CREATION AND MANAGEMENT OF SMALL PROGRAMMING ASSIGNMENTS TO COMBAT PLAGIARISM

Lappeenranta–Lahti University of Technology LUT

Degree Programme in Software Engineering

Master's Thesis

2023

Rami Saarivuori

Examiners: Associate Professor Uolevi Nikula

Master of Science (Technology) Roope Luukkainen

ABSTRACT

Lappeenranta–Lahti University of Technology LUT
School of Engineering Sciences
Software Engineering

Rami Saarivuori

Creation and Management of Small Programming Assignments to Combat Plagiarism

Master's thesis

2023

72 pages, 26 figures, 6 tables and 2 appendices

Examiners: Associate Professor Uolevi Nikula and Master of Science (Technology) Roope Luukkainen

Keywords: design science, programming assignments, assignment management, plagiarism, programming education, CS education

This study applies methods from Design Science Research to build and evaluate short programming assignments and a management tool for them. New assignments are made to combat a form of plagiarism where students that have previously completed the course may share the answers to the next students that are taking the course. Because this causes the amount of assignments to rise considerably, this study also builds a tool to manage these assignments. The tool was named *Mímir* and it features, for example, import and export of these assignments. The study lists potential future areas of development for the tool as well.

To better understand what makes a better assignment and how they are a part of the teaching in general, some related work is analyzed. The study also lists programming assignment assessment tools to present some of the current landscape of them but makes it clear that these tools are not a solution to the problem of a large number of assignments that simply need to be managed.

The results of the study are that the assignments follow similar patterns of completion than previous assignments, though some new topics were seen as difficult for the students that completed these assignments. The instruction papers created with the tool made in this study were found to be clear and understandable, with some improvements compared to the papers not made with the tool.

TIIVISTELMÄ

Lappeenrannan–Lahden teknillinen yliopisto LUT
LUT Teknis-luonnontieteellinen
Tietotekniikka

Rami Saarivuori

Pienten ohjelmointitehtävien luonti ja hallinta plagiarismin kitkemiseksi

Diplomityö

2023

72 sivua, 26 kuvaa, 6 taulukkoa ja 2 liitettä

Tarkastajat: Dosentti Uolevi Nikula ja Diplomi-insinööri Roope Luukkainen

Avainsanat: suunnittelutiede, ohjelmointitehtävät, tehtävien hallinta, plagiarismi, ohjelmoinnin opetus

Tämä tutkimus soveltaa suunnittelutieteen menetelmiä rakentaakseen ja arvioidakseen lyhyitä ohjelmointitehtäviä sekä hallintatyökalun näille. Uusia tehtäviä luodaan, jotta voidaan kitkeä plagiarismin muotoa, jossa kurssin aikaisemmin käyneet opiskelijat jakavat seuraaville tehtävien vastauksia. Koska tämä aiheuttaa tehtävien määrän huomattavaa nousua, tutkimus myös rakentaa hallintatyökalun tehtävien hallitsemisen helpottamiseksi. Työkalu nimettiin Mímiriksi ja sen ominaisuuksia ovat esimerkiksi tehtävien tuonti ja vienti. Lisäksi tutkimus kuvailee mahdollisia tulevaisuuden kehityskohteita työkalussa.

Jotta voidaan ymmärtää mikä tekee tehtävästä paremman ja miten ne sopivat osaksi opetusta, joitakin aiheeseen liittyviä tutkimuksia analysoidaan. Tutkimus myös listaa joitakin ohjelmointitehtävien arviointityökaluja luodakseen kuvan niiden nykyisistä näkymistä mutta tekee selväksi etteivät nämä ole sopivia ratkaisemaan suuren tehtävämäärän tuomaa hallittavuusongelmaa.

Tutkimuksen tulosten perusteella voidaan sanoa, että tehdyt tehtävät seuraavat samankaltaista kaavaa suoritusten osalta kuin aiemmat tehtävät. Joissakin uusissa aiheissa on kuitenkin havaittavissa vaikeuksia opiskelijoilla. Parannetut tehtäväpaperit, jotka tehtiin tämän tutkimuksen työkalulla koettiin selkeiksi ja ymmärrettäviksi verraten niihin joita ei tehty työkalulla.

