Abstract

The command-line interface (CLI) is by many recognized as being restrained to software developers and power users. However, as users get more experienced, they will usually start to prefer the CLI over the modern graphical user interface (GUI), largely due to the fact that the GUI does not allow the same range of control.

While it is clear that the CLI can achieve more efficient and effective execution of tasks, it comes with the limitation of being more challenging to use compared to the GUI. However, when looking at other systems that support both user interfaces, the CLI is often the most popular. Take Git as an example, the majority prefer the CLI due to the fact that it is designed to better scale the functionality and control requirements of the system.

This thesis will explore the challenges in developing a CLI as an alternative to the existing Blackboard LMS web interface. By examining how agile principles and practices can contribute in the development, we will look at advantages and disadvantages of the two systems.

Our assumption that the CLI would for many be difficult to use turned out to be false as users quickly learned how to effectively utilize it. It was discovered that it is difficult to measure efficiency between a CLI and GUI as it usually comes down to preference and experience. However, an experiment shows that the CLI is in fact more efficient when it comes to loading times for certain tasks. The project is made open source with the aim for further expansion with the help of contributors, as there is a clear motivation among students and teachers at NTNU for such a product.

Sammendrag

Kommandolinjen er av mange kjent for å være begrenset til programvareutviklere og 'power users'. Derimot, når brukere blir mer erfarne vil de vanligvis begynne å foretrekke kommandolinjen framfor det moderne, grafiske brukergrensesnittet, i stor grad på grunn av at det grafiske brukergrensesnittet ikke tillater samme rom for kontroll.

Selv om det er klart at kommandolinjen kan oppnå mer effektiv utførelse av oppgaver, kommer den med begrensningen av å være mer krevende å bruke sammenlignet med det grafiske brukergrensesnittet. Derimot, hvis man ser på andre systemer som støtter begge brukergrensesnittene, er kommandolinjen ofte mest populær. Ta Git som et eksempel, majoriteten foretrekker kommandolinjen siden den er designet for å bedre skalere til funksjonaliteten og kravene til kontroll for systemet.

Denne avhandlingen vil utforske utfordringene rundt det å utvikle et kommandolinjeverktøy som et alternativ til det allerede eksisterende grafiske brukergrensesnittet til Blackboard LMS. Ved å undersøke hvordan smidige prinsipper og praksiser kan bidra til utviklingen, vil vi se på fordelene og ulempene til de to systemene.

Antagelsen vår om at kommandolinjen ville for mange bli vanskelig å anvende viste seg å ikke stemme, da brukere raskt lærte seg hvordan systemet kunne brukes effektivt. Det ble observert at det er vanskelig å måle effektivitet mellom de to systemene, da dette vanligvis kommer an på hva man foretrekker å bruke, samt hva man har erfaring med. Derimot viste et eksperiment at lastetiden var i mange tilfeller mye lavere med kommandolinjeverktøyet. Prosjektet har blitt lagt ut med åpen kildekode med mål om å utvide funksjonalitet ved hjelp av bidragsytere, siden det er et tydelig engasjement for et slikt produkt på NTNU.

Problem Description

Purpose of the Assignment

Development of a command-line tool suite to interact with the Blackboard LMS.

Short Description of Assignment Proposal

The conventional way to interact with the Blackboard LMS is by using a web browser. For many tasks this is inefficient and awkward. The aim of this project is to develop a full command-line toolsuite supporting as much of the core Blackboard Learn functionality as possible (time permitting).

Blackboard Learn servers expose a REST API which can be used as a starting point for exploration. However, it may be necessary to design the tools to interact with the web page endpoints and a suitable HTML parsing framework such as BeautifulSoup. This will be determined through in the initial phase of the project.

The command-line toolsuite should follow a simple and intuitive design. The syntax structure of the ”git” command- line tool can be used as a guide, with different functions of the tool accessed via command-line parameters. When an editor is necessary, an editor chosen from the user’s $EDITOR environment variable can be used (e.g., Vim, Emacs, Nano), depending on the OS of the user.

A core set of must-have student and staff features can be brainstormed for the requirements documentation, and additional features added as time permits. The tool must ensure a user is authenticated and cache the session cookie for future invocations of the tool.

An example staff command could be:

```
$ bbcli new-announcement "IDATT2202" "Lecture topic for next Monday"
"Next Monday's lecture will be on CPU scheduling on unicore and multicore
processors. The lecture will start at 12.15 in Blackboard Collaborate."
```

An example student command could be:

```
$ bbcli get-assignments "IDATT2202"
```

```
Assignment1 due 2021-10-03
Assignment2 due 2021-10-23
Assignment3 due 2021-10-30
Assignment4 due 2021-11-05
$ bbcli submit-assignment "IDATT2202" "Assignment1"
/home/user/courses/IDATT2202/assignment1.pdf
```

The tool suite should work on any modern operating system (Linux, MacOS, Windows) via the command line. Thus, an interpreted language such as Python would be appropriate.

The tool suite must be open-source and hosted on Gitlab/Github to facilitate contributions from the wider developer community.

Contents

Figures

Tables

Code Listings

Chapter 1

Introduction

User interfaces (UI) have been around ever since the first computers emerged, even before the field of Human-Computer Interaction originated [1]. Consequently, software developers and designers have been designated to create user interfaces that allow tasks to be performed efficiently and intuitively. The Command-Line interface (CLI) is the most basic and accessible user interface in which to interact with computers [2]. However, in spite of its simplicity, the CLI carries some limitations in terms of user experience, specifically towards inexperienced users. It is worth quoting Olofsson and Husltrand [3]:

> However theoretically, as users get more and more experienced they will gradually start to prefer the command-line interface over a graphical user interface because they are able to complete their tasks faster and to maintain more control over what they are doing and what's going on, in general.

Considering the fact that Blackboard Learn is the main learning management system at NTNU, the university with the greatest information technology related faculties in Norway, it is questionable that there does not exist a CLI equivalent to the web user interface. We are pleased to report that during the last semester we have developed a system that solves a number of the issues the web UI encompasses. With focus on ease of use, numerous tasks are now automated to account for some of Blackboards biggest disadvantages: troublesome navigation and inefficiency.

1.1 Research Questions

The main purpose of the task was to develop a CLI to interact with the Blackboard LMS. Hence, the following problem statement was put together: ***What are the challenges in developing a CLI as an effective and efficient alternative to the existing Blackboard LMS web interface?***

A goal was to increase the efficiency of Blackboard, but efficiency is dependent on different factors, especially when dealing with user bias. Therefore, we wanted to explore how efficiency can be measured when comparing the CLI to the web UI. Additionally, it is natural to discuss the advantages and disadvantages when comparing the two UIs. Also, considering that the task is purely based on development, we wanted to research how agile practices and principles could be implemented to enhance the development process. Thus, the following research questions were derived from the problem description to support the problem statement:

1. How can efficiency be measured when comparing the CLI to the web UI?
2. What are the advantages and disadvantages of the CLI in comparison to the web UI?
3. How can agile principles and practices contribute in developing a versatile CLI to increase user experience and efficiency?

1.2 Agenda

Firstly, relevant background on Blackboard, user interfaces and development theory will be described before presenting how we progressed methodically during development. Additionally, the method chapter will provide an overview of the used technologies and dependencies and why these were chosen. Then we will present results on the process and product, followed by evaluating the research questions. We will discuss how efficiency can be measured when comparing a CLI to a web UI, the advantages and disadvantages of the CLI compared to the web UI, and how the agile mindset can contribute in developing alternatives to existing systems.

Chapter 2

Background and Theory

2.1 Blackboard as a Learning Management System

A Learning Management System (LMS) manages and tracks interaction between a student and the content, and the student and the lecturer. The LMS keeps track of the students progress, test scores, help indicate course completions, and help lecturers follow the performance of the students. The LMS must be able to assemble and deliver learning content rapidly, personalize content and enable knowledge reuse [4]. LMS, including Blackboard are the most commonly used types of e-learning systems for both students and lecturers at universities [5]. The origin of LMS has also improved the online communication for students, motivating them to play an active role in the learning process, in contrast to playing a passive role in the traditional way of receiving information through textbooks and physical notes [6].

Advantages
In a survey conducted by Bouhnik and Marcus [7], the participants noted advantages of using e-learning systems which were concretized into the following four categories: (1) Flexibility of the material and the time, (2) Accessibility to the material, (3) Visibility of the multimedia and (4) Availability of the data.

Disadvantages
Servonsky, Lawrence and Bertha [8] addresses some challenges when using the Blacboard LMS, stating that the time it takes to prepare a lecture using Blackboard LMS takes considerably more time than physical lecturing. Additionally, as technology changes, content must be updated and reviewed constantly. This includes uploading documents, changing content, and copying and reposting content. Bouhnik and Marcus also address disadvantages of e-learning platforms, in which the students' dissatisfaction

was based on several experiences: (1) lack of a firm framework, which encourages laziness, (2) it requires high level of self-dicipline, (3) abscence of learning atmosphere, (4) reduces contact between students, (5) the learning process is less efficient compared to face-to-face learning and (6) teachers have difficulty in widening answers to student questions. Concerning efficiency, Bouhnik and Marcus state that the lack of it requires students to spend more time when learning the material.

2.2 User Interfaces

UIs are used to interact with an application or a website [9]. Different types of user interfaces include graphical user interfaces (GUI), CLI, menu-driven user interfaces, voice user interfaces (VUI), form-based user interfaces, natural language user interfaces and-so-forth. GUI and CLI will be addressed further in the following subsections.

2.2.1 Graphical User Interface

GUI uses visual objects such as buttons, input fields, menu bars etc., as well as mouse clicks to interact with the system [10]. The visual objects represents computation entities, and the interaction with the mouse gives an impression of physical action on those entities [11]. A GUI has both virtues and vices, and will be powerful for some purposes, while poor and weak for others [12].

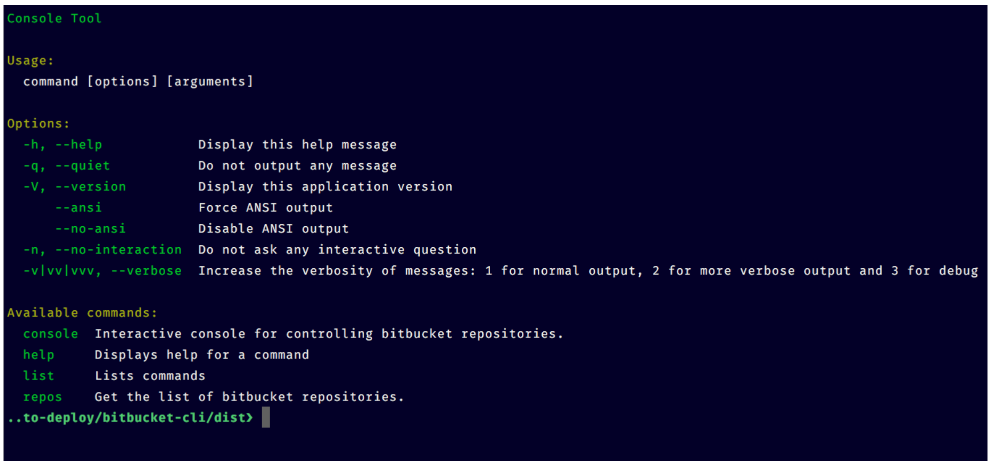
Advantages
Shneiderman [13] states that a GUI can help novice users learn the functionality through demonstration, and experts can easily execute a wide range of tasks. ”For instance, the immediacy of feedback and the natural translation of intentions to actions make some tasks easy” [12]. It can be argued that a GUI is meant for making things simple, because it relies heavily on visual representations [3]. Additionally, Schneiderman states that error messages are rarely necessary, as the GUI is limited to what clickable objects are available on the screen. The result of these clicks should be obvious: either some action takes place or nothing happens.

