation changes were done in this stage, based on explicit and implicit feedback from Students and TAs. Explicit feedback was directly told to us during the interviews or verbally expressed confusion. Implicit feedback was problems in navigation and interpretation of the prototype, which we observed during the interview. Examples include how much time a participant spends on a specific panel or category.

The feedback was mainly related to improving the general readability and structure of the graphical user interface. For example, multiple respondents were confused about what each color code meant, while others wanted percentages instead of numbers because it better conveys the actual severity. See Chapter 6 for details on the findings from the first and second rounds of interviews.

The revised layout is shown in Figure 5.8. Generally, each category is more

distinctly separated into sections[27]. In order to avoid overwhelming a user with information when they first visit the dashboard, all the sections are minimized by default. This is shown in Figure 5.7. In terms of language, each indicator is no longer named "problem" because it confused the user on whether a value was problematic or not. Percentages are favored as units on indicator panels after responses from interviews. A percentage indicates the actual impact better because it considers the total number of merge requests, issues, or commits.

The dashboard header was refined with a short introductory description and a hint to the `"i"`-button on each panel. Additionally, the color convention is illustrated in similar boxes, so the user clearly sees what each color represents.

"**Merge Requests**" . Indicators that are mainly informational have been moved to the left side. "*Total #*" shows the total number of merge requests in their project, giving users an overview of the impact of the other indicators.

"*Open Merge Requests*" has been colored blue to signify that it is strictly informational. Some students were confused by this indicator during the first round because it used the yellow status color (Figure 5.5), signifying that it could be better.

"*Average time to first comment received*" and "*Average time until merge*" are highly related and have therefore been moved closer to each other.

"*Merge by the same person*", "*Without assigned reviewers*", and "*Not referencing Issues or Milestones*" were converted to use percentages after feedback from interview participants. Additionally, the information boxes have been reformulated with better descriptions and examples.

"**GitLab Issues**". All indicators have been given more space, and the informational panel "*Issues closed per week*" was added. Although no directly actionable feedback is given from this panel, it is meant to give a visual view of how much they progress each week and encourage discussion.

"**Git Commits**" contains more significant structural changes. Commit messages are named "commit message titles" to clearly distinguish them from the commit body. The contribution panels from Figure 5.5 have been adjusted to count the number of commits instead of lines added and grouped with other Git commit panels. To avoid contributions and Git commits having too much focus, the entire row has been moved to the bottom. The gauge panel "*Average length of commit message title*" was added to visualize the average length of commit messages and how far they were from the ideal length of 50 characters (Table 2.2). "*Commits with long commit message titles*" and "*Large Commits*" received only minor adjustments, such as using percentages instead of count. Even though not all users saw

[27]Rows in Grafana `https://grafana.com/docs/grafana/latest/dashboards/dashboard-ui/dashboard-row/`

a direct value of "*Number of identical commit message titles*", some did, during the interview, reflect on the potential problems of having duplicated commit messages. We, therefore, decided to keep the table and decrease the threshold from 3 duplicates to 2 duplicates, such that more rows were shown.

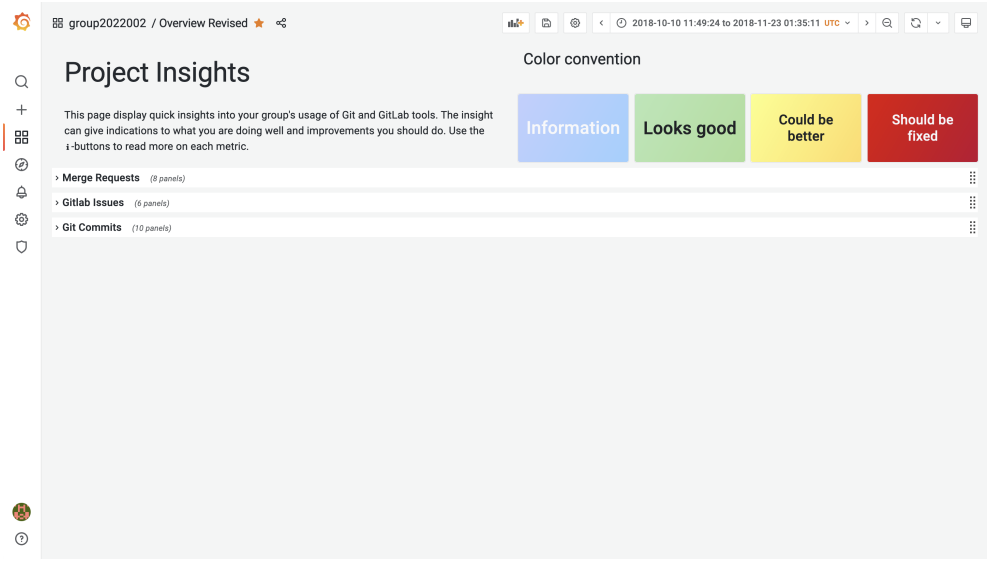


Figure 5.7: Screenshot of the second prototype revision and what a user first sees when visiting the dashboard.

Figure 5.8: Full page screenshot of the prototype from the second revision, where explicit feedback and observations during the user-testing were taken into account. Students participating in round two of the interview (Chapter 6) were presented this prototype.

Chapter 6

Research Results

In total, 11 usability tests were conducted, consisting of 9 with students and with 2 Teaching Assistants (TAs). The testing was organized in two rounds, where 5 students and both TAs were interviewed in the first round. The second round was approximately 2 weeks after the first, with 4 new students. Only the prototype used during the usability task of the interviews was modified between the two rounds. The rest of the interview was identical, following the interview guides (Appendices A and B) discussed in Section 4.4.2.

All the usability testing was conducted in Norwegian, as it was the participants' primary language. Therefore, the quotes presented in this section have been translated from Norwegian to English. In addition, square brackets are used in the quotations to convey context to the quotation and non-verbal expression lost in transcription. I.e., a participant referring to a specific visualization by pointing at it while talking.

This chapter presents and summarizes the findings from the interviews, where Chapter 7 discusses the implications of the findings.

6.1 About the participants

The gender distribution of the participants was about half, with 55%(N=6) females and 45%(N=5) males. 5 of the students were studying *Computer Science*, 3 students *Bachelor in Informatics* and 1 *Natural Science with Teacher Education*. All students took IT1901 (Section 2.3.1) in the fall semester of 2021. 8 of the students were either in their 2nd year or between the 1st and 2nd year of their degree. Both TAs had previously taken IT1901 as students and had the responsibility of following up on an assigned list of student groups. Table 6.1 lists the anonymized identifiers of the participants detailing which round of usability testing they took part in. All of the participants have also taken, or are currently taking, the course Software Engineering TDT4140 (Section 2.3.2), a similar course to IT1901. Additionally, 4 students mentioned that they had taken the course Web Technologies IT2810.

Of the participating students, 5 reported not having used Git before they took IT1901, and 4 said they had used it to some small degree. None of the students considered themselves notably experienced in Git.

Participant Id	Round
Student1 (S1)	1
Student2 (S2)	1
Student3 (S3)	1
Student4 (S4)	1
Student5 (S5)	1
Student6 (S6)	2
Student7 (S7)	2
Student8 (S8)	2
TA1	1
TA2	1

Table 6.1: Anonymized student identifiers, and which round each participated in.

6.2 Usability tests

This section presents the results of the usability tests and the following interview. The results from the first and second round are divided into separate sections, Section 6.2.1 and Section 6.2.2 respectively.

6.2.1 Usability tests first iteration

Table 6.2 shows an aggregated list of findings from the first usability testing round labeled with the respective participants who discussed them.

Location	Discussed topics	Discussed by
Contribution graphs	The *contribution pie charts* are valuable	S1, S2, S3, S4, TA1
	Lines contributed to project might single out individuals	S2, S4, S5
	Lines contributed to project can cause group conflicts	S4, S5
	LoC can be a too simplistic metric	S3, TA1, TA2
	Contribution graphs should not be in focus	TA2
Colored indicator panels in general	The *Indicator panels* in general gave a good overview of improvements	S1, S5,
	The coloring of the *Indicator panels* was intuitive	S1, TA1, TA2
	The coloring of the *Indicator panels* was confusing	S2, S3, S5
	The *Indicator panels* should contain total occurrences	TA1, TA2
	Tried clicking the panels	S2, S3, S4, TA2
Problems with commits	*Problems with commits* is valuable	S1, TA1
	Did not find value in *Duplicate commit titles*	S2, S3, S4, S5, TA1, T2
	Did not see the title of *Duplicate commit titles*	S4, S5
	Long commit titles is not a problem	S2, S5
Problems with Merge Requests	*Problems with Merge Requests* is valuable	S1, S3, S5, TA1, TA2
	Confused by an indicator panel in *Problems with Merge Requests*	S2, S4
Problems with GitLab Issues	*Problems with GitLab Issues* is valuable	S3, S4
	GitLab Issues is not applicable to all groups	TA1
	An Issue description is not necessarily needed	TA2
General notes	Noticed the information buttons	S4, TA1, TA2
	The information buttons should be more eye-catching	ALL PARTICIPANTS

Table 6.2: A summary of all the discussed topics from the first round of usability tests and a mapping to who mentioned them

Feedback on the contribution graphs

The contribution pie charts were generally well-received, with quotes like: "Having this is golden" and "This was nice, easy to understand". The main reasoning behind liking the pie charts was that the graph was easy to understand and gave a good insight into what each member contributed. Student2 liked the contribution pie charts and indicated that it was a better and less revealing way to visualize individual contributions than *Lines contributed to project*.

TA1 found the pie charts and the work distribution between documentation, functional-, and test-code interesting. TA2, on the other hand, did not give much attention to the pie chart metric. TA2 understood what the metric showed but moved quickly to the other metrics.

Student1 was uncertain what the number values in the pie charts were, if it was Lines of Code (LoC), commits, or something else. Student3 suggested adding a fourth pie chart with an overview of the entire code base.

Lines contributed to the project were reported to have several challenges by the respondents. Student2, Student4, and Student5 argued that those graphs could result in some students being singled out if they have not visually contributed to the project, regardless of whether that is true or not. Active use of pair- and mob-programming could result in some students contributing more to the code, even though all have contributed to the exact change.

Student4 said

> "It is almost like a tool that has been made only to get frustrated on your group members "

about the line chart. Student4 and Student5 pointed out that this could result in personal annoyances or conflicts inside the student group because it looks like someone does not contribute enough. However, if the entire group was using the tool actively, Student4 said that the metric could make the group members aware of how much each member works and motivate the least contributing persons to contribute more.

The milestones durations marked as purple rectangular areas in the line chart were emphasized negligibly by the participants. Two students were confused about the milestone durations, but it was caused by the naming of the milestones being different from what they were familiar with. They would probably have understood the metric in a real-world situation where the milestones were based on their GitLab milestones. When they understood the metric, they quickly moved on to the rest of the prototype.

Student3 liked the line chart but suggested labeling the axis and editing the date format from "mm/dd-format" to "dd/mm-format" to make it more readable. Student5 was also confused by the date format.