Disadvantages
Hutchins [12] refers to several downsides of using a GUI. First and foremost, repetitive tasks are best performed with a script, because through a symbolic description it is possible to specify which tasks to accomplish. This is poorly supported in a GUI because visual objects cannot be chained together as sets of operations. The user often relies on several clicks to perform a task, which can lead to an inefficient experience. Hutchins also points out that a well designed interface that exploits semantics can reduce the time it takes to learn the interface. At the same time, it is worth mentioning that interface design is subject to many trade-offs. When designing a GUI, there are certainly instances where it is desired to trade for example

efficiency for directness. A GUI might have articulatory directness when pointing out objects, but at the same time it can be difficult to move the hands between different input devices and have accurate precision when pointing at visual objects [12]. Additionally, it offers less control, because the user is limited to execute actions that are visually available.

2.2.2 Command-Line Interface

Through a CLI, the user uses text to write commands in order to interact with a system or application [14]. Command and response interaction are used to issue a series of commands that the system or program executes. Commands can have arguments, options and flags [15]. Options can come in various types, and can be both optional and required. Basic value option where an argument is accepted is the most common. If no value is provided, the default one will be used. Arguments are similar to options, but they are positional. Flags are options that take the value true if it is specified and false otherwise.



```
Console Tool

Usage:
  command [options] [arguments]

Options:
  -h, --help            Display this help message
  -q, --quiet           Do not output any message
  -V, --version         Display this application version
      --ansi            Force ANSI output
      --no-ansi         Disable ANSI output
  -n, --no-interaction  Do not ask any interactive question
  -v|vv|vvv, --verbose  Increase the verbosity of messages: 1 for normal output, 2 for more verbose output and 3 for debug

Available commands:
  console   Interactive console for controlling bitbucket repositories.
  help      Displays help for a command
  list      Lists commands
  repos     Get the list of bitbucket repositories.
..to-deploy/bitbucket-cli/dist>
```

Figure 2.1: A CLI with commands, options and arguments. [16]

Advantages

Preece [17] expresses several advantages with a CLI, such that it faces less distraction as there is no need for interaction with a graphical user interface. Most interactions with a CLI is only with the use of a keyboard, and according Omanson and others [18], keyboard interactions are favored as being more efficient than mouse interactions. Lastly, Preece expresses that a CLI makes it possible to create advanced shell scripts to solve complex tasks, which particularly experienced users can benefit from.

Disadvantages

Preece stated that a disadvantage with a CLI is that it requires a high cognitive workload, because commands, arguments and options needs to be remembered and used properly together. Hence, a CLI is often limited to experienced users, and can be hard to learn as it is often designed with low guessability (Wobbrock et al. [19]). New inexperienced users have to go through help guides, man pages and other tutorials to increase their proficiency.

2.3 Git as a Use Case

The modern system Git illustrates how having different user interfaces can help reach a versatile user base [20]. Git is a version control system that makes collaboration with developers easier and tracks changes to different working directories [21]. A user can either interact with Git through a CLI or a graphical user interface. Olofsson and Hultsrand [3] addressed that the CLI is the most used among Git users, as it offers more control. A GUI, on the other hand, is meant for simplifying things and offers better visual representations. In the study that Olofsson and Hulstrand conducted, the use of graphical user interfaces declined as knowledge of Git increased. This meant that more experienced users preferred a CLI, because they wanted to optimize work-flows, save time as well as have more control. On the other hand, a CLI is harder to learn, and might be more challenging in terms of cognitive workload as the user has to remember commands, options and arguments. Olofsson and Hulstrand stated that none of these user interfaces can be regarded as the best one, due to the fact that certain target groups may prefer the one over the other. Alltogether, a CLI is associated with terms like ”control” and ”difficulty”, while GUI is related to being ”simple”, ”easy”, ”helpful” and ”time consuming”.

2.4 Existing Alternatives

bb [22] is a CLI developed by RocHack, ”a group of hackers, programmers, and entrepreneurs at the University of Rochester” in 2015. It is used for interacting with a Blackboard Learn installation, currently only with support for University of Rochester. With the CLI users can submit labs and projects from the command-line. The tool is written in only with Bash (Shell script), and uses Curl to communicate with the server, not the official Blackboard Developer API. Consequently, the tool is arguably not future-proof, as small changes in the website will affect its functionality. Due to being limited to University of Rochester, it cannot be tested from people from other universities.

2.5 Development Theory

2.5.1 Agile Development

Agile is a mindset that provides values and principles on how to create and respond to change, and how to deal with uncertainty [23]. The mindset is implemented into software development frameworks, such as Scrum and Kanban. Delivering working software frequently, with high quality and low budget are the main characteristics of agile methods compared to traditional methods [24].

The agile manifesto [25] has derived values and principles from other agile methods and mindsets to create a more united agile movement in the software development industry.

The agile values are:

- Individuals and interactions
- Working software
- Customer collaboration
- Responding to change

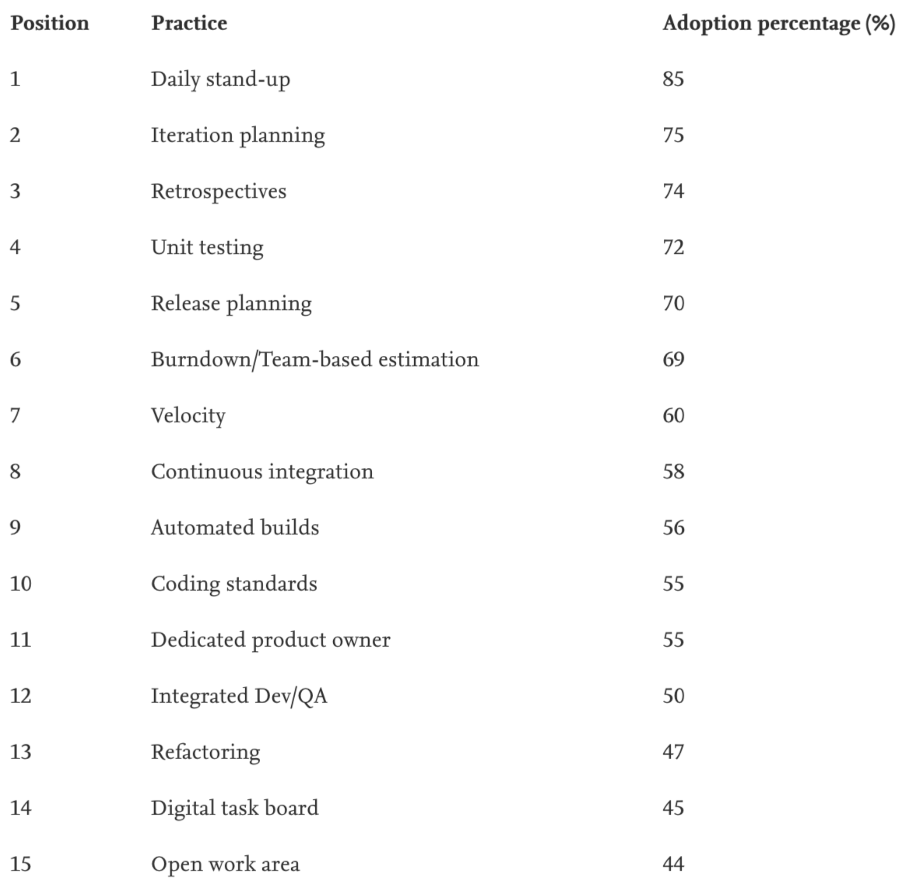
The agile principles are:

- Early and continuous delivery of valuable software
- Change is welcome
- Deliver frequently
- People interaction (business and developers)
- Motivated people
- Face-to-face communication
- Working software is progress
- Constant pace
- Technical excellence and good design
- Simplicity
- Self-organized teams
- Continuous improvement

There are several practices that agile methods usually implement. *Software development practices* include pair programming and unit testing. *Management practices* are for example daily stand-ups and open work area. *Software process practices* include simplicity in design and iterative workflows [23]. Figure 2.2 lists the most adopted agile practices [26]. The key practices used in this project are described in detail below.

2.5.1.1 Daily stand-up meeting

A stand-up meeting is a meeting where the team members typically stand up when participating. This is meant to keep the meeting short and effective. There are, however, other advantages to implementing stand-up meetings in a project. Some usual problems that are solved when teams implement the stand-up meeting practice are shared understanding of goals, coordinated efforts, problem sharing and improvements, and identifying as a team [27]. The meetings typically take place at the same time and place every work day, even if some team members are not present. The stand-up is meant to give everyone a mutual understanding of the project status. Additionally, it is intended for the participants of the development team only, and not for stakeholders and management. Thus, it is informal, quick and concise.



Position	Practice	Adoption percentage (%)
1	Daily stand-up	85
2	Iteration planning	75
3	Retrospectives	74
4	Unit testing	72
5	Release planning	70
6	Burndown/Team-based estimation	69
7	Velocity	60
8	Continuous integration	58
9	Automated builds	56
10	Coding standards	55
11	Dedicated product owner	55
12	Integrated Dev/QA	50
13	Refactoring	47
14	Digital task board	45
15	Open work area	44

Figure 2.2: Most adopted agile practices

2.5.1.2 Iteration planning

Iterative and incremental development is used to improve the technical manageability of projects and avoid the risk of large integration's at the end of the project [28]. This ensures that the next step is implemented successfully without errors, as well as reducing the probability that it will negatively affect other parts of the system.

During iteration planning, the team decides how much of the team backlog each member can complete during the upcoming iteration. The purpose of iteration planning is to gain a common understanding of the work needed to be done, as well as defining a realistic scope for the iteration [29].

2.5.1.3 Retrospectives

The retrospective is a meeting taken place at the end of an iteration of agile software development [30]. This gives the team a chance to reflect on the previous iteration, and can evaluate improvements for the next one. Typical question that are discussed during this meetings are:

- What was successful factors?
- What did not work well?
- What can be done for improvement going forward?

A retrospective meeting shall lead to acquiring more knowledge, and applying changes based on that[31]. To quote Albert Einstein: ”The definition of insanity is doing the same thing over and over again, but expecting different results”.