General feedback on the contribution graphs. Both TAs argues that Lines of Code (LoC) gives a too restricted view on individual contributions, to yield particular value to students. This metric was used in both the pie charts and the line chart. Student3 had a similar concern and gave an example of how an automatic code cleanup tool could result in thousands of edited lines of code. TA1 suggested using *Number of commits* or *Number of issues completed* as metrics instead of LoC but noted that these also had the same weaknesses. TA2 also discussed the possibility of using commits, as this is used by GitLab on their analytics page. However, TA2 noted that it could lead to an unwanted situation where students would spam the repository with small commits to increase their commit count.

Regarding the contribution part of the dashboard, TA2 stated that

> "I do not know to what extent this should be in focus, as there is much focus on pair programming"

and suggested moving it to the bottom of the dashboard. Having it as the first thing you notice on the dashboard might put focus too much on it. TA2 also pointed out that active use of pair- and mob-programming could result in students not getting credited for their work in the dashboard. However, this is taken into account in the dashboard if the students use the GitLab functionality of tagging the other members of the pair programming team with a co-author tag. Student2, on the other side, assumed that the prototype took co-authoring into account. When Student2 was asked how they would use the prototype, they stated

> " [...] if I were pair programming, I would sign it so that I would get my name up "

referring to the co-authoring functionality in Git.

Colored indicator panels

The colored indicator panels are an important visualization on the dashboard used 13 times across *Problems in commits*, *Problems with Merge Requests*, and *Problems with GitLab Issues*. The feedback on these metrics was mixed, were 3 of the interviewed students misunderstood or were uncertain of what the coloring indicated. Student2 misinterpreted the indicator panels thinking every panel was a problem and that new panels would spawn when a new problem arose. Student3 and Student5 both initially understood the coloring of the indicator panels but became uncertain after browsing the dashboard. Student5 said:

> "I do not quite understand the system of it, I kind of thought the green at first meant that it was good, but now I start to think that it was something else."

When asked what caused the uncertainty, Student5 replied that

> "Because you use a green color under *Problems in GitLab*".

From this statement and later discussion, it was found that the confusing element probably was the wording of the category titles. As the titles start with "Problems in," it follows that everything is a problem.

Both TA1 and TA2 pointed out that having just the number of occurrences in the indicator panels was insufficient to get a good overview. They both argued that it was impossible to grasp the extent of a problem without knowing the total number of the metric. TA2 gave an example:

> "I do not know how many [merge requests] they have had. So maybe it could have had a total. How much is 27 [merge requests without reviewers] if they have 120 merge requests...then its not so bad, but if they had 30... they could probably have become better at it".

When asked whether a percentage or total would be most helpful, TA1 said that using percentage could work well and suggested also adding the total.

Multiple participants (Student2, Student3, Student4, and TA2) tried clicking the indicator panels during the usability tests. When asked what they expected to see, they all wanted to see more granular data of the respective chart: i.e., Student2 clicked on *Merged by the same person*, expecting to see which person was in question. Student3 expected to see the open merge requests when clicking on the *Open merge request*-panel.

TA2 remarked that the thresholds where the colors of the indicator panels changed could be revised. That it sometimes should have been colored green and not red. For example, in the indicator panels regarding issues, 3 issues without description were harsh if there were a decent number of issues. TA2 also suggested making the color thresholds more apparent to the user.

Problems in commits was the category most liked by the participants. The main problem was the duplicated commit titles, which only Student1 fully understood, stating:

> "It is pretty lovely to have an overview of. It is not a thing you think so much about, I think, that you should write a little better commit titles ".

Student4 and Student5 did not see the chart's title, assumed it was a list of the most recent commits, and disregarded it. Others read the title, understood that it was a list of duplicate commit titles, but did not know how or whether to act upon that information. Student2 and Student5 were unsure why or disagreed that long commit titles were a bad practice.

Problems with Merge Requests was well accepted by the participants. However, student2 and Student4 were both uncertain of what *Without Reviewers* meant, and Student4 confused it with *Merged by the same person*. This may indicate that the wording of the panel titles should be more precise. The information buttons helped the users understand the indicator panels but could also be improved.

Problems with GitLab Issues were not as emphasized by the participants as the other categories. All participants understood the panels, but only Student3 and Student4 explicitly mentioned that they were of any value. Student3 noted that:

> "This [*Problems with GitLab Issues*] can be a good tool for the group because then you at least see 'It is 2 Issues where there is a problem' and you can fix that instead of it getting messy in GitLab".

TA1 noted that of the groups under their guidance, one group was using Trello instead of GitLab Issues. So for them, the GitLab Issues part of the dashboard would be empty. TA1 suggested making the visibility of the different dashboard parts togglable. TA2 argued that a GitLab description was not necessarily needed if the Issue's title was good enough.

General feedback on the prototype

The information buttons in the top left of the charts give further information to the users on what the chart displays and how it is calculated. Both TAs noticed the information buttons. Among the students, however, only Student4 noticed the buttons. When the students who did not notice the information buttons were made aware, they all responded that they were either too small or not eye-catching. Even the ones who discovered the buttons suggested making them more visible. The participants who found the buttons and used them when exploring them all found them valuable. Student4 went from not understanding the duplicate commit titles chart to understanding and liking it after reading the information. Some students wished to add more information to the buttons and for more dashboard metrics to have an information button. TA2 argued that the information was valuable but could be more targeted towards guiding the users on how to resolve the problems, thus making the tool more actionable.

6.2.2 Usability tests second iteration

The usability tests of the second iteration were done with a revised version of the prototype. However, some of the metrics on the prototype remained unchanged. The changes are described and justified in Section 5.5.2.

The topics discussed during the usability testing in the second round are summarized in Table 6.3 and elaborated further in this section.

Location	Discussed topics	Discussed by
Contribution graphs	The *contribution pie charts* are valuable	S7, S9
	Commits contributed to project is valuable	S6
	Commits contributed to project might single out individuals	
	Commits contributed to project can cause group conflicts	
	LoC can be a misleading metric	S9
Colored indicator panels in general	The *Indicator panels* in general gave a good overview of improvements	S6, S7, S9
	The coloring of the *Indicator panels* was intuitive	S6, S7, S9
	Tried clicking the panels	S7
Git Commits section	*Git Commit section* is valuable	S6, S7
	Found value in *Duplicate commit titles*	S6, S7, S8
	Did not find value in *Duplicate commit titles*	S9
	Did not understand the *Average commit title length* chart	S8
Merge Requests section	*Problems with Merge Requests* is valuable	S6, S7, S8
GitLab Issues section	*Issues section* is valuable	S7
	GitLab Issues is not applicable to all groups	
	An Issue description is not necessarily needed	
General notes	Noticed the information buttons	S6, S7
	The information buttons should be more eye-catching	S6, S9
	Read the introductory information text	S7, S9

Table 6.3: A summary of all the discussed topics from the second round of usability tests and a mapping to who mentioned them

Git commits category

Commits contributed to the project received much less attention in the second round than in the first. Student6 and Student9 were the only two who made any remarks regarding the chart. Student6 found it valuable and suggested using the metric to understand the group's workflow and helping the person with the least commits contributed. Student9 also liked the line chart:

> "Here we get to see how fast your development speed is. So that is handy".

The contribution pie charts were in general well received in interview round two, but with some confusion as to whether it was based on Lines of Code (LoC) or *Number of commits*. Student8 argued that the contribution pie charts are very relevant and valuable for IT1901:

> "But it is actually very relevant to the course, and it shows who has done the most, and then you can think about whether it makes sense. It often does. You know who has done the most. However, it makes you talk about it ".

Later on, Student8 seemed confused as to whether the *Commits contributed to project* and pie charts were based on the number of commits or LoC. Student9 was also uncertain of what the pie charts denoted when discussing that LoC is a limited metric:

> "The only problem is that there is a lot of code. So it depends on how this[the pie chart metric] is made. But I feel that code is... you can spend a long time on very little code... and then you can spend a short time on a lot of code ".

Student9 also gave an example that the use of a Linter could bring substantial changes to a repository when evaluated by the number of lines.

Duplicate commit titles were better understood by the participants of the second round. Out of the 4 participants in the second round, 3 understood the table and found it valuable. Student6 praised both *Duplicate commit titles* and *Commits contributed to project*:

> "You can get a lot of use for this one here[commits contributed to project] eventually when everyone is more comfortable. And at least the one with identical commit lines. Because then you can see how you can improve it. Because these [panels] come automatically when you have done something ".

Student8 was able to give a thorough analysis of problems in the group project solely based on *Duplicate commit titles*. An example of the insight the student found was:

> "Okay, this is a little weird. That it is a branch that has been merged 3 times into master. It seems to have been merged into the master before it is finished. And that is a little weird, so I would have considered it in the next sprint: not to merge into the master until you are completely done with an issue."

Student9 did not see any value in *Duplicate commit titles*, arguing that it was reasonable to use duplicate commit titles and that using the default commit messages when merging branches is justifiable. Student7 mentioned that not everyone might know what a *Commit message title* is.

Average length of commit titles. Student8 did not understand the *Average length of commit titles*. The student thought the number in the gauge panel was the total number of long commit titles, while it is the average length. Further, Student8 argued that a short or long commit message title depends on the content of the text and not its length. Finally, the student said:

> "I would not say that I can say anything about whether it is good or bad based on this ".

Merge Requests & Issues categories

The Issues section received, in similarity to the first round, little attention. However, it seemed that the metrics were found understandable but maybe not so actionable. Student8 remarked:

> "I thought this was very good, because at least for that course[IT1901] because there it was so important that you assigned yourself[to GitLab Issues] and such. So it would have helped to see that you actually are doing that ".

Student9 was unsure whether the metrics in the Issues section were based on closed or open issues.

The Merge request section was understood by all the participants. After remarking that the issues section was very relevant for IT1901, Student9 continued talking about the Merge Request section:

> " [...] And same with merge request. You can see that it is not the same person[creating and merging merge request] and is not reviewed... Because people have probably not thought about it. I do not think we did either. I never think we assigned reviewers[to merge requests]. So it would be nice to be a little aware of that."

Student9 argued that having the same person merge and create merge requests is not necessarily a bad thing:

> "When you who make a merge request, then you know best what your code does. So the others can go in and look over, but if there is a merge conflict, you often should resolve it because you know best what should be where. And the others do not necessarily know, so it is not necessarily stupid that it is the same person who merges.

General feedback on the revised prototype

The introduction text was read by 2 of the 4 students in the second round. Student6 noticed the introduction text immediately when opening the prototype but did not bother reading it. Student8 did neither notice nor read the text. The text informs the reader about the info buttons, but of the two students reading it, only Student7 found them without assistance.

The information buttons were found without help by Student6 and Student7. Between the two interviews, an effort to make the info buttons more aimed towards guiding the students on how to fix the respective problems. Student7 commented:

> "On the info button, it says and how we can improve. Which is very good ".

Student9 did not find the information buttons without help but found them helpful when made aware of them. Student8 also had to be made aware of the info buttons. When made aware, Student8 argued that the buttons' principal value was comprehending how the metrics were calculated. Student9 and Student6 both suggested making the buttons more eye-catching.

The color coding of the prototype was commented by Student9, who liked it but was also uncertain about the thresholds:

> "It is very nice with the colors because they are the ones that indicate what is good and bad on the page. But you can, of course, disagree a bit with the colors... It depends a bit what you are assessed on, I guess."

Getting more focused on graphs rather than code was argued by 3 students to be a potential adverse side effect of the tool. Student1 and Student2 discussed it during the first round of usability tests, and Student7 also brought it up during the second round. Student1 said the following when asked whether there were any problems with such a tool:

> "That people stress a lot about getting all the squares to be green, and then it might affect the group's efficiency".

Student2 mentioned a similar problem, but was more unsure to whether it was a problem or not:

> "Maybe you get too hung up on the graphs and not the actual code... No, that can not be a problem.. Or I do not know... Maybe!"

Student7 noted that the tool could be used in an unintended way, just to improve the graphs:

> "I think I could have abused it and just [commited] lots of commits just to bump it[the metrics in the prototype] up. And have an internal competition, but that might also have increased the workload in the course because you actually get to see how much you do ".

Having to learn a new system was argued by Student6 as a potential problem with such a tool. When asked whether there were any downsides to such a tool, Student6 answered:

> "That it will be another tool to get acquainted with for those who are not very comfortable with new tools. But it is also just statistics about the code base, so I think I would use it personally." .

TA2 similarly argued that the tool's value would depend on its implementation. Whether the students needed to download the application to their computer or if there was a struggle setting it up:

> "One thing I think is how to use this tool. Will it be a browser extension?... do you have to download it?.... And are the students obliged to use it? Or is it optional? I think these things have a bit to say about whether people will bother to use something extra ".

When being told that with the current and intended architecture, the student only needs to log on to a web page without any required setup, TA2 answered:

> "Then I think there would have been effortless to do so[login to use the tool]."

6.3 The perceived value of the tool

This section presents the results from both rounds of usability testing regarding the tool's value to the students and the Teaching Assistants (TAs). The students' and TAs' expressions regarding positive and problematic values have been clustered into 8 positive and 4 problematics. The positives are shown in Table 6.4, and problematic values in Table 7.1 labeled with the students and TAs who discussed it.

Proposed positive value	**Mentioned by**
Gives a structured overview of the project	S1, S2, S3, S5, S8, S9, TA2
Gives concrete proposals for improvements	S1, S4, S5, S6, S9, TA1, TA2
Helps students learn recommended practices in Git and GitLab	S2, S4, S7, S8, TA2
Helps project groups to work agile	S3, S4, S6, S7
Encourages discussions around work culture	S4, S7, S8
Motivates to follow recommended practices	S2, S7
Makes TAs guidance preparations more effective	TA1, TA2
Gives TAs better insight to groups' work culture	TA1

Table 6.4: The positive values discussed by students and TAs during both rounds of usability testing

Getting a structured overview of the project was frequently mentioned as a positive value of the tool. The participants expressed that getting an overview of how well one and the rest of the group work and how structured the group works is a valuable trait of the tool. Student2 argued that strictly following an agile methodology could be problematic for him/her as an unstructured person, but that the tool may make it easier to work agile:

> "I do not know if I am structured enough to be able to work like that [following a process methodology] because I have to concentrate on the code. Such overall mess becomes a bit... Or maybe it is nice with such a tool to make it a little easier" .

Receiving concrete proposals for improvements was another positive value that many participants highlighted. Student1 argued:

> I think it is beneficial to have such a tool to see what you should do better. Because many who do not have that much experience with Git, and such, do not quite know what to do.

Another trait mentioned by Student4 is that having a concrete list of things to improve is valuable at the team meetings.

> *Student6: "But I do not think it[the tool] will be used very often, to begin with, because there are so many other things to familiarize with."*
> *Interviewer: "You think it would have been used more eventually?"*
> *Student6:"Yes, when you get more experience in Git and GitLab*".

Helps students learn recommended practices in Git and GitLab was valued by 5 participants. They argued that Git and GitLab practices can be hard to grasp and that the tool might be beneficial for students to learn the different practices. Student2 said:

> "It took a long time before I became proficient in Git, so maybe this[the prototype] had helped me to see what is important, and what was the best practices".

Similarly, Student8 said:

> "It [the tool] might have made it easier in the beginning. Because in the first sprint, there were many of these things [practices] we did not do because we were unaware we should. But we found out in the second sprint because we were instructed, 'Now you actually have to do it.' So if we had this[the tool] from the start, maybe we would have understood what you should do earlier."

Student4 added that the tool could be used to guide the ones who had fallen behind:

> "In my experience, GitLab and its functionality can be confusing for many in the beginning, so then [with the tool] you might see who is not fully acquainted with it [GitLab and its functionality] and may need guidance. There are many benefits".

Helps project groups work agile was a suggested value by 3 students. Student3 stated:

> "The value would possibly be to help, considering IT1901 at least, then it will help the group and go a little more into that role-playing game of being agile and scrum and such because eventually it could become a bit messy and you lose a little focus on being structured with commits and reviews.

Student6 argued that it would help groups during the retrospective of a sprint:

> "It could have been beneficial, at least in retrospectives looking back at what could be improved to the next release," and
> "I think at least that would have made it easier for us to keep short retrospectives" .

Student7 stated that the tool was valuable for the groups' collaboration:

> "So there is an excellent basis for improving the collaboration. Through learning the conventions of Git, and collaboration in general "

Encouraging discussions around work culture was brought up by 3 students. Student7 said:

> "At least I had appreciated this during IT1901 so that you might get a little more openness about how the group works together and what you need to become better at. Then it becomes a little easier to put your finger on it, whatever the problem is".

Student4 mentioned that the tool encourages a group to discuss their work culture and align. Student7 said the following when asked what value the tool could give students:

> "I think it will be easier to communicate the working methods because you get something concrete to refer to."

Motivates following recommended practices was a value mentioned by two students. Student2 argues that the tool could motivate to follow the practices it visualizes:

> "It took a long time before I became experienced in Git, so maybe this had helped me to see what is important or what was good practice. And perhaps motivated to be more careful about using milestones and issues so that it would look nice here" .

However, the intention to follow the practices seems to be more driven by making the dashboard look nice. A similar pattern is found in a quote that already has been referred to in Section 6.2.2 by Student7:

> "I think I could have abused it and just [commited] lots of commits just to bump it[the metrics in the tool] up. And have an internal competition, but that might also have increased the workload in the course because you actually get to see how much you do ".

The tool's value for counseling was discussed by both TAs. Both TA1 and TA2 stated that the preparations for the group guidance sessions were tedious, with a random sampling of branches, commits, and issues at each GitLab repository looking for very similar things to the prototype. TA1 would rather use the tool and avoid the random sampling:

> "And here you have all of the data gathered, so you do not have to take random samples and actually see if they have done so. So I had definitely used this as opposed to what I did ".

TA2 suggests that the tool would be brought up in every group guidance session:

> I would probably bring it up at every meeting. Suppose I were to have a meeting with them and had to go through how they are doing. It is perhaps not so appropriate in the middle of [a sprint], but right after a deadline, for example. Take a snapshot or something.

6.3.1 How students would use the tool

When asked how often they think they would have used the tool if they had it during IT1901, all students responded that they would use it at least in-between sprints in the retrospective or sprint planning. Two students said they would use it weekly during team meetings. Student7 stated that the tool would be used daily during the most intense phases, indicating that its usage could vary through the project:

> "If I had worked like we did last year, where we almost entirely did all the work the last 2 weeks before submission, then I would not have used the tool before those 2 weeks. But while working actively with the subject, I think I would have used it daily. Almost like a social medium. Where you go in to look for updates".

Some students also reckoned they would frequently use the tool before deliveries to make necessary adjustments to their project. Student1 thought it would be used frequently by students, especially towards deadlines. Student2 argued that the tool would be more valuable to use along with the project rather than right before the deadlines, saying:

> "If we had not used it along with the project, I do not think I would have used it before the deadlines either. Cause it would have been like 'OH NO, we have bad practices, I do not want to know about that now'".

Student2 also argued that the development process, being part of the grading, was a prerequisite for the tool to be used:

> "[...] and that I knew it in advance, that this was something which was assessed[during grading]."

Student4 explained that it is essential to use the tool together in the groups:

> "If you use it[the tool] only for yourself when you sit at home and think, 'Holy f***, what a group I have', then you just get frustrated at it. Then we only make things worse. So you have to use it together with the group."

Chapter 7

Discussion

In this chapter, we discuss how the findings from Chapter 6, in conjunction with existing literature, demonstrates the value of a Git and GitLab Visualization tool for Software Engineering students (*RQ1*), and limitations future uses should consider. Lastly, the benefits and limitations of using third-party visualization software in this context are discussed (*RQ2*) based on student feedback and limitations in the underlying software.

7.1 RQ1 - Students value of the visualizations

The value of the prototype is documented through the 11 interviews summarized in Section 6.3 and Table 6.4. In general, the students found the prototype valuable, with all students reckoning using it at least once during each sprint if they had it available during IT1901. Some students even suggested they would use it every week during every team meeting. However, some participants reported negative side effects, which should be considered. This section discusses the different findings regarding students' perceived value of the developed prototype and what should be considered when creating such tools to maximize the students' value.

1) Students value rapid feedback to changes made on their project. Getting feedback with concrete proposals on how to improve their development processes was valued by the participants. The students expected the feedback to be continuous, instantly updating after a repository change was made. Some students were also suggesting using the tool daily after making changes in GitLab, validating that their actions were done satisfactorily (Section 6.3.1). Gustavsson and Brohede [40] argued that weekly data updates were sufficient for their use, and Gary and Xavier [41] chose a daily approach. A weekly approach may seem sufficient based on the suggested use frequency from students (Section 6.3.1). Some metrics are possibly unsuited for continuous feedback and may need moderation from course staff before it becomes available for students. However, in most situations, updates should be near real-time to support continuous feedback where students

can make changes and use the tool for validation.

2) Students value having a visual overview of the project. This was together with *concrete proposals for improvement* (Table 6.4) the most frequently mentioned positive value during the usability testing. Part of it was related to getting an overview of how one worked compared to the rest of the group. Such an overview was argued to encourage group discussions and reflections (Table 6.4). However, there were also concerns about the metrics singling out individuals, potentially causing conflicts.

Eventually, how the groups approach the tool and insights seems crucial to its perceived value. It can be envisaged that some pedagogical activities are built on top of the insights to facilitate learning. If a group uses the insight to form discussions, it might prove easier to act upon the insight and yielding value to the group. Acting upon the insight is much harder if used individually, and the tool might be more counter-productive than valuable. This is discussed and illustrated by Student6 in Section 6.3.1.

3) Help students learn Git and GitLab practices. This was mentioned to be of value to 5 of the participants. They argued that learning Git and GitLab in their experience was difficult and that the prototype could help them in their learning process. Some argued that some of the recommended practices were unknown to many, particularly those unfamiliar with Git and that the tool could make it more accessible.

It was also mentioned that the tool could help highlight who was struggling with the concepts, which would make it easier to point out who needed additional guidance. A similar trait was found by Haugse and Aalberg [1], where it was pointed out that their implementation could help detect problems earlier and make the course staff able to act on the problems before it was too late.

4) Using the insights to improve student counsel for TAs. The TAs that partook in the usability tests mainly worked on helping students at the process and team level, helping them in case of conflicts, confusion, or answering questions. Giving the TAs a deeper insight into the groups' projects and work culture makes giving targeted feedback easier. The participating TAs were very positive towards applying the prototype in guidance sessions. They both argued that the tool would make their preparations more effective and, to some degree, more precise. TA1's preparations before meetings with students involved random sampling of the Git repositories. This tool mitigated that randomness, making it more precise. However, with only two participants, further study is required to understand its value to TAs and whether that is reflected in students' perceived value.