2.5.1.4 Unit testing

Unit testing involves testing individual parts, often known as units, of the project [32]. This is done by defining the results that the code is expected to produce. The input data shall produce the same result, otherwise the test will fail. There are several reasons why unit testing can be crucial:

- By testing smaller units of the code, one can more precisely locate where errors occur.
- Better code can be produced, because the code is more reusable as it is divided into smaller units.
- Easier for other developers to better understand code, because they can see what the code is expected to produce.

Different approaches range from testing the units before or after the code is implemented. According to agile development, projects are built upon iterative feedback, which means that the tests should be performed afterwards. This is different to *Test Driven Development* where the tests are written before.

2.5.1.5 User Testing

Moderated testing
One-on-one user testing is a way to unveil more than when having the user follow the steps described in the unmoderated tests. The moderator should never interfere the instincts of the tester, but being in the same room as the tester and hearing their thoughts can gain a deeper understanding that otherwise would be let out in the unmoderated tests [33].

Unmoderated testing
Unmoderated testing does not require the presence of anyone other than the participant during the test. It can be done anytime, anywhere, and the participant is free to complete the tasks in their own time. Unmoderated user tests are typically sent to a large number of participants simultaneously, which is favourable if large sample sizes are necessary [34].

2.5.1.6 Continuous integration and deployment

Continuous integration (CI) and continuous deployment (CD) are practices used in software development to implement changes at a more frequent rate [35]. It leads to teams being able integrate code more consistently, and collaborate more effectively. Changes are validated through a build and the automated tests are executed against the build [36]. When code has to go through tests to be merged, the same requirements are expected across the team. This gives a consistent mechanism to integrate and validate changes. CD, on the other hand, releases all the changes that goes through the automated tests to the customers or the users. This will contribute in including users along the process, and thus accelerate the feedback loop.

Chapter 3

Method

In research and development, methodology is distinguished between *research methodology* and *software development methodology*, where the former is about building knowledge based on research and the latter is a software development framework for controlling the process in software development.

This chapter will elaborate on the applied research and development methodology, and how they were used. In addition, a section concerning the implementation of the product will go into detail about the technical implementations carried out during the development phase. Lastly, a simple experiment that compares the efficiency between the CLI and the web UI will be described.

3.1 Development Process

The development process was based on agile methods that revolve around the users and their requirements. It was performed as an iterative process, where each iteration consisted of collecting information about the users and their requirements, followed by development and evaluation in form of testing. Based on this methodology, an inductive approach was performed, where data was gathered empirically first, before grounding theory on this data.

Prior to development, data about the environment (people, technical systems and opportunities) was collected empirically through surveys. The empirical data and our own experiences were the basis for the theory detailed in chapter 2. The goal of the initial research was trifolded: (1) if there was a motivation amongst students and teachers at NTNU for a solution, (2) the target group's requirements, and (3) what the solution aimed to produce and improve.

During the development phase, three iterations were performed. Each iteration consisted of:

Iteration planning: This is where issues from the backlog was distributed between the team members. In addition, a general discussion on the scope and goals of the following iteration was held.

Development: During development, stand-up meetings were performed every morning. At the end of every week, an additional meeting was held, where the weeks work was summarized.

Evaluation: At the end of each iteration, evaluation in form of user testing was performed. This information together with newly discovered science theories and methods laid the foundation for the next iteration. The iterations were concluded with a retrospective.

3.1.1 Data Collection Methods

When collecting data it is important to use different methods to counteract misconceptions and bias from the developers. The methods used were *surveys* and *user tests*, with surveys being the dominant method of data generation prior to the development phase. All forms of data generation also collected the following details about the participants: field of study, role and experience with the command-line. The data generation methods used were:

Survey: The main development goal of the task is based on solving an existing problem, where the hypothesis is that Blackboard's web interface lacks user friendliness and efficiency. Accordingly, surveys were used to gather information about the scope of the problem. The first survey consisted of the following questions:

- Do you think the Blackboard website is user friendly?
- Do you think the Blackboard website is effective to use?
- How experienced are you with using the command line?
- Which functionalities in Blackboard do you think need improvement?
- Would you use a CLI that replaces the web interface to execute tasks in Blackboard more efficiently?

Evaluation: On each iteration during development, user tests were performed to understand how the user base interacts with the product. The user tests were separated into moderated and unmoderated user tests:

- **Moderated testing:** The importance in having participants from different fields of study testing the application in greater detail is essential, especially considering the large and diverse user base at NTNU. A test logger was present on each test and noted insights in which the unmoderated tests might have missed. The tests were informal and open for conversation with the participants, without assisting the user during the test. The conversation assisted in getting further insight in how the participants felt about the product.

- **Unmoderated testing:** For unmoderated testing, a Google forms document with predetermined tasks was issued to a set of contributors at the end of each milestone. The contributors were free to complete the tests in their own time.

The user tests were conducted in three iterations, where the goal of each iteration was to gather information on the status of the CLI, e.g., perception of the CLI syntax, effectivness etc. The user tests consisted of a set of tasks for the participants to follow, and noting whether or not they could manage to finish the task.

In addition, every user test included the following questions:

1. How intuitive and simple do you think the CLI is?
2. Why do you think it is, or is not, intuitive?
3. Do you think the CLI is more effective and simple to use than the web interface?
4. Why or why not?
5. General feedback or tips for improvement?
6. If you had any technical issues with the CLI or installation, please let us know below.

3.1.2 Data Analysis

The data collected from the surveys and user tests was there to give insight in how the users expect a learning management system to work in addition to general testing of the CLI. The data was used to gain deeper knowledge of the use needs of users with different backgrounds and roles. Due to its nature, it is natural to assume the CLI is best suited for students and staff with an IT-background, but data collection was also analysed to see how users in other fields of study can benefit from the product.

3.2 Choice of Technology and Implementation

3.2.1 Blackboard Learn REST API

Blackboard offers a REST API to help developers communicate with Blackboard Learn through an application, plugin or other integration. It contains a number of endpoints for communicating with Blackboard Learn, e.g., for handling courses, announcements, contents, assignments, grading, users, etc.

3.2.2 System Architecture and MVC

It was desired to have a separation of concerns in order to keep the code well structured, and reduce the likeliness of large modifications at multiple places. Thus, *model-view-controller* was a good fit. In this project, the controller is called *commands* and model is called *services*, because it fits better descriptively. The commands were communicating with the respective service, which made the call to the Blackboard API to retrieve or submit the data. Lastly, *view* made sure that outputs were printed in an easy readable format.

From the beginning of the project, it was known that the syntax structure most likely would change during the development due to the focus on responding to user feedback. By using the software design pattern MVC, the CLI was able to separate the code regarding the syntax structure and the code regarding retrieving and posting data, allowing changes to be made in the commands syntax structure without affecting the service logic.

3.2.3 Design Pattern

In terms of design patterns, a builder was implemented for creating URLs. With an URL builder, creating URLs became a more efficient process, and made sure that the correct URL was created, minimizing the risk of typos. In addition, if the API endpoints change in the future, only the URLs in the builder itself need to be modified.

3.2.4 Technological Dependencies

Python

Compatibility with Linux, MacOS, Windows, etc was one of the requirements for the CLI. Therefore Python was a good choice, not only because it is an interpreted language, but also because there exists a number of resources on CLIs.

Pip and PyPI

Pip is a package installer for python and was used in this project to download dependencies that were required in the CLI. To make the CLI easily accessible, it is packaged into a python package and uploaded to PyPi where others may use pip to install the CLI and use it in their terminal.

Beautiful soup

Beautiful soup is a library for extracting data from HTML and XML files. For this project, Beautiful soup was used to scrape the HTML responses from the HTTP requests sent using the Python requests library.

Requests

Requests is a python library used to send HTTP requests and is known for its simplicity. Requests plays an essential part in the CLI since almost every command communicates with the Blackboard Learn REST API through sending HTTP requests.

Click

Click is a python package used to create CLIs. It was regarded as the best alternative, due to its possibilities to highly configure the commands with different options and arguments. Help pages for each command were automatically created, which made navigation through the CLI more convenient. The most valuable feature was the possibility to divide the commands into groups. Hence, the different modules could be made into groups, which strengthened the separation in the CLI. This also made it possible to create sub-commands for the different modules.

3.3 Measuring Efficiency

For a set of tasks, efficiency was measured between the CLI and Web UI in the form of key presses, number of commands, HTTP requests, loading time and Time To Complete (TTC). The tasks were performed by us, the developers due to the fact that we recognize ourselves to have the least amount of bias toward each system. When performing the tests, the minimal amount of commands and clicks needed were used to fully execute the tasks without skipping essential parts, i.e., assuming that the person who carries out the task does not have any remembrance of IDs. Each task was performed on the same computer and on the same Wi-Fi. The measurements were:

Key Presses

The number of key presses performed to finish the task when using the web UI.

Number of Commands

The number of commands performed to finish the task when using the CLI.

HTTP Requests

The total amount of HTTP requests performed to finish the task. By inspecting the site, the total amounts of HTTP requests were shown when performing tasks.

Loading Time

Loading is the total time spent on each task excluding time spent on navigation.

Time To Complete

Time To Complete is the total time spent on a set of tasks including time spent on navigation.

Chapter 4

Results

4.1 Data

This section will describe the data that was collected through surveys and user tests, to aid in the development of the CLI.

4.1.1 Surveys

As described in section 3.1.1, the data collected prior to development was conducted using surveys. The horizontal axis represents the score from 1 to 10, and the vertical axis represents the number of responses. The first survey got 73 responses, from a total of 15 field of studies. 90.9 percent of the participants were students, the rest were teachers and teaching assistants.

As figure 4.1 shows, the majority of the participants rated the user friendliness of the Blackboard website to be below average.

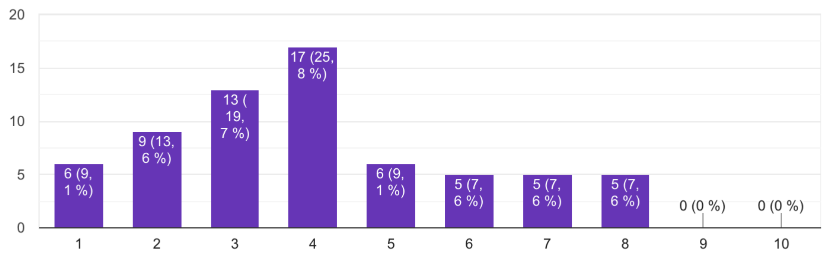


Figure 4.1: Do you think the Blackboard website is user friendly?

When asked about the effectiveness of the Blackboard website, the majority rated it to be below average, as illustrated in figure 4.2

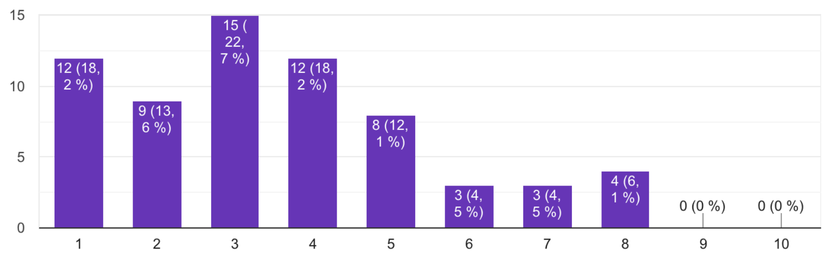


Figure 4.2: Do you think the Blackboard website is effective to use?