5) Duplicate commit titles have potential value in future work. During the interview with Student8, much time was spent analyzing the table of *Duplicate*

commit message titles. As a result, based on the table alone, Student8 was able to discover and reflect on poor Git commit (Tables 2.1 and 2.2) and branching practices. This was a surprising finding, as it was an even deeper insight than what the *duplicate commit message* metric was intended to provide.

Git commit messages are, in general, likely data that can be used to give feedback on practices. However, as the commit messages are in a natural language, extracting insight from this data is difficult. However, a simple aggregation of duplicates seems to provide a way of extracting some valuable insight from the commit messages. Unfortunately, from the interview rounds, the *duplicate commit message titles* was proven challenging to understand, with a minority of the participants understanding the table visualization. Therefore, finding better ways of presenting the data is necessary to make the data source more valuable to students.

7.1.1 Considerations affecting the value

Several concerns were discussed by the participants of the interviews in Chapter 6 which need consideration as they may affect the students' value of the tool. These concerns and who mentioned them are summarized in Table 7.1.

Concerns	**Discussed by**
Presents an inaccurate measure of work load	S3, S9, TA1, TA2
Getting more focused on graphs than code	S1, S2, S7
Group members can easily get singled out	S2, S4, S5
Cause conflicts/bad group atmosphere	S2, S5, TA1

Table 7.1: The concerns discussed by students and TAs during both rounds of usability testing

***Lines of Code (LoC)* and *Number of commits* do not tell the whole story** and are both argued by the participants to be too simplistic. However, there are no perfect way of measuring the value of contributions [29] as discussed in Section 3.2.2. LoC was used on the first version of the prototype and was discussed by both TAs and Student3. They were concerned with whether the metric could give a good insight into contributions. Student3 gave an example that a quick code cleanup could result in a disproportionate number of lines added. It was suggested by the TAs that *Number of commits* and *Number of completed Issues* could be used for contributions, but also pointed out that it had a similar problem as LoC. The concern *Presents an inaccurate measure of work load* in Table 7.1 is grounded in the contribution metrics being too simplistic. These results build on existing evidence that LoC and *Number of commits* can be too simplistic or even misleading [36, 39, 43].

In the second revision of the prototype, *Number of commits* was chosen as the metric for both the line chart and pie charts. Interestingly, in the second interview

round, Student9 raised a similar concern as the TAs, and Student3 did in the first interview round, with LoC being too simplistic. However, in this interview round, LoC was not used anywhere. Student9 assumed LoC to be the metric behind the contribution charts, which might indicate how indoctrinated LoC is as a metric of individual code contributions.

LoC and *Number of commits* have the problem that they are all too simplistic to capture the complete task of quantifying individual contributions fully. Firstly, by using LoC as a measure, students are measured in how many lines they write. Secondly, writing many lines is by itself not a goal. It might even be counter-productive as code quality is preferred over quantity [53]. Moreover, the difficulty and effort put into a contribution are not reflected in the lines added. Similar effects are found in *Number of commits*. Having smaller commits is generally considered a good practice [14, 15, 17–19]. However, committing excessively to get better feedback on the tool can become a problem. TA2 discussed this concern in Section 6.2.1.

As an alternative, combining the metrics might be a better approach. By having LoC and *Number of commits* work together, they give a broader insight and can cancel out each other's unwanted effects. For example, if a student were to commit big commits rarely, it will be reflected in a high LoC and low *Number of commits*. Similarly, committing smaller commits often is reflected in a higher commit metric than LoC. Additionally, taking time and frequency into consideration may be helpful to understand which students are consistent and distributes work over time, such as *Time Spent on a Project per day* (Table 3.2). As a result, harmonizing contribution metrics seems a more insightful way to visualize the health and steadiness of contributions, which should be further researched.

The feedback visualized can alter the behavior of students. Although not a new finding [35], it is still important to be aware of the implications of visualizing feedback to students. Forsgren *et al.* [20, p. 45] discuss the importance of correct measurements in DevOps, and flaws of measuring the wrong outputs. Student7 suggested that the visualization tool would be "abused" to get better-looking statistics for their group, confirming these assumptions.

Looking at an example from the usability tests. In the first round, we observed multiple students spending more time on the Contribution graphs and arguing that these presented too simplistic a view of contributions (Table 6.2). Before the second round of usability tests, contributions were moved to the bottom of the dashboard and grouped closer to Git commits. This resulted in students focusing much less on the contribution graphs in isolation, resulting in much less discussion Table 6.3. Instead, it may seem that the students considered contributions part of the total overview.

When planning which metrics to visualize and how to visualize them, it is important to be aware of such constraints and consider how students may abuse the metrics. Significantly, if the visualization tool shows students that their project has little that requires improvement, it should not contradict the course goals.

Otherwise, students risk focusing on improving metrics that may negatively affect their grades.

Students expected pair-programming to be reflected in the contributions. Some participants, such as TA2, were concerned with whether a co-author's contribution was reflected in the visualization or not. Similarly, findings from Haugse and Aalberg [1] noted similar concerns particularly in courses where pair programming are encouraged. This possibly results in students being discouraged from doing pair programming. The visualization tool in this thesis takes into account co-authoring during contribution classification (Section 5.3.3), assuming the students have utilized the co-authoring functionality in Git by simply attributing the same commit to multiple *authors*. However, from feedback received during the usability tests (Section 6.2.1), it seemed that the students were not sure whether these were accounted for or not, indicating that how contributions are summarized were not properly described. Future improvements to the visualization tool should better inform the students that co-authors are attributed and possibly describe how to assign co-authors in Git.

It is important to find the right degree of individualized performance metrics. The visualization tool gives many insights into a group's development process and, to some degree, measure group individuals' performance. With the data accessible, it is possible to make the prototype more focused on individual performance and less. Displaying insight into how each group member contributes is, on one side, intuitive, as a group's effort is the sum of its member's contributions. Seeing how the different group members work can form the basis of fruitful discussions regarding work culture and team dynamics, as pointed out by Student7 in Section 6.3. On the other side, individualized performance metrics were questioned by participants to make some individuals feel singled out and potentially cause conflicts within the group.

The prototype implementation was consciously done with a careful application of individualization. Despite this, 4 participants were concerned with individualization making group members feel singled out or that it could cause conflicts. Then again, the insight was found valuable by others. Interestingly, all the participants who discussed this concern were part of the first round of usability tests.

The contribution charts were moved to the bottom of the page between the rounds, as suggested by TA2. Consequently, after the revisions, no single participant raised any concerns about the matter. From this, it seems that moving the contribution charts out of the initial focus when loading the prototype made the users put less emphasis on it. However, this is only indicative as there were only 4 participants in the second round.

The course requirements have to be reflected in the tools' implementation. The course requirements in Software Engineering courses can vary greatly, depending on the objective of the course and what practices the course teaches. As

explained in Section 2.1.1, the strictness of different requirements varies with the different practices. In order to use the prototype in a specific course, it should be tailored to match the specific requirements of the particular course. This is important to prevent the tool from motivating practices that conflict with those taught and graded in the course. Additionally, it should be put down an effort to implement any requirements that the course has, which is not reflected in the tool to give the students a valuable overview of their project concerning the requirements. As discussed in Section 6.3, having a good overview of the project and how to improve the project regarding the course requirements were recognized as valuable for the interviewed students. If a mismatch between the prototype and course requirements were to occur, it would probably lead to confusion and significantly reduce the tool's value.

7.1.2 Summary

After discussing the value of providing students with continuous feedback, we should question where this tool should be positioned relative to other tools that provide continuous feedback. In Software Engineering, receiving continuous feedback on changes has through empirical research on DevOps shown to be essential [20, 25]. When comparing our visualization tool against typical Continuous Integration (CI) tools, such as Automated testing and code quality Linters, it should not attempt to solve the same problems. Automated testing does a much better job at validating that functionality works as intended and that the correct functionality is implemented and enforced as part of the project's CI pipeline. Similarly, code quality tools such as Linters and more advanced code quality software can enforce styles as part of the CI pipeline or before any changes are committed.

The visualization tool proposed in this thesis may utilize data from such systems to enrich the insight and feedback given to students. We argue that the ideal role of this visualization tool is to provide students with insights into the overall project, looking at the bigger picture and customized for the learning goals of each course. It can more easily aggregate data over a longer duration and join different but related information. However, the visualization tool cannot mark a pipeline as failed without considerable work put into customizing the system to different stakeholders' needs and existing systems. Nor are all problems visualized (Figure 5.8) bound to a Git commit or push. Therefore, it is not ideally suited to enforce policies when students can ignore the problem or postpone it to a later moment when the work has multiplied.

Most of the metrics in Table 3.2, and recommended Git practices Table 2.1 & Table 2.2 cannot easily be measured or verified as part of a CI & CD pipeline, because they require historical data, data unrelated to code changes, or it is generally too difficult to place an enforceable number on it. In such situations an analysis and visualization tool are more suitable.

We have seen that most literature has built tools targeted toward course staff (Chapter 3). Although, feedback from students and Teaching Assistants (TAs) in

Software Engineering courses indicate that there is value in presenting students with a visualization tool, focused on presenting them with continuous feedback on Git and GitLab metrics, that existing tools do not solve already. What specific metrics to present is likely dependent on the learning goals of each course and its difficulty. Although, the metrics used in the prototype tested can be used as a baseline, together with alternatives from Tables 2.1, 2.2 and 3.2. However, more data is required to draw general conclusions, and the research would benefit from a field study to test the tools in an actual course to gain knowledge of how it is valued.

7.2 RQ2 - Using third-party visualization software

This section discusses the benefits and challenges of using the third-party software Grafana to visualize Git and GitLab metrics (*RQ2*). It reflects on whether the other third-party software would have fewer limitations after two rounds of Usability tests or when a custom-built solution would be more suitable. Lastly, we discuss where the third-party software is most suitable in this educational context.

7.2.1 Grafana as Visualization software for educational uses

Grafana was the third-party visualization software selected for further prototyping after considering the alternatives Section 5.4.1. This section summarizes the experiences of using Grafana in this context, its benefits, and its limitations.

Benefits of using Grafana

The benefits of Grafana stated in Section 5.4.1 make it easy to connect multiple data sources and start to build Dashboards. In addition, during the building of the prototype, a tool that amplified the feedback loop allowed for more experimentation with different visualization styles and overall structure.

Several students noted that the criteria for using such a visualization tool are that it should be easily accessible. This condition was expected before the interviews were conducted (Table 5.1). It can be argued that user-friendly authentication mechanisms, such as SSO through their university account, are criteria for an easily accessible solution. A solution that requires many unnecessary steps to access increases the threshold to visit it regularly. Therefore, reducing the usability of the visualization tool. Given that Grafana can be visited from the browser and supports SSO, it should be considered easily accessible by these criteria.

Limitations of using Grafana

After testing Grafana during prototyping and the Usability tests, several limitations have been discovered that should be considered. These limitations are mainly oriented around the Graphical User Interface (GUI), affecting how students understand the visual presentations.

Users expect to click on a panel to see details. Several of the respondents did during the interviews attempted to click on a panel because they wanted to see which Git commit, GitLab Issue, or Merge Request was violating an indicator panel. Unfortunately, Grafana has limited support for such functionality, requiring workarounds and likely resulting in less user-friendly overviews.

The information box is too hidden. Even though the second revision added a hint in the introduction text for the `"i"`-button, it had minor improvements to the click rate. Unfortunately, the amount of effort necessary to modify the visibility of information boxes was not feasible in the constraints of this project. An alternative would be to use dedicated panels of text. However, text panels had similar restrictions, which would disrupt the user interface or be challenging to read. It is worth noting that after users have seen the `"i"`-button, they start looking for it during the user tests. However, the barrier of entry is too significant, and some students did not find this information until we later pointed it out.

The panel and row names are too small. Panels and rows in Grafana are mainly designed to show key metrics on a more specified measurement while having room for many panels in a dashboard. However, multiple students were observed being confused by a panel because they did not understand what it was until they saw the panel title or it was pointed out.

Customizability of the visualization options in Grafana is limited. Grafana has a wide variety of built-in visualization options, but they are all, to some degree, limited. Therefore, during the development phase, visualizations had to be made around what Grafana supported and not the other way around.

The dashboards had to be duplicated for each student group. To ensure that a student group only sees data for their group, Grafana required each of these groups to have a copy of the same dashboard. Without underlying changes to Grafana, it is too resource-intensive to set a parameterized value based on which student group (represented as Grafana Teams) the authenticated student belongs to. Given that Grafana Dashboards are represented as JavaScript Object Notation (JSON), duplicating dashboards and injecting custom parameters were straightforward. However, it still requires a custom script to run, which has pre-existing knowledge of which student groups exist. This added complexity should be considered when considering the total cost of using third-party software. Similarly, course staff must visit multiple dashboards to compare each group, which can take considerable time in large courses.

7.2.2 Comparing the limitations of Grafana with the other third-party software

Comparing the limitations of Grafana discovered during usability testing with the other third-party software (Section 5.4.1), it is likely that some tools will not have the same limitations. For example, PowerBI, Tableau, and Apache SuperSet are designed as more advanced business intelligence tools and give more freedom in visualizing dashboards, text, and graphs. These have, therefore, more alternatives for how it can present information and likely avoid the problems of *hidden information boxes*, and *too small panel names*. However, all tools are limited in how to *manage multiple dashboards* and what each group has access to, which is particularly problematic in courses with many groups.

7.2.3 Considering custom-built visualization software

Considering the limitations of Grafana and the other third-party software, a custom-built visualization tool for students may be a better fit. At the expense of higher development costs, students get a tailor-designed visualization purpose-built for feedback and learning. Although, with modern front-end technology, libraries and framework creating a visualization tool do not guarantee higher development costs when compared to third-party solutions. JavaScript (JS) libraries such as Highcharts[1], Chart.js[2], and others simplifies the complexity of interactively visualizing data on a custom front-end. JS frameworks such as Vue.js[3], ReactJS[4], and others significantly reduce the amount of work required to build and maintain interactive websites. The developers fully control how students should interact with the visualization tool. To customize dashboards for each student group in third-party systems requires a custom script to generate every dashboard beforehand with knowledge about each group.

7.2.4 Summary

Grafana shows several promising aspects in its use during a Software Engineering course. Notably, it provides users with great flexibility in ways to visualize information, creates dashboards and graphs on-demand, parameterized dashboards, connects to multiple external sources, and uses Single Sign-On (SSO) through University accounts. Such features are precious for Course staff who want to drill down on specific per-project metrics or compare projects, as demonstrated by how TA1 would use the visualization tool (Section 6.3). Furthermore, the possibility to self-host, being open-source and basic Role-based access control (RBAC), makes it easier to handle privacy concerns compared to the commercial solutions, PowerBI and Tableau.

[1] `https://www.highcharts.com/`
[2] `https://www.chartjs.org/`
[3] `https://vuejs.org/`
[4] `https://reactjs.org/`

However, for the target of this thesis, the student groups, Grafana does not seem to be the best tool for the job. Students expect more details and guidance for each indicator, helping them understand what to correct or why it is essential. In such situations, a custom-built solution will likely be worth the effort, even though it may require more time initially to design and build the system.

Concluding, the limitations of Grafana and the other third-party visualization software (Section 5.4.1) indicate that these may not be the best tool to use for the guidance of students and to provide them with live feedback. In such cases, teaching students how to interact with such tools and where to find more information would require time. However, the usability limitations are likely not equally problematic for TAs, making the benefits of Grafana and similar third-party software more noticeable.

7.3 Research limitations

This section presents and discusses potential limitations to the results of this study.

All participants have already learned about collaborative programming. A limiting factor to the conducted interviews is that all the participants had completed the Software Engineering course IT1901, where they learned about the use of Git and GitLab. During the usability tests, the participants could think back to how the tool would have been for them as unfamiliar with Git and GitLab, but only to a certain degree. Already familiar with GitLab functionality and -jargon might influence the interviewee's understanding of the prototype. On the other hand, people without programming experience in teams might have a tough time grasping and judging the utility of such tools and visualizations. In conclusion, testing the tool's usability is preferably performed on students without collaborative programming experience, while validating the tool's value is favorably done after the participants have been through a similar course.

The usability tests are not tested in a natural environment. Ideally, a field study of the prototype during the whole course track would present more realistic data because it would enable the project to observe students using the prototype in a natural environment as they gradually learn the course material. However, the course IT1901 is only taught in the fall semester and was, therefore, unavailable for this project. Alternatively, TDT4140 is taught in the spring semester, which could have been possible to test. However, the time required to plan and build the prototype and receive data processing approval from NSD would limit the time available to run a good field study. Therefore, it was decided to instead focus on running Usability tests, with pre-and post-interviews, with students that had recently taken IT1901. The fictional case was constructed to resemble a project in relevant courses. This approach required fewer resources to conduct these tests while gaining knowledge on what feedback and indicators students appreciated,

including discovering any usability problems. Future work can build upon this data to plan and run the field study.

A significant difficulty for the thesis was the recruitment of participants. A cause was likely the overlap between usability tests, due dates, and exams, making more students hesitant to dedicate time to participate. In addition, the processing time to receive approval from Norwegian Centre for Research Data (NSD) prevented us from conducting usability tests earlier. However, it is not likely that it would have increased the participation percentage. Therefore, a strategy to mitigate the low participation count is to divide the usability tests into two rounds. A key benefit is that usability problems from the first round can be mitigated, reducing the number of usability problems in the second round.

Few Teaching Assistants (TAs) limit the validity of their feedback. The two TAs provided this thesis with much insight into their process, valuable suggestions for improvements, and future use cases. However, only *two* TAs that partook are not sufficient to provide confident results. Only 2 out of 11 available TAs (group supervisors) answered and agreed to participate in either of the interview rounds. Options to increase the response rate could have been to contact TAs early in the process to avoid interviews conflicting with other assignments and exams. Alternatively, contact previous TAs from 2020. However, a more suitable option is to conduct a field study, as mentioned in the previous paragraphs, and include TAs as participants.

All participants were recruited from the same Software Engineering course. Even though all participants also had taken the course TDT4140, and often compared the value of the visualization tool for both IT1901 and TDT4140, the findings are still not generalizable. IT1901 were selected because of their large participation counts, course difficulty, and use of group assignments. Direct recruitment from TDT4140 or other similar courses may increase the diversity and participation count. However, this would also risk diluting findings or receiving more contradicting responses because of different learning goals and course structures.

Participants receiving gift cards may have influenced the feedback of the usability tests. Considerations should be made when using payment, such as gift cards, to incentivize participation in interviews. This is particularly important when the participants are a vulnerable group. For example, some participating students and TAs could feel an obligation to answer positively due to getting a payment, or if they thought their answers could affect their grades or chances to become hired as TAs at a later semester. In order to mitigate such problems, every participant is informed through an interview consent form (Appendix D) that their participation had no implication to their relations with NTNU or its employees, what so ever.

The thesis does not consider alternatives to visualization tools. Most related work (Chapter 3) and similar tools (Section 2.4) use visualizations to provide feedback to developers, students, and course staff. However, there is a possibility that alternative systems can be more suitable. A tool integrated into a Continuous Integration (CI) & Continuous Delivery (CD) pipeline for each project, or alerting system, could potentially give students more direct feedback. In the course IT1901 using CI & CD pipelines are optional. In this thesis, looking at alternatives to visualization tools was considered out of scope.

Chapter 8

Conclusion and Future Work

Version Control Systems (VCSs) and supporting tools have become a central utility for students in Software Engineering courses, for its relevance in the industry, including its utility in collaboration with peers. For educators, such systems generate massive data and potential insights into how student groups collaborate and solve problems. Existing research has found multiple uses of the data generated by Git, GitLab and similar tools to help educators improve the course and assess student contribution. However, few studies have investigated the value of this data to the students directly.

This thesis explored how Software Engineering students would value a visualization tool that gives them continuous feedback on their project by visualizing their Git and GitLab usage. Over two rounds, a prototype was built and tested on 11 students and Teaching Assistants (TAs). The data collected are primarily qualitative, containing reflections from students and TAs and statements and themes coded from audio transcriptions.

Two main contributions are made from the results of this thesis. *1) Gained knowledge into how students in Software Engineering courses value continuous feedback of their Git and GitLab usage, through a visualization tool. 2) Increased knowledge of the benefits and limitations of using third-party visualization tools, particularly focusing on Grafana, in Software Engineering courses*. Feedback from students and our own experience working with the third-party visualization software indicate that the effort required to customize it for use in student projects may be too high. However, feedback from TAs show promises for the value of giving them access to insights about student groups from third-party software to support improved counseling of students. This should be researched further.

This knowledge can be used with existing research to design and build a complete system that gives students continuous feedback on their projects based on data generated from relevant data sources, such as Git and GitLab. Therefore, supporting and streamlining students' learning journey.

8.1 Future work

Future research should conduct a field study throughout the duration of a Software Engineering course to observe students and Teaching Assistants (TAs) behavior in a natural environment and with more participants. The findings from our usability tests are based on what participants think they value after completing the course. An interesting question is whether they would value the same information during the learning process of the course or if other forms of feedback are perceived to be valuable. Alternatively, if the Software Engineering course has enough students, or the tool is planned to be used throughout multiple semesters, running A/B testing could be considered a strategy to run experiments [54].

The field study may benefit from being tested in multiple different Software Engineering courses and at different difficulty levels. Students likely value different forms of feedback based on the teaching goals of a course, the number of people in each group, or what existing knowledge the students have. Most students participating in the interviews in this thesis had little knowledge of Software Engineering beforehand, resulting in the visualized metrics being tailor-made for them. However, students in advanced courses likely appreciate other metrics or consider the proposed visualization tool redundant.

Commit messages were discussed as a promising data source in Section 7.1. However, as the data is based on unstructured text from developers, analysis of the data is fairly complex, even though Git best practices favor some structure and consistency. A simple count of duplicates was promising during the usability tests, with some participating students being able to reflect on their group's Git commit practices. Conversely, the other participants generally found little value in the data or did not find the visualizations understandable. Although inconclusive, this shows that there might be potential in presenting students with the times every commit message has been duplicated. However, more work is needed to find better ways of mining the data and visualizing it.