When asked about prior experience with using the command-line, 21% of the participants stated that they have not used the command-line before, as shown in figure 4.3

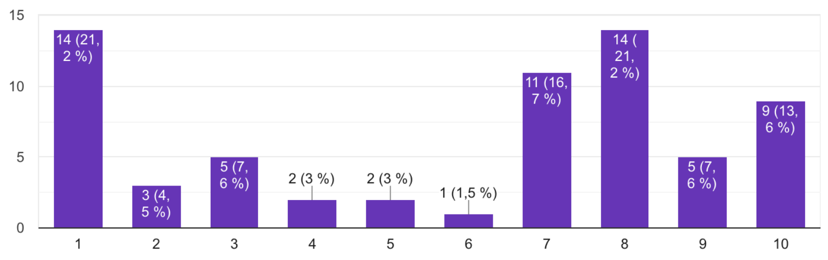


Figure 4.3: How experienced are you with using the command line?

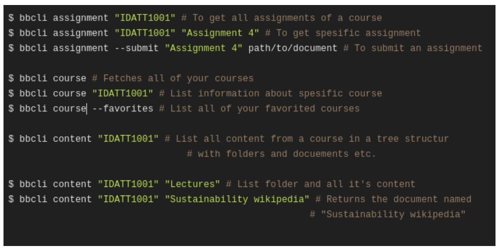
The participants were also asked to list which functionalities in Blackboard that they think need improvement. The most requested functionality improvements were improved navigation, more specifically reducing number of clicks and enhancing the overview of contents in a course. In addition, there was a desire to have the same layout on every course. Key points on surveys can be found in appendix F. Lastly, the participants were asked if they would use a CLI that replaces the web interface to execute tasks in Blackboard more efficiently. Out of the participants, 55.4 percent would use a CLI that replaces Blackboards web interface. 19.6 voted no, and the remaining 25 percent voted that they do not know.

The second iteration of surveys had the goal of finding the preferred CLI syntax. It consisted of two surveys, one for students and one for teachers.

Out of 35 students, 57 percent preferred option 1. The teacher survey got 36 responses, where 52.8 percent voted for option 1.

4.1.2 User Tests

A total of three user test iterations were performed, where each component of the command line tool was tested. The full user tests are listed in appendix E.



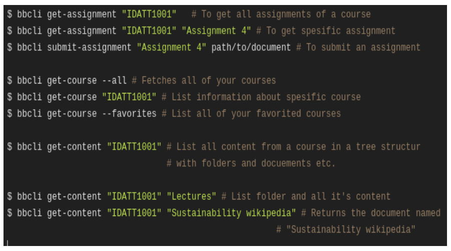
```
$ bbcli assignment "IDATT1001" # To get all assignments of a course
$ bbcli assignment "IDATT1001" "Assignment 4" # To get spesific assignment
$ bbcli assignment --submit "Assignment 4" path/to/document # To submit an assignment

$ bbcli course # Fetches all of your courses
$ bbcli course "IDATT1001" # List information about spesific course
$ bbcli course --favorites # List all of your favorited courses

$ bbcli content "IDATT1001" # List all content from a course in a tree structur
                                # with folders and docuements etc.

$ bbcli content "IDATT1001" "Lectures" # List folder and all it's content
$ bbcli content "IDATT1001" "Sustainability wikipedia" # Returns the document named
                                                      # "Sustainability wikipedia"
```

Figure 4.4: Option 1



```
$ bbcli get-assignment "IDATT1001"   # To get all assignments of a course
$ bbcli get-assignment "IDATT1001" "Assignment 4" # To get spesific assignment
$ bbcli submit-assignment "Assignment 4" path/to/document # To submit an assignment

$ bbcli get-course --all # Fetches all of your courses
$ bbcli get-course "IDATT1001" # List information about spesific course
$ bbcli get-course --favorites # List all of your favorited courses

$ bbcli get-content "IDATT1001" # List all content from a course in a tree structur
                                # with folders and docuements etc.

$ bbcli get-content "IDATT1001" "Lectures" # List folder and all it's content
$ bbcli get-content "IDATT1001" "Sustainability wikipedia" # Returns the document named
                                                           # "Sustainability wikipedia"
```

Figure 4.5: Option 2

Two teachers and nine students participated in the first iteration of user tests. The student participants that lacked experience with using the command-line needed a quick introduction with how the command-line worked. A repeated misconception was that the participants did not understand the use of subcommands. This happened primarily with participants who lacked experience with the command-line. It was noted, however, after realizing the use of the help-command, that the participants picked up on the structure of the CLI much easier. As listed in table 4.1, the majority of the tasks were successful for all student participants.

Student user test 1	
Task	Success rate
Log in	100%
List courses	100%
List announcements from a course	88.9%
Get a specific content based on its ID	100%

Table 4.1: Student user test 1

The teachers that participated noted that the use of the IDs was too cumbersome, specifically when creating content requires both course ID and content ID. The teacher did specify that the task was feasible, but had spent more time than the user test asked already. The satisfaction results

Staff user test 1	
Task	Success rate
Log in	100%
List courses	100%
List announcements from a course	100%
Create an announcement	100%
List contents from a course	100%
Get a specific content based on its ID	100%
List only content folders of a course	100%
Create a content in a course	50%

Table 4.2: Staff user test 1

based on user friendliness were 65% and 76% from teachers and students respectively.

On the second iteration, the content visualization was being tested. All participants managed to execute the tasks, but with the following remarks from the test logger. The participants experienced that the help-flag was very useful, and when they understood how commands, options and arguments were structured, the CLI was very intuitive to use. Again, as in the first iteration, the participants requested allowance of using external IDs (e.g. IDATT2900). Several participants stated that the primary drawback to ease of use is the difficulty in remembering the IDs.

Student user test 2	
Task	Success rate
List all courses	100%
List contents from a course	100%
List folders in a course	100%
List content from a specific folder	100%
List all contents of type ”document”	100%

Table 4.3: Student user test 2

Only moderated user tests were performed on teachers on the second iteration. Similar to the student tests, all tasks were successfully executed. When asked whether the CLI was intuitive or not, the response was that the hierarchy of commands and modules was clear and consistent. In addition, all commands and subcommands have a help-command that explains its arguments, options and commands. Contrary, the lack of a progress bar when listing announcements got the participant wondering whether the program was stuck.

When asked if the CLI is more effective and simple to use than the web interface, the response was positive, considering the possibility to wrap the program and write scripts for certain tasks and reuse them later. In addition, the participant noted that the web interface makes the user click through several pages to perform a specific tasks, which the CLI avoids.

Staff user test 2	
Task	Success rate
List all courses	100%
List announcements from a course	100%
Create an announcement using markdown	100%
List content from a course, folders only	100%
Create a hidden content with a start date for when to make it available for students	100%

Table 4.4: staff user test 2

The satisfaction results based on user friendliness were 90% and 82.5% from teachers and students respectively.

Student user test 3	
Task	Success rate
List all courses	100%
List all assignments from a course	100%
Submit an assignment	100%

Table 4.5: Student user test 3

Staff user test 3	
Task	Success rate
Log in	100%
List all courses	100%
List assignments from a course	100%
Create an assignment	100%
Grade an attempt	100%

Table 4.6: staff user test 3

The third iteration had to be performed as a moderated test due to the need of using a sandbox course to submit assignments. The user was able to execute all tasks successfully. However, the user mentioned that it was ineffective to only grade one assignment attempt at the time. A recommendation from the user was to download all attempts, grade them in a CSV file before uploading them simultaneously.

4.2 Commands and Functionality

Table 4.7 lists all supported commands at time of delivery. The commands are structured into groups, where each group can have a set of subcommands. The main commands are *announcements*, *assignments*, *contents* and *courses*. In addition, *login* and *logout* are singular commands not attached to any group. The modularity is there to ensure support for simple expansion of the tool, in order for contributors to easily modify and expand as they please.

Command/Group	Command/Group	Command	Description	Example usage
	login		Log in user	bb login
	logout		Log out user	bb logout
Announcements	create		Create an announcement	bb announcements create --course _123_1
	delete		Delete an announcement	bb announcements delete --course _123_1 --announcement _111_1
	list		List one or more announcements	bb announcements list --course _123_1
	update		Update an exisiting announcement	bb announcements update --course _123_1 --announcement _111_1
Assignments	attempts	get	Get a specific attempt for an assignment	bb assignments attempts get --course _123_1 --assignment _222_1 --attempt _333_1
		list	List attempts for an assignment	bb assignments attempts list --course _123_1 --assignment _222_1
		submit-draft	Submit assignment draft	bb assignments attempts submit-draft --course _123_1 --assignment _222_1 --attempt _333_1
	create		Create an assignment	bb assignments create --course _123_1 --folder _444_1
	grade		Grade an assignment	bb assignments grade --course _123_1 --assignment _222_1 --attempt _333_1
	list		List all assignments from a course	bb assignments list --course _123_1
	submit		Submit assignment attempt	bb assignments submit --course _123_1 --assignment _222_1
Contents	create			
		assignment	Create an assignment	bb contents create assignment --course _123_1 --folder _444_1
		attachment	Add attachment to content	bb contents create attachment --course _123_1 --content _555_1
		course-link	Create course link content	bb contents create course-link --course _123_1 --folder _444_1 --target _666_1 TITLE
		document	Create document content	bb contents create document –course _123_1 --folder _444_1 TITLE
		file	Create file content	bb contents create file --course _123_1 --folder _444_1 TITLE PATH
		folder	Create folder	bb contents create folder --course _123_1 TITLE
		web-link	Create web-link	bb contents create web-link --course _123_1 --folder _444_1 TITLE URL
	delete		Delete content	bb contents delete --course _123_1 --content _555_1
	get		Get a specific content	bb contents get --course _123_1 --content _555_1
	list		List contents	bb contents list --course _123_1
	update		Update content	bb contents update –course _123_1 --content _555_1
Courses	list		List courses	bb courses list

Table 4.7: Supported commands

4.2.1 Functional requirements

Following is a list of all functionality at time of delivery, supported by the commands listed in table 4.7.

- The user is able to login to Blackboard using the two-factor authentication.
- The user can logout.
- The user can list courses that they are currently taking or that they have taken.
- The user can list all announcements across several courses, as well as within a specific course.
- Announcements can be created, updated or deleted for a specific course.
- All contents can be listed within a course.
- Specific folders can be listed.
- Contents such as documents, assignments, folders, external links, forum links and blank pages can be listed within a course.
- Assignment, attachment, document, file, folder and web-link can be created, updated and deleted.
- Documents and files can be retrieved and opened.
- Web-links, external links and course links can be opened.
- Blank pages can be opened.
- The user can get an assignment and download the attachments.
- Assignments can be graded.
- Assignment attempts can be listed.
- A specific assignment attempt can be listed.
- Both assignments and draft assignments can be submitted.
- Assignments can be updated.