The interviews with TAs revealed several benefits of using a third-party visualization tool, such as Grafana, to help them analyze the progress and behavior of student groups to counsel them better. The idea of using Git and GitLab data for consultation is not new [1, 6, 39–41]. However, these mainly use custom-built visualization tools or generated reports. A field study observing how Course staff would utilize the third-party visualization tool during a Software Engineering course and what benefits or limitations it has could be interesting. Notable questions would be how much time Course staff could save utilizing such tools and how it affects the counseling process.

Bibliography

[1] Å. Haugse and T. Aalberg, "Git in an educational context," eng, 2021, ISSN: 1892-0713. [Online]. Available: https://hdl.handle.net/11250/2991478.

[2] J. Chen, G. Qiu, L. Yuan, L. Zhang, and G. Lu, "Assessing teamwork performance in software engineering education: A case in a software engineering undergraduate course," in *2011 18th Asia-Pacific Software Engineering Conference*, Dec. 2011, pp. 17–24. DOI: 10.1109/APSEC.2011.50.

[3] B. Oakley, D. Hanna, Z. Kuzmyn, and R. Felder, "Best practices involving teamwork in the classroom: Results from a survey of 6435 engineering student respondents," *Education, IEEE Transactions on*, vol. 50, pp. 266–272, Sep. 2007. DOI: 10.1109/TE.2007.901982.

[4] C. Ghezzi and D. Mandrioli, "The challenges of software engineering education," in *Software Engineering Education in the Modern Age*, P. Inverardi and M. Jazayeri, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 115–127, ISBN: 978-3-540-68204-2.

[5] J. Hayes, T. Lethbridge, and D. Port, "Evaluating individual contribution toward group software engineering projects," in *25th International Conference on Software Engineering, 2003. Proceedings.*, 2003, pp. 622–627. DOI: 10.1109/ICSE.2003.1201246.

[6] H. Tarmazdi, R. Vivian, C. Szabo, K. Falkner, and N. Falkner, "Using learning analytics to visualise computer science teamwork," in *Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education*, ser. ITiCSE '15, Vilnius, Lithuania: Association for Computing Machinery, 2015, pp. 165–170, ISBN: 9781450334402. DOI: 10.1145/2729094.2742613. [Online]. Available: https://doi.org/10.1145/2729094.2742613.

[7] M. A. Busseri, M. A. Busseri, and J. M. Palmer, "Improving teamwork: The effect of self-assessment on construction design teams," *Design studies*, vol. 21, no. 3, pp. 223–238, 2000, ISSN: 0142-694X.

[8] S. Chacon and B. Straub, *Pro Git*, 2nd ed. Springer Nature, 2014. [Online]. Available: https://git-scm.com/book/en/v2.

[9] N. N. Zolkifli, A. Ngah, and A. Deraman, "Version control system: A review," eng, *Procedia computer science*, vol. 135, pp. 408–415, 2018, ISSN: 1877-0509.

[10] Stack Overflow, *Stack Overflow Developer Survey 2021*. [Online]. Available: https://insights.stackoverflow.com/survey/2021#technology-most-popular-technologies.

[11] University of Leeds, *What is grey literature?* https://library.leeds.ac.uk/info/1110/resource_guides/7/grey_literature, Accessed 2022-05-19, Apr. 2017. [Online]. Available: https://library.leeds.ac.uk/info/1110/resource_guides/7/grey_literature.

[12] Conventional Commits, *Conventional Commits*, Accessed 2022-05-19, May 2022. [Online]. Available: https://www.conventionalcommits.org/en/v1.0.0/#specification.

[13] P. Hammant, *Trunk Based Development*, https://trunkbaseddevelopment.com/, Accessed 14.03.2022, 2017. [Online]. Available: https://trunkbaseddevelopment.com/.

[14] L. Matos, *Gitcommitbestpractices.md*, https://gist.github.com/luismts/495d982e8c5b1a0ced4a57cf3d93cf60, Accessed 2022-05-18, Jan. 2019. [Online]. Available: https://gist.github.com/luismts/495d982e8c5b1a0ced4a57cf3d93cf60.

[15] Perforce, *5 git best practices for git commit*, https://www.perforce.com/blog/vcs/git-best-practices-git-commit, Accessed 2022-05-18, Nov. 2019. [Online]. Available: https://www.perforce.com/blog/vcs/git-best-practices-git-commit.

[16] cbeams, *How to write a git commit message*, https://cbea.ms/git-commit/, Accessed 2022-05-18, Aug. 2014. [Online]. Available: https://cbea.ms/git-commit/.

[17] F. Matthew, *Git commit messages: Best practices & guidelines*, https://initialcommit.com/blog/git-commit-messages-best-practices, Accessed 2022-05-19, Mar. 2022. [Online]. Available: https://initialcommit.com/blog/git-commit-messages-best-practices.

[18] M. Tsitoara, "Git best practices," in *Beginning Git and GitHub: A Comprehensive Guide to Version Control, Project Management, and Teamwork for the New Developer*. Berkeley, CA: Apress, 2020, pp. 79–86, ISBN: 978-1-4842-5313-7. DOI: 10.1007/978-1-4842-5313-7_6. [Online]. Available: https://doi.org/10.1007/978-1-4842-5313-7_6.

[19] GitLab B.V., *What are git version control best practices?* https://about.gitlab.com/topics/version-control/version-control-best-practices/, Accessed 2022-05-19, May 2022. [Online]. Available: https://about.gitlab.com/topics/version-control/version-control-best-practices/.

[20] N. Forsgren, N. Forsgren, J. Humble, and G. Kim, *Accelerate : the science behind DevOps : building and scaling high performing technology organizations*, First edition. Portland, Oregon: IT Revolution, 2018, ISBN: 9781942788331.

[21] S. McIntosh, Y. Kamei, B. Adams, and A. E. Hassan, "An empirical study of the impact of modern code review practices on software quality," *Empirical Software Engineering*, vol. 21, no. 5, pp. 2146–2189, 2016, ISSN: 1573-7616. DOI: 10.1007/s10664-015-9381-9. [Online]. Available: https://doi.org/10.1007/s10664-015-9381-9.

[22] C. Sadowski, E. Söderberg, L. Church, M. Sipko, and A. Bacchelli, "Modern code review: A case study at google," in *Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice*, ser. ICSE-SEIP '18, Gothenburg, Sweden: Association for Computing Machinery, 2018, pp. 181–190, ISBN: 9781450356596. DOI: 10.1145/3183519.3183525. [Online]. Available: https://doi.org/10.1145/3183519.3183525.

[23] A. Bacchelli and C. Bird, "Expectations, outcomes, and challenges of modern code review," in *2013 35th International Conference on Software Engineering (ICSE)*, May 2013, pp. 712–721. DOI: 10.1109/ICSE.2013.6606617.

[24] H. Kniberg, *Scrum and XP from the Trenches*, 2nd ed. lulu.com, 2015, ISBN: 9781329224278.

[25] N. Forsgren, M. C. Tremblay, D. VanderMeer, and J. Humble, "DORA Platform: DevOps Assessment and Benchmarking," in *Designing the Digital Transformation*, A. Maedche, J. vom Brocke, and A. Hevner, Eds., Cham: Springer International Publishing, 2017, pp. 436–440, ISBN: 978-3-319-59144-5.

[26] C. Wohlin, "Guidelines for snowballing in systematic literature studies and a replication in software engineering," *ACM International Conference Proceeding Series*, pp. 1–10, May 2014. DOI: 10.1145/2601248.2601268.

[27] J. Kay, N. Maisonneuve, K. Yacef, and O. Zaïane, "Mining patterns of events in students' teamwork data," in *Proceedings of the Workshop on Educational Data Mining at the 8th International Conference on Intelligent Tutoring Systems (ITS 2006)*, 2006, pp. 45–52.

[28] S. Hamer, C. Quesada-López, A. Martínez, and M. Jenkins, "Using git metrics to measure students' and teams' code contributions in software development projects," *CLEI electronic journal*, vol. 24, no. 2, 2021, ISSN: 0717-5000. DOI: 10.19153/cleiej.24.2.8. [Online]. Available: http://www.clei.org/cleiej/index.php/cleiej/article/download/502/409.

[29] K. Buffardi, "Assessing individual contributions to software engineering projects with git logs and user stories," in *Technical Symposium on Computer Science Education*, ser. SIGCSE '20, ACM, 2020, pp. 650–656, ISBN: 9781450367936. DOI: 10.1145/3328778.3366948. [Online]. Available: https://dl.acm.org/doi/pdf/10.1145/3328778.3366948.

[30] R. M. Parizi, P. Spoletini, and A. Singh, "Measuring team members' contributions in software engineering projects using git-driven technology," in *2018 IEEE Frontiers in Education Conference (FIE)*, IEEE, 2018, pp. 1–5, ISBN: 1538611740. DOI: 10.1109/FIE.2018.8658983.

[31] H.-M. Chen, B.-A. Nguyen, and C.-R. Dow, "Code-quality evaluation scheme for assessment of student contributions to programming projects," *Journal of Systems and Software*, p. 111 273, 2022, ISSN: 0164-1212. DOI: https://doi.org/10.1016/j.jss.2022.111273. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0164121222000358.

[32] M. Macak, D. Kruzelova, S. Chren, and B. Buhnova, "Using process mining for git log analysis of projects in a software development course," *Education and information technologies*, vol. 26, no. 5, pp. 5939–5969, 2021, ISSN: 1360-2357. DOI: 10.1007/s10639-021-10564-6. [Online]. Available: https://link.springer.com/content/pdf/10.1007/s10639-021-10564-6.pdf.

[33] L. Baumstark and M. Orsega, "Quantifying introductory cs students' iterative software process by mining version control system repositories," *J. Comput. Sci. Coll.*, vol. 31, no. 6, pp. 97–104, Jun. 2016, ISSN: 1937-4771.

[34] P. G. Rein, T. S. Tefre, and G. A. Stoica, *Creating a web application supporting git in software development courses in higher education*, Generic, 2021.

[35] D. T. Campbell, "Assessing the impact of planned social change," *Evaluation and Program Planning*, vol. 2, no. 1, pp. 67–90, 1979, ISSN: 0149-7189. DOI: https://doi.org/10.1016/0149-7189(79)90048-X. [Online]. Available: https://www.sciencedirect.com/science/article/pii/014971897990048X.

[36] C. Hundhausen, P. Conrad, A. Carter, and O. Adesope, "Assessing individual contributions to software engineering projects: A replication study," *Computer Science Education*, vol. 0, no. 0, pp. 1–20, 2022. DOI: 10.1080/08993408.2022.2071543. eprint: https://doi.org/10.1080/08993408.2022.2071543. [Online]. Available: https://doi.org/10.1080/08993408.2022.2071543.

[37] Á. M. Guerrero-Higueras, C. Fernández Llamas, L. Sánchez González, A. Gutierrez Fernández, G. Esteban Costales, and M. Á. Conde González, "Academic success assessment through version control systems," *Applied sciences.*, vol. 10, no. 4, p. 1492, 2020, ISSN: 2076-3417.