4.2.2 Arguments and Options

Each command supports either positional arguments or options or both. Arguments do not require flags as options do. For example in the command

```
$ bb contents create document [-c <COURSE_ID>] [-f <FOLDER_ID>]
[--start-date <START_DATE>] [--end-date <END_DATE>] [--reviewable]
[--hide-content] TITLE [ATTACHMENTS]
```

TITLE is a positional argument and --start-date, --end-date, --reviewable and --hide-content are optional. For easier usability, -c (--course) and -f (--folder) are also required, despite technically being options.

4.2.3 Help Snippets

Each command contains a --help option which lists a concise description of the command and its arguments and options, as can be seen in example snippet 4.1.

```
$ bb contents create document --help
Usage: bb contents create document [OPTIONS] TITLE [ATTACHMENTS]...
```

```
    Creates a document content, optionally with file attachments

  Options:
    -c, --course TEXT     COURSE ID, of the course you want to create content.
                          [required]
    -f, --folder TEXT     FOLDER ID, of the folder you want to create content in.
                          [required]
    --end-date TEXT       When to make content unavailable. Format: DD/MM/YY
                          HH:MM:SS
    --start-date TEXT     When to make content available. Format: DD/MM/YY
                          HH:MM:SS
    -r, --reviewable      Make content reviewable
    -h, --hide-content    Hide contents for students
    --help                Show this message and exit.
```

Code listing 4.1: Help snippet

4.2.4 Input

When dealing with text input for HTML bodies the editor from the user's $EDITOR environment variable is used, e.g., when creating announcements, creating assignments etc. The different input options are plain text, Markdown or HTML syntax.

4.2.5 Output

The default output is shown in code listing 4.2, where essential information is printed in a tabular format.

```
$ bb announcements list -c _33050_1 -a _394612_1

Id:                _394612_1
 Title:            Announcement
Date:              2022-05-12

This is an announcement.
```

Code listing 4.2: Default output

Optionally, the --json flag can be used to print the entire response message in json format as shown in code listing 4.3.

```
$ bb announcements list -c _33050_1 -a _394612_1 --json
{
  "id": "_394612_1",
  "title": "Announcement",
  "body": "<p>This is an announcement.</p>",
  "creator": "_111522_1",
  "draft": false,
  "availability": {
    "duration": {
      "type": "Restricted",
      "start": "2022-05-12T16:12:26.238Z",
      "end": null
    }
  },
  "created": "2022-05-12T16:12:26.240Z",
  "modified": "2022-05-12T16:12:26.277Z",
  "position": 1
}
```

Code listing 4.3: Json output

4.2.6 Authorization

To authorize a user, the CLI executes the same HTTP POST requests as the Blackboard web interface until the cookies and headers needed to communicate with the Blackboard Learn REST API are retrieved. However, for these requests to be successful, each request needs certain cookies and headers and one or more hidden values from the HTML response body of the previous request.

These hidden values are required in the next HTTP POST request for them to give a success response and either a new hidden value or the Blackboard session cookie which is the end goal of the login process.

4.2.6.1 Simple login

A user who has not activated 2-step authentication only need to enter username and password. The CLI can skip a step in the login process and only send four HTTP POST requests.

4.2.6.2 Login with 2-step authentication

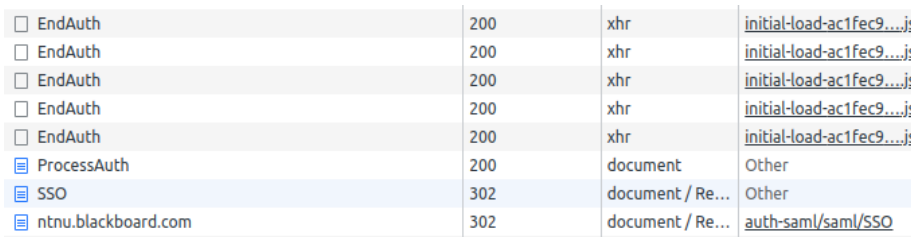
If the user has activated 2-step authentication, a few extra steps is required in the login process. First, the CLI tries to find the default authentication method in one of the HTML responses. However, if no default option is found, the user is prompted with 3 alternatives:

1. PhoneAppNotification
2. PhoneAppOTP
3. OneWaySMS

PhoneAppNotification: a series of HTTP POST requests are sent to check whether the user has accepted the login attempt in the Microsoft Authentication app. These requests can be seen in figure 4.6 and have the name ”EndAuth”.

PhoneAppOTP: the CLI waits for the user to enter the code from Microsoft Authentication app, which is included in the next HTTP POST request to verify if the code was correct.

OneWaySMS: a code is sent to the users phone which is then included in the next HTTP POST request to verify if it is valid.



EndAuth	200	xhr	initial-load-ac1fec9....js
EndAuth	200	xhr	initial-load-ac1fec9....js
EndAuth	200	xhr	initial-load-ac1fec9....js
EndAuth	200	xhr	initial-load-ac1fec9....js
EndAuth	200	xhr	initial-load-ac1fec9....js
ProcessAuth	200	document	Other
SSO	302	document / Re...	Other
ntnu.blackboard.com	302	document / Re...	auth-saml/saml/SSO

Figure 4.6: EndAuth POST requests being sent repeatedly until the user accepts.

4.3 Non-functional Requirements

4.3.1 Usability

Initially, the root command was *bbcli*, but results from the first user survey showed that the most preferred root command was *bb* followed by subcommands with different options and arguments. The syntax that was chosen in the end was a slightly modified version of option 1 in figure 4.4. Most users preferred having less commands, and more options for each command. For example, bb contents list was preferred over bb list-contents.

4.3.2 Reliability

Error handling was implemented to give the user a meaningful feedback if something unexpected would occur, instead of the standard json response provided by the REST API.

4.3.3 Performance

Across the user tests, it was emphasized the inconvenience of having to go in and out of folders to fetch content. The solution was a visual representation of the folder structure as a tree. To retrieve all the content, multiple requests to the Blackboard API was needed. For each folder, there was a need to check content within that folder, which resulted in a complexity of $O(n^2)$ (appendix A.1). When all the content was retrieved, a depth-first search (DFS) was used to traverse the tree. DFS has a complexity of $O(n)$, because all the n-nodes needs to be traversed (appendix A.2). This gives a total complexity of $O(n^2)+O(n)$, and could be time consuming due to the fact that some courses are large and packed with content. By separating the execution, the run-time was was in certain cases remarkably reduced. Table 4.8 shows the time to retrieve and print contents with and without threads.

Course Name	Total contents	Total threads	W/O threads	W/ threads
IFYT1001 Fysikk (2021 VÅR)	967	24	17.085 s	5.576 s
IDATT2105 Full-stack applikasjonsutvikling (2021 VÅR)	53	5	1.521 s	1.237 s
IDATT2101 Algoritmer og datastrukturer (2020 HØST)	37	5	0.814 s	0.607 s
IDATT1001 Programmering 1 (2019 HØST)	107	8	2.428 s	1.562 s
INGT2300 Ingeniørfaglig systemtenkning (2022 VÅR)	115	7	2.747 s	1.868 s

Table 4.8: Run-time to list contents in seconds

4.3.4 Supportability

The CLI is supported on both Mac, Linux and Windows with only one command required for installation. The only prerequisites is that the user has Python installed.

At time of delivery, the product is being made publicly accessible, i.e., open source and is designed to be easily modified so that other developers can add or modify functionality to their preference.

4.3.5 Security

The username is stored in an environment variable and the password is not stored anywhere.

4.4 Efficiency

The results from the efficiency experiment are listed in table 4.9.

Key presses

The amount of key presses needed to execute simple tasks were usually not noteworthy more than the amount of commands required for the same task. For more complex tasks, however, the amount of key presses were significantly higher.

Number of Commands

The number of commands needed to execute a task or set of tasks were never more than 3 commands. The length of the commands varied some, but no command was inconveniently long.

HTTP Requests

The amount of HTTP requests that was sent during the execution of the task varied greatly between the CLI and web UI. The web UI uses for some tasks more than 200 times more HTTP requests than the CLI.

Loading time

The total loading time was for almost all tasks significantly lower on the CLI.

Time To Complete

For simple tasks, time to complete is almost equal for both systems. For more complex tasks, the TTC on the CLI was significantly lower.

	Key presses	Commands	HTTP Requests		Loading time		Time to Complete	
	Web UI	CLI	Web UI	CLI	Web UI	CLI	Web UI	CLI
Login	2	1	260	10	6.43 s	4.04 s	16.05 s	12.83 s
List courses	1	1	36	29	1.56 s	2.17 s	2.62 s	5.01 s
List course announcements	3	2	348	28	5.67	3.60 s	8.39 s	11.91 s
Create an announcement	8-12	1	1210	28	8.29 s	0.5 s	20.52	19.02
Get specific content item	5	3	744	47	5.83 s	6.3 s	12.57	31.35
Create a document	6	2	1446	37	7.69 s	5.31 s	18.48 s	29.84 s
Find all assignment due dates from a course	40	2	7295	29	28.47 s	3.85 s	69.06 s	12.5 s
List all documents from a course	23	2	7307	45	38.17 s	2.89 s	53.13 s	16.72 s
List all announcements from courses from the last two semesters)	13	1	1368	1	22.35 s	4.21 s	29.37 s	7.06 s

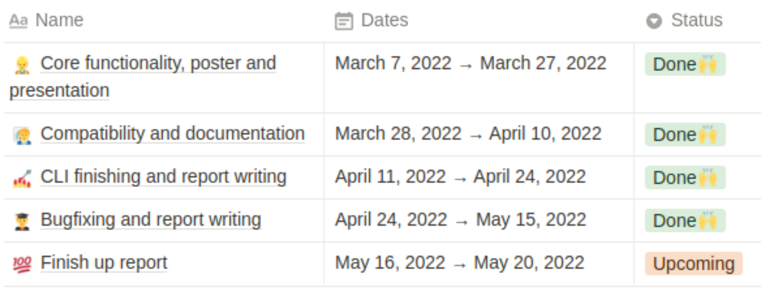
Table 4.9: Comparison of efficiency between the CLI and the web UI

4.5 Administrative Results

This section contains an overview over what was gained from the administrative tools that were used to keep track of progress, time management, and tasks in general.

4.5.0.1 Progress plan

The roadmap was used as a progress plan, with goals regarding the development of the product, documentation and report writing, as shown in figure 4.7 and 4.8. Each milestone and its goals works similar to sprints and and backlogs. The roadmap put everyone on the same page regarding process, defined a clear path from start to delivery, and aid in prioritizing and re-prioritizing tasks.



Name	Dates	Status
Core functionality, poster and presentation	March 7, 2022 → March 27, 2022	Done 🙌
Compatibility and documentation	March 28, 2022 → April 10, 2022	Done 🙌
CLI finishing and report writing	April 11, 2022 → April 24, 2022	Done 🙌
Bugfixing and report writing	April 24, 2022 → May 15, 2022	Done 🙌
💯 Finish up report	May 16, 2022 → May 20, 2022	Upcoming

Figure 4.7: Milestones in roadmap.

4.5.1 Time Management

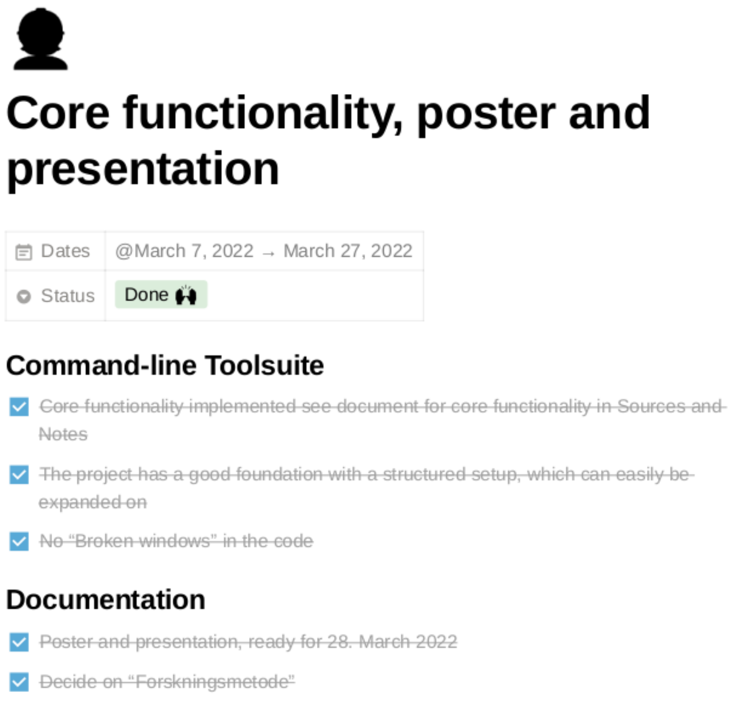
A status report was created at the end of each week to keep track over hours worked and what was done, shown in table 4.9.

4.5.2 Agile Process Documentation

The agile process of this project was primarily built on the roadmap with milestones similar to iterations. Each iteration consisted of gaining feedback from users, review feedback, and apply changes which contributed in a more versatile CLI. In addition, other process tools like retrospectives, daily stand-ups and project status reviews gave structure and flow in the development process.

4.5.2.1 Backlog

As mentioned above, the milestones, shown in figure 4.10 had goals categorized in development of the product, documentation and report writing. All goals regarding development of the product were again split up into even more detailed issues in GitLab, as seen in figure 4.11. To avoid misinterpretation of the issues, the aim was to create them as small and concrete as possible. All issues were linked to a milestone in GitLab which



Core functionality, poster and presentation

Dates	@March 7, 2022 → March 27, 2022
Status	Done

Command-line Toolsuite

- ✓ ~~Core functionality implemented see document for core functionality in Sources and Notes~~
- ✓ ~~The project has a good foundation with a structured setup, which can easily be expanded on~~
- ✓ ~~No "Broken windows" in the code~~

Documentation

- ✓ ~~Poster and presentation, ready for 28. March 2022~~
- ✓ ~~Decide on "Forskningsmetode"~~

Figure 4.8: Milestone goals.

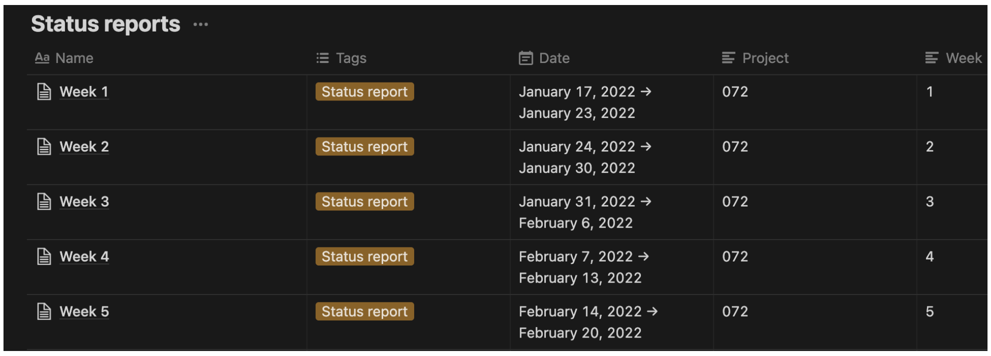
again reflected the goals in the roadmap concerning development of the product.

4.5.2.2 Retrospectives

At the end of each milestone, a retrospective was performed to reflect on the completed iteration. This was to find out what worked well, what did not work well, and possible areas of improvement. Some subjects that was brought up during the retrospectives were working without distractions, more frequent updates about each others work, and more consistent work-hours.

4.5.2.3 Daily stand-up meetings

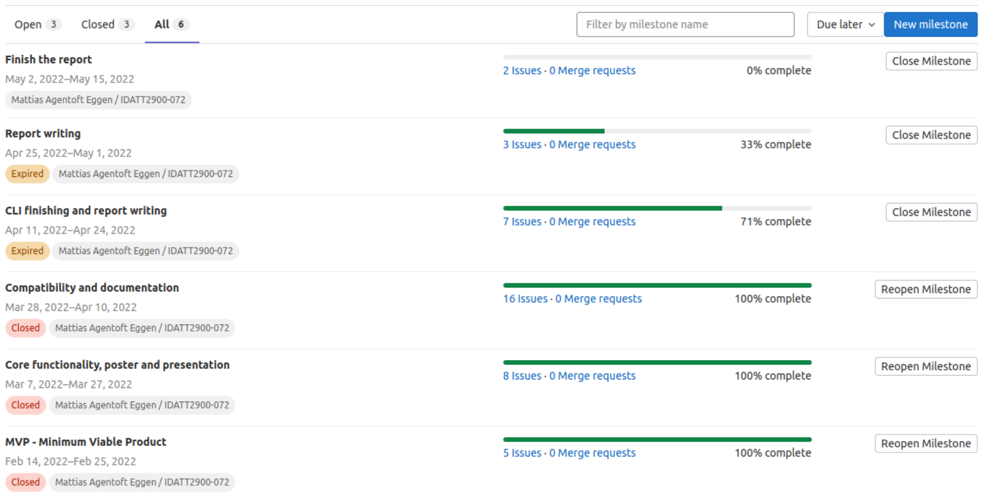
The daily stand-up meetings consisted of every member standing up, explaining what they did the previous day and what they planned to work on that day, which was an efficient way of keeping all team members on the same page. In addition, it prevented misunderstandings like two members working on the same task and improving the communication in the group.



Status reports ···

Name	Tags	Date	Project	Week
Week 1	Status report	January 17, 2022 → January 23, 2022	072	1
Week 2	Status report	January 24, 2022 → January 30, 2022	072	2
Week 3	Status report	January 31, 2022 → February 6, 2022	072	3
Week 4	Status report	February 7, 2022 → February 13, 2022	072	4
Week 5	Status report	February 14, 2022 → February 20, 2022	072	5

Figure 4.9: List of the status reports.



Open 3 Closed 3 All 6

Filter by milestone name | Due later | New milestone

Finish the report May 2, 2022–May 15, 2022 Mattias Agentoft Eggen / IDATT2900-072	2 Issues · 0 Merge requests	0% complete	Close Milestone
Report writing Apr 25, 2022–May 1, 2022 Expired Mattias Agentoft Eggen / IDATT2900-072	3 Issues · 0 Merge requests	33% complete	Close Milestone
CLI finishing and report writing Apr 11, 2022–Apr 24, 2022 Expired Mattias Agentoft Eggen / IDATT2900-072	7 Issues · 0 Merge requests	71% complete	Close Milestone
Compatibility and documentation Mar 28, 2022–Apr 10, 2022 Closed Mattias Agentoft Eggen / IDATT2900-072	16 Issues · 0 Merge requests	100% complete	Reopen Milestone
Core functionality, poster and presentation Mar 7, 2022–Mar 27, 2022 Closed Mattias Agentoft Eggen / IDATT2900-072	8 Issues · 0 Merge requests	100% complete	Reopen Milestone
MVP - Minimum Viable Product Feb 14, 2022–Feb 25, 2022 Closed Mattias Agentoft Eggen / IDATT2900-072	5 Issues · 0 Merge requests	100% complete	Reopen Milestone

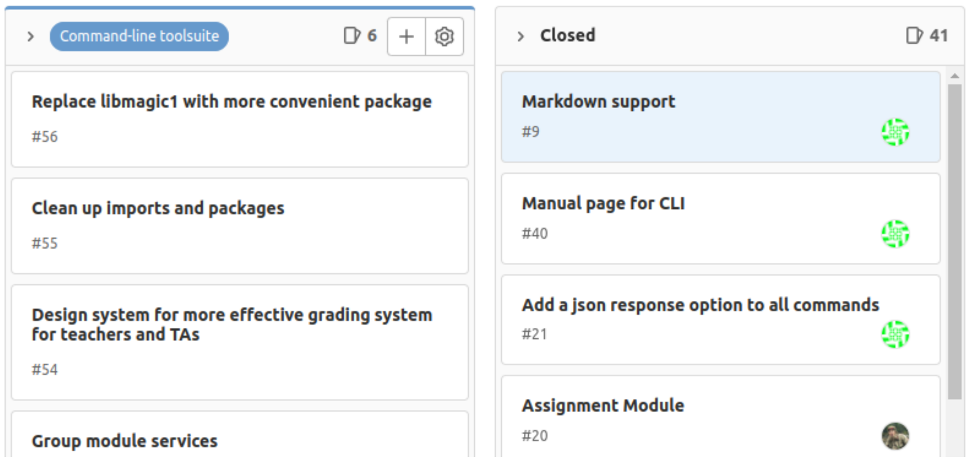
Figure 4.10: Milestones in GitLab connected to related issues. In sync with roadmap milestones.

4.5.2.4 Project status reviews

Each Friday, a project status review was held to reflect upon the progress, and plan the upcoming week. This was an additional tool to make sure everyone was on the same page and made it easier to prioritize tasks. The difference between the project status reviews and retrospectives is that the former purely focuses on progression in the project, not work methodology. A status review could also take place in the middle of the week or whenever it was necessary to take a step back and gain an overview of the overall status of the project.

4.5.2.5 Unit testing

To easily detect whether any new implementations in the code introduced new errors, unit tests were created for all service methods and could be run by executing the command `pytest`. This made it less complicated to debug the code if an error occurred.


Command-line toolsuite 6

Replace libmagic1 with more convenient package
#56

Clean up imports and packages
#55

Design system for more effective grading system for teachers and TAs
#54

Group module services

Closed 41

Markdown support
#9

Manual page for CLI
#40

Add a json response option to all commands
#21

Assignment Module
#20

Figure 4.11: Issue board in GitLab.