[38] Á. M. Guerrero-Higueras, V. Matellán-Olivera, G. E. Costales, C. Fernández-Llamas, F. Rodriguez-Sedano, and M. Conde, "Model for evaluating student performance through their interaction with version control systems," *Proceedings of the Learning Analytics Summer Institute Spain*, 2018.

[39] S. Eraslan, K. Kopec-Harding, C. Jay, S. M. Embury, R. Haines, J. C. Cortés Ríos, and P. Crowther, "Integrating gitlab metrics into coursework consultation sessions in a software engineering course," *The Journal of systems and software*, vol. 167, p. 110 613, 2020, ISSN: 0164-1212. DOI: 10.1016/j.jss.2020.110613.

[40] H. Gustavsson and M. Brohede, "Continuous assessment in software engineering project course using publicly available data from github," in *Proceedings of the 15th International Symposium on Open Collaboration*, ser. OpenSym '19, Skövde, Sweden: Association for Computing Machinery, 2019, ISBN: 9781450363198. DOI: 10.1145/3306446.3340820. [Online]. Available: https://doi.org/10.1145/3306446.3340820.

[41] K. A. Gary and S. Xavier, "Agile learning through continuous assessment," in *2015 IEEE Frontiers in Education Conference (FIE)*, vol. 2015, 2015, pp. 1–4, ISBN: 9781479984534. DOI: 10.1109/FIE.2015.7344278. [Online]. Available: https://ieeexplore.ieee.org/document/7344278/.

[42] T. Nguyen and C. Chua, "Predictive tool for software team performance," in *2016 23rd Asia-Pacific Software Engineering Conference (APSEC)*, Dec. 2016, pp. 373–376. DOI: 10.1109/APSEC.2016.063.

[43] E. Aivaloglou and A. v. d. Meulen, "An empirical study of students' perceptions on the setup and grading of group programming assignments," *ACM Trans. Comput. Educ.*, vol. 21, no. 3, Mar. 2021. DOI: 10.1145/3440994. [Online]. Available: https://doi.org/10.1145/3440994.

[44] B. J. Oates, *Researching Information Systems and Computing*, ser. Researching Information Systems and Computing. SAGE Publications, 2006, ISBN: 9781412902243. [Online]. Available: https://books.google.no/books?id=ztrj8aph-4sC.

[45] G. Kim, J. Humble, P. Debois, J. Willis, and N. Forsgren, *The DevOps Handbook, Second Edition: How to Create World-Class Agility, Reliability, & Security in Technology Organizations*, 2nd ed. IT Revolution, Nov. 2021, ISBN: 9781950508402.

[46] W. Adams, "Handbook of Practical Program Evaluation," in 4th ed. Jossey-Bass, Aug. 2015, ch. Conducting Semi-Structured Interviews. DOI: 10.1002/9781119171386.ch19.

[47] J. S. Dumas and J. C. Redish, *A Practical Guide to Usability Testing*, 1st. GBR: Intellect Books, 1999, ISBN: 1841500208.

[48] J. R. Lewis, "Usability testing," *Handbook of human factors and ergonomics*, vol. 12, e30, 2006.

[49] J. Nielsen and T. K. Landauer, "A mathematical model of the finding of usability problems," in *Proceedings of the SIGCHI conference on Human factors in computing systems - CHI '93*, ACM Press, 1993. DOI: 10.1145/169059.169166.

[50] J. Spool and W. Schroeder, "Testing web sites: Five users is nowhere near enough," in *CHI '01 Extended Abstracts on Human Factors in Computing Systems*, ser. CHI EA '01, Seattle, Washington: Association for Computing Machinery, 2001, pp. 285–286, ISBN: 1581133405. DOI: 10.1145/634067.634236. [Online]. Available: https://doi.org/10.1145/634067.634236.

[51] D. A. Norman, *The psychology of everyday things*. Basic books, 1988, ISBN: 9780465067091.

[52] C. Auerbach and L. B. Silverstein, *Qualitative data: An introduction to coding and analysis*. NYU press, 2003, vol. 21.

[53] R. T. Mercuri, "Security watch: Computer security: Quality rather than quantity," eng, *Communications of the ACM*, vol. 45, no. 10, pp. 11–14, 2002, ISSN: 0001-0782.

[54] Y. Xu, N. Chen, A. Fernandez, O. Sinno, and A. Bhasin, "From infrastructure to culture: A/b testing challenges in large scale social networks," in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '15, Sydney, NSW, Australia: Association for Computing Machinery, 2015, pp. 2227–2236, ISBN: 9781450336642. DOI: 10.1145/2783258.2788602. [Online]. Available: https://doi.org/10.1145/2783258.2788602.

Appendix A

Interview Guide - Students

Intervjuguide Studenter

Informere om hva vi skal gjøre

Vi lager en analyse plattform hvor studenter i programvareutviklingsfag skal kunne logge inn og få en bedre oversikt over hvordan de jobber og samarbeider med Git og GitLab. Plattformen er spesielt tiltenkt IT1901 – Informatikk prosjektarbeid 1, men er også egnet for bruk i andre lignende fag, feks. Programvareutvikling eller Webutvikling.

Vi ønsker å vise frem en prototype av en slik plattform, la deg navigere litt rundt og gjøre noen oppgaver og så stille deg noen spørsmål til slutt, slik at vi får oversikt over hvordan plattformen skal være for å skape mest mulig verdi for studentene.

Litt formelt rundt personvern som vi må gå igjennom:
Vi kommer til å ta opp intervjuet på denne lydopptakeren fra NTNU. Kun vi to vil ha tilgang til lydopptaket, som i etterkant blir transkribert og anonymisert. Taleopptakene blir slettet når oppgaven er ferdig og i oppgaven vil alle referanser til intervjuene være anonymisert. Hvis du skulle ønske å trekke tilbake ditt samtykke til å være med i prosjektet, kan du gjøre dette ved å ta kontakt med en av oss to eller veilederen vår, George Adrian Stoica.

Bakgrunnsinformasjon:

Hvilken linje går du?

Hvilket studieår er du på?

Hva slags fag har du tatt med gruppebaserte programmeringsprosjekter?

- *IT1901 - Informatikk prosjektarbeid 1*
- *IT2901 - Informatikk prosjektarbeid 2 (bacheloroppgave) / Kundestyrt*
- *IT2810 - Webutvikling*
- *TDT4140 – Programvareutvikling*
- *Andre?*

Hadde du noen tidligere erfaring med GitLab og Git før du tok et av disse fagene?

Hadde du noen erfaring med programmering før du tok IT1901?

- Var noe av dette programmering i gruppe?

- Hva slags erfaring hadde du med utviklingsprosess før faget? Feks. Scrum, kanban, etc.

Overgangsspørsmål

Kan du fortelle om hvordan dere jobbet underveis i faget?

- Opprette Issues og milestones. Hvilke label dere satt på disse
- Hva prosess dere hadde for å tildele folk oppgaver
- Hva prosess dere hadde for Peer Review, som å lage Merge Requests, se over disse og flette inn til master. (Få ut om dette var noe de begynte å gjøre senere i prosjektet).

Hva prosess dere hadde ved Parprogrammering.

- Gjorde dere parproggrammering hele tiden, ble det gjort sporadisk eller begynt senere i prosjektet.
- Brukte dere co-author funksjonalitet i Git/GitLab for å markere når flere bidro?

Hvilke utfordringer hadde dere med samarbeid i IT1901?

Hvilke utfordringer hadde dere med oppfølging fra studentassistenter/fagstabben?

- Var det tilbakemeldinger dere kunne fått tidligere.
- Måtte dere gjøre større omstruktureringer/revideringer etter tilbakemeldingene. I så fall hvilke,
- og kunne de blitt oppdaget/påpeikt tidligere?

Hadde dere noen andre utfordringer under IT1901 som du tror kan være relevant?

Hvilke verktøy brukte dere i IT1901? Hvorfor var disse verdifulle / Hvorfor ikke?

Hvordan syntes du det var å lære bruk av samarbeidsverktøy som GitLab og Git?

Oppgave

Du er med på en gruppe med 2 andre studenter og dere jobber med å lage en hotell-bookingside. Prosjektet er delt inn i 3 innleveringer gjennom prosjektet og dere bruker GitLab og Git som samarbeidsverktøy. I faget vurderes ikke bare koden dere leverer inn, men også måten dere samarbeider på. Dere er akkurat ferdig med andre innlevering i faget og skal i gang med siste innlevering. Foreleseren deres har akkurat lagt ut en beskjed på Blackboard om et verktøy som dere kan bruke for å få oversikt over prosjektene deres og se hvordan dere kan utbedre måten dere samarbeider på i GitLab og git. Du går derfor inn på Grafana, som verktøyet kalles.

Påpeik at alle tankar og poeng er relevante. Ikkje ver for kritisk

Gå inn på Grafana og få oversikt over prosjektet som ligger der.

Hva bør gruppa gjøre for å forbedre prosessen sin?

Gå igjennom de ulike panelene;

Panel	**Hva forteller denne visualiseringen deg?**	**Hva tenker du må endres for å score bedre på denne metrikken?**
Pie-chart		
Lines contributed to project		
Problems in commits		
Problems with Merge Requests		
Problems with GitLab issues		

Spørsmål

Hva syntes du om plattformen?

- Hva var bra?
- Hva bør forbedres eller fjernes?
- Noe du ikke forsto?

Hva føler du mangler ved plattformen?

Hvordan hadde du brukt et slikt verktøy i et gruppearbeid?

Hva er verdien til en slik plattform?

- Hadde du brukt en slik plattform?
- Hvor ofte tror du hadde besøkt plattformen?

Hvilke utfordringer ser du ved en slik plattform?

Noe mer du vil legge til som du ikke fikk sagt?

Oppsummerende tanker (for intervjuere)

Nøkkelpoeng fra intervjuet, eventuelle forbedringer av selve intervjuet.

Appendix B

Interview Guide - TAs

Intervjuguide TA

Informere om hva vi skal gjøre

Vi lager en analyse plattform hvor studenter i programvareutviklingsfag skal kunne logge inn og få en bedre oversikt over hvordan de jobber og samarbeider med Git og GitLab. Plattformen er spesielt tiltenkt IT1901 – Informatikk prosjektarbeid 1, men er også egnet for bruk i andre lignende fag, feks. Programvareutvikling eller Webutvikling.

Vi ønsker å vise frem en prototype av en slik plattform, la deg navigere litt rundt og gjøre noen oppgaver og så stille deg noen spørsmål til slutt, slik at vi får oversikt over hvordan plattformen skal være for å skape mest mulig verdi for studentene.

Litt formelt rundt personvern som vi må gå igjennom:
Vi kommer til å ta opp intervjuet på denne lydopptakeren fra NTNU. Kun vi to vil ha tilgang til lydopptaket, som i etterkant blir transkribert og anonymisert. Taleopptakene blir slettet når oppgaven er ferdig og i oppgaven vil alle referanser til intervjuene være anonymisert. Hvis du skulle ønske å trekke tilbake ditt samtykke til å være med i prosjektet, kan du gjøre dette ved å ta kontakt med en av oss to eller veilederen vår, George Adrian Stoica.

Bakgrunnsinformasjon

Kan du fortelle litt om rollen du hadde i IT1901?