4.5.2.6 Continuous Integration and Deployment

The unit tests were also used when implementing continuous integration and deployment. Every time a branch was pushed to remote main branch, all unit tests were run to make certain no unknown bugs were present in the code. The project was automatically uploaded to PyPi as a python package, but only if all unit tests were successfully passed.

Chapter 5

Discussion

5.1 Evaluation of the Research Questions

The researched questions will be evaluated based on the background theory in chapter 2 in relation to the results produced from chapter 4.

RQ1. How can efficiency be measured when comparing the CLI tool to the web UI?

When comparing the CLI to the web UI, convenience and time spent on a task are metrics to measure the efficiency.

Convenience in this context more specifically means the users satisfaction, and their prior knowledge with using the UI. A UI can be regarded as convenient if tasks can be performed with the most amount of efficiency and the least amount of effort. Some may be biased to the web UI because they have been using it for a longer period of time. Therefore, they might find it more convenient and efficient to use.

Another way of measuring efficiency between the CLI and the web UI is by looking at the time it takes to complete tasks. For simple tasks, it can be difficult to see any real difference in TTC, a consequence of the tasks being small. However, it is easier to see clear differences in TTC when performing a large number of tasks consecutively, e.g. when retrieving large sets of data from different locations. It is often difficult to compare the CLI to the web UI since some of the functionality is not comparable. For example, it is not possible to list all assignments at once in the web UI, but the CLI does it in a few seconds.

Related to time spent is loading time. In other words the time spent waiting for pages to load or a response from a command. In addition, measuring the number of HTTP requests sent during a task seems to have a positive correlation with the loading time. However, this is not absolutely accurate because two HTTP requests do not necessarily take the same amount of time to load. This depends on the size of the payload.

It is also important to address how experience affects efficiency. A user does not require the same amount of experience to comfortably use the web UI compared to the CLI. As stated in section 2.2.1, the GUI's visual representation makes it easy for novice users to navigate and execute tasks. However, as users get more experienced, they will usually prefer the command-line as it allows them to maintain more control and increase efficiency.

In the survey conducted by Omanson and others, it was found that keyboard interactions are more efficient. However, it is important to emphasize that preferences and different use cases play an important role. If keyboard interactions are favored, the user might find the CLI to be more convenient as they can type commands fast, and use arrow keys to retrieve previously used commands. On the other hand, if the user is quick with a mouse pointer, interactions with a web UI can be beneficial.

Measuring efficiency can be difficult, because it is highly dependent on personal preference and prior experience. Additionally, the results will highly differ from person to person depending on how they prefer interacting with the UI.

RQ2. What are the advantages and disadvantages of the CLI in comparison to the web UI?

For simple tasks such as listing course announcements, the CLI and web UI are quite similar in TTC. That being said, it is clear from the results in table 4.9 that when retrieving large amounts of information the CLI is noticeably faster than the Web UI. These are tasks where the Web UI does not have any good alternative functionality, which causes the user to navigate back and fourth exceedingly.

Feedback from the first survey showed interest in including custom functionality that addresses these issues. Among the feedback was the disadvantage of having to go in and out of folders to retrieve content. The CLI offers a command to list all the contents in a course to get an overview of locations and content IDs. Not to mention that the command offers an option to list only folders, documents, assignments, external links and so fourth. A drawback, however, is that the user has to find the course ID first, which results in two commands in total.

Additionally, the CLI offers functionality to list all the assignments of a course. This is significantly more convenient compared to the web UI where the user needs to search and interpret what are assignments or not. In table 4.9, results show that it takes five times longer to get the assignments using the web UI compared to the CLI.

It is important to note that during the efficiency experiment, the contents location were known, which resulted in minimal search time when navigating. Had the IDs of the courses and contents been known, the TTC would be reduced noticeably on the CLI. As discussed in the evaluation of RQ1, it is difficult to compare remembrance since it is easier to memorize locations in a web UI than IDs in a CLI. However, a number of participants mentioned that Blackboard's web UI is not user friendly when it comes to navigation. Courses rarely have the same structure, which results in having to spend a lot of time navigating to find the desired content. The CLI fixes this issue by having identical commands for every course. Additionally, course IDs should not be difficult to remember since a student rarely uses more than four course IDs each semester.

In addition to efficiency, the CLI only shows the necessary output and avoids distracting visuals that increases loading times. On the other hand, the CLI has an arguable disadvantage regarding the use of IDs. The user tests showed that among those who preferred the Web UI over the CLI, the biggest reason was that the IDs were too cryptic to remember and it was cumbersome to use several different IDs for one command. It would be easier to use the course code and title of contents and announcements, however these are not guaranteed to be unique and therefore the format of the IDs still remains a weakness in the CLI. Despite this fact, almost all participants were successful in getting the content based on IDs.

Illustrating Git as an example conducted by Olofsson and Hulstrand in section 2.3, it was stated that a CLI was most preferred for Git users, but that the GUI was an alternative for less proficient users. It was also revealed that users would prefer using a CLI as their experience increased. This substantiates the results gained from the user tests, namely that a CLI is advanced to learn but effective once you get a certain level of proficiency. This was a consistent finding across the moderated user tests, were it was observed that the user became more comfortable after having read the help messages and executed some commands. However, it is important to emphasize that there are no results that will tell how the proficiency will look like in the long run. At the same time, it is difficult to tell whether the potential proficiency gained will be compared with the high cognitive workload required of using the CLI. Different commands with their respective arguments, options and flags need to be memorized.

If the users become familiar and experienced with the CLI, there is also a possibility to expand the usage of the CLI to be more advanced. For example it is possible to create scripts that uses the CLI to automate tasks. An advantage with creating scripts to automatically execute commands is that the cryptic IDs will not be as big of an issue anymore since the IDs will be written in the script and saved.

RQ3. How can agile principles and practices contribute in developing a versatile CLI to increase user experience and efficiency?

The agile mindset provides, as mentioned in section 2.5.1, values and principles on how to respond to change. These values and principles were followed thoroughly during the project. Change has always been welcome, and the design and motivation behind the product has changed several times. A key motivation has been to keep the design simple, in order to encourage quick and easy expansion of functionality.

As described in chapter 2 and chapter 4, several agile practices were implemented in the development process. However, looking back at these practices, it is difficult to see how they actually contributed in increasing the versatility. No metrics were taken during development to see for example how unit testing contributed to the research question. In fact, for many practices this would not make any sense, since their only purpose is to assist the development of the product itself, not its functionality. Therefore, it is pertinent to address that many agile practices are solely meant to aid the development.

However, a practice that clearly contributed was the user tests. Some requested markdown as input, while others desired HTML syntax, as stated in 4.2.4. Additionally, it was discovered during the user tests that there was a desire for different output formats, such as tabular and json. Therefore, it is safe to say that user tests are essential in increasing versatility.

All the practices mentioned earlier did encourage a constant pace during development, which resulted in quickly noticing issues and responding to change. Consequently, it can be said that the practices indirectly contributed in including more users and increasing efficiency and effectiveness of the CLI.

5.2 Critical Analysis

In order to evaluate the project as a whole, it is appropriate to provide a critical analysis of the results.

5.2.1 Limitations of Data

A number of agile principles and practices were implemented in the development process. However, not enough data about the use of agile principles and practices were gathered to adequately support the evaluation of RQ3. The only way of evaluating the research question was by reflecting around the experiences gathered throughout the project. It is difficult to find how agile principles and practices can contribute in development when the only metric is prior experiences. Had different agile practices been tested against non-agile practices during the development process, more data could be used to evaluate the research question.

The user tests also provided unsatisfactory data when used to evaluate RQ3. The research question asks how agile principles and practices can contribute in increasing efficiency and effectiveness. However, the tasks performed on each user test only focuses on whether the task was successful or not. Had other metrics been taken to measure efficiency and effectiveness, RQ3 could have been evaluated more thoroughly.

The data gathered from surveys were partly unsatisfactory due to the fact that too few metrics were taken. In addition, more contributors from a wider range of field of studies, especially more teachers, would give better insight in how the CLI should be implemented. Towards the end of the project, it was discovered that teachers would benefit the most from using the CLI. That is because teachers need to perform tasks that are a lot more complex than what the students need. These are tasks that can be automated or made a lot more efficient to execute. Had interviews been conducted or more surveys with more metrics been made in the beginning, more of this functionality could have been implemented.

5.2.2 Applicability of the CLI

Long-term Use

The longest lasting user tests lasted around 20 minutes, which limits to show how users would handle the tool in a real working environment. The user tests fail to give insight in how effectiveness and efficiency is affected after long-term use, as an effect of learning and customization by the user.

Whilst the long-term use of the CLI has not been tested, it is reasonable to assume that it will stay functional for a long time. The tool was programmed against Blackboard's own REST API, which is more reliable than web scraping directly from the website. Rockhack's CLI *bb*, which was mentioned in the background theory chapter, does not use the REST API, which greatly decreases the reliability of the tool. Our tool almost solely relies on the REST API, which theoretically increases it's lifetime compared to Rockhack's tool.

Expansion and Customization

The CLI is designed to be easily expanded and customized by contributors. The open source project includes issues for future work and detailed explanations on how expansion can be done. The different options for input and output give the user more freedom to write scripts against the tool.

5.2.3 Team Evaluation

It was important to get the team organized with roles, goals and process. The responsibilities for each roles were clear and consistent throughout the project. The team chemistry was good and conflicts were handled quickly and efficiently.

Chapter 6

Conclusion

6.1 RQ1

Measuring the efficiency when comparing the CLI tool to the web UI can be done by evaluating the convenience of use, time to complete a task and loading time. However, meanwhile these measuring methods may give an indication of how efficient the two alternatives are, users' personal preferences and prior experience will affect the results.

6.2 RQ2

The CLI has implemented functionalities that serve as a supplement to the web UI's weaknesses. By removing the necessity of navigation, the efficiency is increased for certain tasks, like listing contents and assignments. Another advantage is the potential for contributors to expand the functionality of the CLI to better suit their needs, or include the CLI in their own scripts to automate tasks.

To the contrary, the use of cryptic IDs are cumbersome. Some tasks require several commands to be executed, as multiple IDs are needed. Especially for less proficient CLI users, this can lead to a high cognitive workload.

6.3 RQ3

It was experienced that the agile mindset and principles encouraged in developing a versatile CLI.

Critical functionality was discovered through close cooperation with a versatile user base. Of the used practices, it was evident that user tests were the most crucial in increasing versatility, efficiency and effectiveness of the CLI.

Other agile practices gave the impression of utility, but there are no quantitative data and metrics that support this. They had an indirect impact in improving the development, which facilitated better opportunities for creating a versatile product.

6.4 Key Findings

This thesis has explored some of the challenges entailed in the development of a CLI as an alternative to an existing web UI. Following are the challenges that were found the most significant:

- Measuring efficiency of use in different UIs often depends on personal preference and experience.
- Quantifying how agile principles and practices can contribute in the development of an efficient and effective alternative to an already existing system.
- Tailoring the system to satisfy the utmost user preferences.

6.5 Societal Impact

Arguably, the work does not have any significant impact on the society. It however opens up for better learning opportunities, as certain tasks can be executed more efficiently. Users can have a better workflow and save time when performing day to day tasks. Additionally, it includes more versatile user base because of people having different preferences to what UI they want to use.

Focusing on the environmental impact, results show that the CLI sends less HTTP requests than the web UI. It is hard to say whether this impact has any significance at all, since size of the payloads are not known. However, cloud computing is costly in general, and therefore the motivation for the CLI is reduce the carbon footprint as opposed to the web UI. On the other hand, these HTTP requests are in certain cases drastically reduced, which arguably leads to both better energy cost savings and climate footprint. It is safe to say that the project does not have any major impact on other sustainability assessments including gender differences, disabled people, underrepresented minorities and so fourth. The CLI can be used efficiently no matter what gender you are. People with disabilities may find it more difficult to use this tool as results show that the CLI is advanced to learn and has small amounts of visual objects. It is available for all people independent of birth place and what religion is supported.

6.6 Future Work

The CLI has not included all the functionality provided by the Blackboard Learn REST API. New features can be requested on the GitHub page (Blackboard-LMS-CLI) by contributors. Even though the team used GitLab as the version-control system during development, the project was eventually transferred to GitHub to reach out to more people. Additionally, the following issues are facilitated through the open source project.

6.6.1 Groups

The groups module should contain functionality for listing, creating, updating and deleting groups or whole sets of groups. Importing groups and group sets should also be supported in this module.

In addition, a command for listing users based on different filters is desired. Some examples of filters are: all users that are not in a group, all users that are in at least one group, all users from a specific groups.

For students, a command to register for a group is needed.

6.6.2 Grading

In the assignments module, it would be convenient to have a command for downloading all assignment attempts in a CSV file with an open score field. After reviewing and grading the attempts, the CSV can be uploaded to automatically update the grade of all the assignments simultaneously.

6.6.3 Stdin

A --stdin flag should be added as an option for all commands that require text input, to allow for standard input instead of input from the editor. This is to increase flexibility when creating custom scripts that use the CLI.

Bibliography

[1] A. H. Jørgensen and B. A. Myers, 'User interface history,' *CHI '08 extended abstracts on Human factors in computing systems*, pp. 2415–2418, 2008.

[2] P. Verma, 'Gracoli: A graphical command line user interface,' *CHI EA '13: CHI '13 Extended Abstracts on Human Factors in Computing Systems*, pp. 3143–3146, 2013. [Online]. Available: https://doi.org/10.1145/2468356.2479631.

[3] R. Olofsson and S. Hultstrand, 'Git - cli or gui,' 2015.

[4] S. A. DAVID. 'A critical understanding of learning management system.' (), [Online]. Available: https://www.academia.edu/3681177/A_Critical_Understanding_of_Learning_Management_System. (accessed: 25.04.2022).

[5] C. C. Chang, 'Exploring the determinants of e-learning systems continuance intention in academic libraries, library management,' pp. 40–55, 2014. [Online]. Available: https://doi.org/10.1108/01435121311298261, (accessed: 25.04.2022).

[6] A. Tella, 'Reliability and factor analysis of a blackboard course management system success: A scale development and validation in an educational context,' pp. 50–80, 2011. [Online]. Available: https://doi.org/10.28945/1368, (accessed: 26.04.2022).

[7] D. Bouhnik and T. Marcus, 'Interaction in distance-learning courses,' *Journal of the American Society for Information Science and Technology*, pp. 299–305, 2005. [Online]. Available: https://doi.org/10.1002/asi.20277, (accessed: 25.04.2022).

[8] B. L. D. E Jane Servonsky W Lawrence Daniels, 'Evaluation of blackboards as a platform for distance education delivery,' *The ABNF Journal*, pp. 132–135, 2005, (accessed: 06.05.2022).

[9] F. Churchville. 'User interface (ui).' (), [Online]. Available: https://www.techtarget.com/searchapparchitecture/definition/user-interface-UI. (accessed: 10.05.2022).

[10] 'Gui (graphical user interface).' (), [Online]. Available: https://www.gartner.com/en/information-technology/glossary/gui-graphical-user-interface. (accessed: 10.03.2022).

[11] R. B. Cabot, 'Re-imagining the command line user experience for problem solving,' pp. 20–30, 2017, (accessed: 09.05.2022).

[12] D. A. N. Edwin L. Hutchins James D. Hollan, 'Direct manipulation interfaces,' 1985. [Online]. Available: http://www-ihm.lri.fr/~mbl/ENS/FONDIHM/2013/papers/Hutchins-HCI-85.pdf, (accessed: 29.03.2022).

[13] B. Shneiderman, 'Direct manipulation: A step beyond programming languages,' 1983. [Online]. Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1654471, (accessed: 29.03.2022).

[14] J. Ellis. 'Command line interface.' (), [Online]. Available: https://www.comms-express.com/infozone/article/command-line-interface/. (accessed: 14.03.2022).

[15] 'Click documentation.' (), [Online]. Available: https://click.palletsprojects.com/en/8.1.x/#documentation. (accessed: 16.02.2022).

[16] G. Nguyen. 'Make cli apps great again.' (), [Online]. Available: https://codeburst.io/make-cli-apps-great-again-c93221422cdb. (accessed: 16.02.2022).

[17] J. Preece, *Human-Computer Interaction: Concepts And Design*. Addison Wesley, 1994, (accessed: 09.04.2022).

[18] R. C. Omanson, C. S. Miller, E. Young and D. Schwantes, 'Comparison of mouse and keyboard efficiency,' [Online]. Available: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.1087.6916&rep=rep1&type=pdf, (accessed: 30.03.2022).

[19] J. O. Wobbrock, H. H. Aung, B. Rothrock and B. A. Myers, 'Maximizing the guessability of symbolic input,' [Online]. Available: https://dl.acm.org/doi/pdf/10.1145/1056808.1057043, (accessed: 09.04.2022).

[20] C. Coyier. 'Graphical user interfaces for git.' (2019), [Online]. Available: https://css-tricks.com/graphical-user-interfaces-for-git/. (accessed: 11.03.2022).

[21] 'What is git and why should you use it?' (), [Online]. Available: https://www.nobledesktop.com/learn/git/what-is-git. (accessed: 06.03.2022).

[22] RocHack, *Bb*, https://github.com/RocHack/bb, 2015.

[23] M. REHKOPF. 'Agile 101.' (), [Online]. Available: https://www.agilealliance.org/agile101/. (accessed: 22.02.2022).

[24] V. E. Jyothi and K. N. Rao, 'Effective implementation of agile practices ingenious and organized theoretical framework,' *International Journal of Advanced Computer Science and Applications*, pp. 41–48, [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.674.6165&rep=rep1&type=pdf#page=45, (accessed: 28.02.2022).

[25] K. Beck, M. Beedle, A. van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, J. Kern, B. Marick, R. C. Martin, S. Mellor, K. Schwaber, J. Sutherland and D. Thomas. 'Manifesto for agile software development.' (), [Online]. Available: http://agilemanifesto.org/. (accessed: 28.02.2022).

[26] A. S. Campanelli and F. S. Parreiras. 'Agile methods tailoring – a systematic literature review.' (), [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0164121215001843. (accessed: 26.04.2022).

[27] J. Yip. 'It's not just standing up: Patterns for daily standup meetings.' (2016), [Online]. Available: https://www.martinfowler.com/articles/itsNotJustStandingUp.html. (accessed: 12.03.2022).

[28] J. Link, 'The role of unit tests in the software process,' 2003. [Online]. Available: https://www.sciencedirect.com/topics/computer-science/incremental-development, (accessed: 02.04.2022).

[29] 'Iteration planning.' (), [Online]. Available: https://www.scaledagileframework.com/iteration-planning/. (accessed: 02.04.2022).

[30] 'Agile retrospective.' (), [Online]. Available: https://www.techtarget.com/searchsoftwarequality/definition/Agile-retrospective. (accessed: 02.04.2022).

[31] 'Retrospective 101.' (), [Online]. Available: https://www.retrium.com/ultimate-guide-to-agile-retrospectives/retrospectives-101. (accessed: 06.04.2022).

[32] J. Hofmann, 'Unit testing in agile web projects,' 10th May 2017. [Online]. Available: https://medium.com/aperto-an-ibm-company/unit-testing-in-agile-web-projects-4db5547a733b, (accessed: 10.04.2022).

[33] 'Moderated testing 101.' (), [Online]. Available: https://www.usertesting.com/blog/moderated-testing-101. (accessed: 06.04.2022).

[34] J. DeRome. 'Moderated vs. unmoderated usability testing: The pros and cons.' (), [Online]. Available: https://www.usertesting.com/blog/moderated-vs-unmoderated-usability-testing. (accessed: 06.04.2022).

[35] I. Sacolick, 'What is ci/cd? continuous integration and continuous delivery explained,' 15th Apr. 2022. [Online]. Available: https://www.infoworld.com/article/3271126/what-is-cicd-continuous-integration-and-continuous-delivery-explained.html, (accessed: 20.04.2022).

[36] S. Pittet. 'What are the differences between continuous integration, continuous delivery, and continuous deployment (ci/cd)?' (), [Online]. Available: https://www.atlassian.com/continuous-delivery/principles/continuous-integration-vs-delivery-vs-deployment. (accessed: 02.05.2022).

Appendices

- Appendix A - Code Listings
- Appendix B - Vision Document
- Appendix C - Requirements Document
- Appendix D - System Document
- Appendix E - User Tests
- Appendix F - Surveys
- Appendix G - Project Handbook

Appendix A

Code Listings

Code listings

Code listing A.1: Get children

```
def get_children(ctx, course_id, worklist, folder_ids, node_ids):
    if len(worklist) == 0:
        return
    else:
        node = worklist.pop(0)
        node_id = node.data['id']
        response = contents_service.get_children(
            ctx.obj['SESSION'], course_id, node_id)
        if check_response(response) == False:
            return
        else:
            children = response.json()['results']
            for child in children:
                child_node = Node(child)
                node.add_child(child_node)
                if is_folder(child):
                    worklist.append(child_node)
                    folder_ids.append(child['id'])
                else:
                    node_ids.append(child['id'])

            return get_children(ctx, course_id, worklist, folder_ids
                , node_ids)
```

Code listing A.2: Depth-first search

```
def preorder(self):
        root = self

        root_node = Nd(root.data['id'] + ' ' + root.data['title'])
        def dfs(node: Node, root_node: Nd, parent: Nd) -> None:
                if not node: return
                elif parent is None:
                        parent = root_node
                else:
```

```
                nd = Nd(node.data['id'] + ' ' + node.data['
                    title'], parent)
                parent = nd
            for c in node.children:
                dfs(c, root_node, parent)

    dfs(root, root_node, None)
    return root_node
```