- Hvor lenge du har vært i rollen,
- dine ansvarsområder,
- Tatt faget selv som student?

Hoveddel

Hvordan veiledet du studentene i prosjektet?

- Hvordan var et veiledningsmøte organisert?
- Gjorde du noen forberedelser på forhånd?
- Gjorde studentene noen forberedelser før veiledningen?

Hva var typiske spørsmål/utfordringer studentene hadde?

Hva var typiske tilbakemeldinger studentene kunne få i løpet av prosjektet?

- Var det noen av disse tilbakemeldingene som kunne blitt fanget opp tidligere og rettet av studentene?

Vise prototypen

Hvordan kan disse panelene hjelpe deg med å veilede studentene?

- Hvilke paneler fremstår som de mest nyttig?
- Er det noen fargeterskler som bør justeres på for å være mer nyttige?
- Er dette poeng som har blitt veiledet / undervist om?

Hvilke paneler tror du ikke vil være nyttige?

Hvilke andre paneler/grafer kan være nyttig for studentene?

Hvordan tror du studentene kommer til å bruke et slikt produkt? Og hvor aktivt?

Hvordan hadde du brukt dette verktøyet til å veilede/hjelpe studentene? (Eventuelt ikke brukt)

Avslutning

Noe mer å legge til?

Oppsummerende notater (mellom intervjuere)

(Fri skriving rundt umiddelbare tanker eller noe vi bør rette på)

Appendix C

NSD Approval

Vurdering

Referansenummer
731360

Prosjekttittel
Masteroppgave med bruk av data fra Git og GitLab til å forbedre læringsutbytte i IT1901

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

Prosjektansvarlig
George Adrian Stoica

Student
Fredrik Førde Lindhagen

Prosjektperiode
31.03.2022 - 31.07.2022

Dato
05.04.2022

Type
Standard

Kommentar
Det er vår vurdering at behandlingen vil være i samsvar med personvernlovgivningen så fremt den gjennomføres i tråd med det som er dokumentert i meldeskjemaet 05.04.2022 med vedlegg, samt i meldingsdialogen mellom innmelder og Personverntjenester. Behandlingen kan starte.

TYPE OPPLYSNINGER OG VARIGHET
Prosjektet vil behandle alminnelige personopplysninger frem til 31.07.2022.

LOVLIG GRUNNLAG
Prosjektets formål er å undersøke hvordan studenter ved kurset IT1901 kan øke sitt læringsutbytte av kurset og hvordan platformene Git og Gitlab kan gjøres så verdifull som mulig. For å oppnå formålet vil prosjektet undersøke tidligere studenters opplastede arbeid. Da studentene tok kurset samtykket de til at data fra kurset i form av git og gitlab data kunne benyttes til forskningsformål etter endt kurs.

Informasjonen studentene mottok om denne behandlingen oppfylte ikke kravene til informasjon i personvernforordningen, og det avgitte samtykke oppfyller dermed ikke kravene i art. 6 nr. 1 a. Det avgitte samtykke kan sees som et forskningsetisk samtykke. Prosjektet vil derfor behandle overnevnte kategorier av personopplysninger med grunnlag i at oppgaven er nødvendig for å utføre en oppgave i allmennhetens interesse og for formål knyttet til vitenskapelig forskning. Lovlig grunnlag for behandlingen av alminnelige personopplysninger er dermed at den er nødvendig for å utføre en oppgave i allmennhetens interesse, jf. personvernforordningen art. 6 nr. 1 bokstav e, samt for formål knyttet til vitenskapelig forskning, jf. personopplysningsloven § 8, jf. personvernforordningen art. 6 nr. 3 bokstav b. Behandlingen er omfattet av nødvendige garantier for å sikre den registrertes rettigheter og friheter, jf. personvernforordningen art. 89 nr. 1.

PERSONVERNPRINSIPPER
Personverntjenester vurderer at den planlagte behandlingen av personopplysninger vil følge prinsippene i personvernforordningen: - formålsbegrensning (art. 5.1 b), ved at personopplysninger samles inn for spesifikke, uttrykkelig angitte og berettigede formål, og ikke viderebehandles til nye uforenlige formål - dataminimering (art. 5.1 c), ved at det kun behandles opplysninger som er adekvate, relevante og nødvendige for formålet med prosjektet - lagringsbegrensning (art. 5.1 e), ved at personopplysningene ikke lagres lengre enn nødvendig for å oppfylle formålet

DE REGISTRERTES RETTIGHETER
Så lenge de registrerte kan identifiseres i datamaterialet vil de ha følgende rettigheter: innsyn (art. 15), retting (art. 16), sletting (art. 17), begrensning (art. 18) og protest (art. 21). Vi vurderer at det er grunnlag for å unnta fra informasjonsplikten etter art. 14 nr. 5 b), der personopplysninger ikke har blitt samlet inn fra den registrerte. De registrerte består av studenter som tidligere har gjennomført kurset IT1901 ved NTNU. Informasjonen studentene mottok om denne behandlingen da de tok kurset oppfylte ikke forordningens krav til informasjon. Det vil likevel forsøkes å å informere studentene om behandlingen ved å sende ut informasjon gjennom NTNUs læringsportal Blackboard. 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
Personverntjenester legger til grunn at behandlingen oppfyller kravene i personvernforordningen om riktighet (art. 5.1 d), integritet og konfidensialitet (art. 5.1. f) og sikkerhet (art. 32). Microsoft Office er databehandler i prosjektet. Vi legger til grunn at behandlingen oppfyller kravene til bruk av databehandler, jf. art 28 og 29. For å forsikre dere om at kravene oppfylles, må dere følge interne retningslinjer og/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 oss 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 oss før endringen gjennomføres.

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

Lykke til med prosjektet!

Appendix D

Interview Consent form

Vil du delta i forskningsprosjektet

” Bruk av data fra GitLab til å forbedre læringsutbytte i IT1901”?

Dette er et spørsmål til deg om å delta i et forskningsprosjekt hvor formålet er å *utvikle en plattform for å gi studentgruppene i IT1901 økt innsikt i hvordan de samarbeider på utviklingsprosjektene i faget*. I dette skrivet gir vi deg informasjon om målene for prosjektet og hva deltakelse vil innebære for deg.

Formål

Forskningsprosjektet er del av en masteroppgave på Institutt for Datateknologi og Informatikk (IDI) ved NTNU. Formålet er å kunne tilby studentene i faget IT1901 en plattform som gir dem bedre innsikt i hvordan de som gruppe samarbeider med bruk av Git og GitLab. Plattformen er tenkt å vise en rekke grafer og metrikker som blant annet skal gi gruppene innsikt i hvordan arbeid er fordelt i gruppen og eventuelle forbedringspotensialer. For å gjøre denne plattformen så verdifull som mulig trenger vi i innsikt i studenter og ansattes erfaringer fra ulike gruppearbeid i programmeringsfag. Vi trenger særlig å få innsikt i hvilke visualiseringsteknikker og metrikker som er nyttige for studentgrupper i programmeringsprosjekter.

Hvem er ansvarlig for forskningsprosjektet?

Forskningsprosjektet utføres av masterstudentene Fredrik F. Lindhagen og Sigurd M. Melsom med veiledning av førsteamanuensis George Adrian Stoica ved IDI NTNU.

Hvorfor får du spørsmål om å delta?

Du er ønsket som deltager på dette intervjuet som følge av dine erfaringer som student med lignende programmeringsfag eller har vært ansatt av NTNU i lignende fag.

Hva innebærer det for deg å delta?

Hvis du velger å delta i prosjektet, innebærer det at du deltar på et intervju. Det vil ta ca. 20-40 minutter. Intervjuet vil inneholde spørsmål om erfaringer relatert til programmering og samarbeid i programmeringsprosjekter, samt en brukertest av en prototype. Vi tar lydopptak og notater fra intervjuet.

Det er frivillig å delta

Det er frivillig å delta i prosjektet. Hvis du velger å delta, kan du når som helst trekke samtykket tilbake uten å oppgi noen grunn. Alle dine personopplysninger vil da bli slettet. Det vil ikke ha noen negative konsekvenser for deg hvis du ikke vil delta eller senere velger å trekke deg.

Din deltagelse i dette prosjektet vil på ingen måte påvirke ditt forhold til NTNU eller dets ansatte.

Ditt personvern – hvordan vi oppbevarer og bruker dine opplysninger

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

Det er kun masterstudentene Fredrik Førde Lindhagen og Sigurd Marius Melsom, samt veileder George Adrian Stoica som vil ha tilgang til dataene. Lydopptak av intervjuene vil bli gjort på lydopptakere eid av NTNU, og lagres på NTNU sin Office 365-plattform som er adgangsbegrenset og beskyttet med tofaktorautentisering. Notater lagres også på denne plattformen.

I rapporten brukes anonymiserte koder som Student 1, Student 2 og lignende, når vi trenger å henvise til konkrete tilbakemeldinger, og kan garantere at tilbakemeldingen ikke kan spores tilbake til en bestemt person. For ansatte vil lignende praksis brukes, med Ansatt 1, 2, osv. Unntaket er når en ansatt har samtykket til at vi kan henvise til ansattrolle.

Hva skjer med opplysningene dine når vi avslutter forskningsprosjektet?

Opplysningene anonymiseres når prosjektet avsluttes/oppgaven er godkjent, som etter planen er innen 31. juli 2022. Etter dette vil lydopptak og personopplysninger slettes.

Dine rettigheter

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

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

Hva gir oss rett til å behandle personopplysninger om deg?

Vi behandler opplysninger om deg basert på ditt samtykke.

På oppdrag fra *Institutt for Datateknologi og Informatikk (IDI) ved NTNU* har NSD – Norsk senter for forskningsdata AS vurdert at behandlingen av personopplysninger i dette prosjektet er i samsvar med personvernregelverket.

Hvor kan jeg finne ut mer?

Hvis du har spørsmål til studien, eller ønsker å benytte deg av dine rettigheter, ta kontakt med Institutt for Datateknologi og Informatikk (IDI) NTNU ved:

- Førsteamanuensis ved IDI NTNU George Adrian Stoica – stoica@ntnu.no

Vårt personvernombud:

- Personvernombud ved NTNU, Thomas Helgesen – thomas.helgesen@ntnu.no

Hvis du har spørsmål knyttet til NSD sin vurdering av prosjektet, kan du ta kontakt med:

- NSD – Norsk senter for forskningsdata AS på epost (personverntjenester@nsd.no) eller på telefon: 55 58 21 17.

Med vennlig hilsen

George Adrian Stoica
(Forsker/veileder)

Fredrik F. Lindhagen og Sigurd M. Melsom
(Masterstudenter)

Samtykkeerklæring

Jeg har mottatt og forstått informasjon om prosjektet, og har fått anledning til å stille spørsmål. Jeg samtykker til:

- ☐ å delta i intervjuet
- ☐ *at opplysninger om min ansattrolle publiseres slik at jeg kan indirekte gjenkjennes – hvis aktuelt*

Jeg samtykker til at mine opplysninger behandles frem til prosjektet er avsluttet

---

(Signert av prosjektdeltaker, dato)

Fredrik F. Lindhagen, and Sigurd M. Melsom

Visualizing Repository Data to Facilitate Feedback in SE courses

NTNU
Norwegian University of
Science and Technology