ACKNOWLEDGMENTS

Thank you Roope Luukkainen & Uolevi Nikula for the guidance, the great comments and the opportunity to do this study.

A big thanks to my family for the understanding and the support during my studies.

Another big thanks to all the wonderful people in various Discord servers who have managed to entertain & keep me sane.

Thanks to all my friends for making my time at this university the most memorable time of my life so far. Without you, I would not have all these wonderful memories.

Last, but in no way the least, I wish to extend the biggest of thank you's to you, Achlys & Aino. You two were the best support I could ask for. Thank you so much for the advice in life and love as well as allowing me to vent about my problems. Without your help I would not have made it this far. *May the Sun light your way. As you have illuminated mine.*

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

AI Artificial Intelligence.

ASCII American Standard Code for Information Interchange.

CAA Computer-Assisted Assessment.

CS1 Computer Science 1.

CS2 Computer Science 2.

CSV Comma-Separated Values.

DSR Design Science Research.

FREQ Functional Requirement.

GUI Graphical User Interface.

HTML HyperText Markup Language.

I/O Input/Output.

ID Identification.

IDE Integrated Development Environment.

IT Information Technology.

JSON JavaScript Object Notation.

MSPs Many Small Programs.

MVP Minimum Viable Product.

NFREQ Non-Functional Requirement.

OLP One Large Program.

PDF Portable Document Format.

PoC Proof-of-Concept.

RQ Research Question.

RTF Rich Text Format.

UI User Interface.

XML Extensible Markup Language.

1 INTRODUCTION

This chapter provides the introduction to this study. Motivations for it are briefly explained, with detailed explanations in the later chapters. The structure of the work is also outlined in this chapter.

1.1 Background and motivations

Teaching new topics to students is not an easy task. Students have varying degrees of knowledge of the topics that one tries to teach them beforehand and different students learn differently. Bromage and Mayer (1986) have demonstrated in their paper that repetition increases learning. This is why assignments are a big help in learning, as they provide a way to repeat the topic without, for example, trying to read the same text over and over again. Assignments help master the topic and this is especially the case in programming, as Denny et al. (2015) write in their research paper: "A key determinant of success for novice programmers is the extent to which they practice writing code". A number of researchers and companies have developed tools to aid in either creation or grading of programming assignments (Ala-Mutka 2005a). These however, are mostly auto-graders that help the instructor grade submissions or students to get instant feedback on their submission. None of the tools found were a match for the needs of the LUT Software Engineering faculty, who needed a tool to manage the small programming assignments used on their courses. They wanted to be able to easily select and create a set of assignments to use on a course, so that the sets selected are not too similar between years but still provide the same topics each lecture week.

The assignments on the LUT University's *"Principles of C Programming"* course have been similar for many years. The worry of the course staff is that this creates a possibility for plagiarism when the solutions can be copied or even bought from students who have completed the course in previous years. While the detection of plagiarism in normal academic writing is fairly straightforward, the detection of plagiarism in code can be difficult, as simple plagiarism detection tools can be easily fooled by changing variable names or the structure of the program (Jankowitz 1988; Mann and Frew 2006). To combat this problem, this study also includes the creation of new assignments, that are then tested on the mentioned course.

1.2 Goals and exclusions

The main goal of this study is to build a tool to aid in the management of programming assignments. It will also aid in the automation of the selection of programming assignments

and generating the instruction papers. It is intended to be language independent so that it can be used on multiple courses that use different programming languages. As this tool is intended to be used on programming courses specifically, it will not have features or options for any other type of assignment. The second goal is to create new assignment sets and variations of existing assignments to the C programming course at LUT University. To aid in the creation of these, this study analyzes previous assignments and research papers relating to the subject.

The Research Questions (RQs) are as follows:
RQ1: How to create and control multiple sets of programming assignments?
RQ2: How the layout of generated instruction paper compares to the handmade paper?
RQ3: Does changing the assignments and instruction paper affect the submission rate or quantity?

1.3 Study structure

This study consists of seven (7) sections. In this first section, the motivations, goals and exclusions are presented. The second section, Related Work, gives a quick look at literature relating to programming education and assignment management. The third section explains the research method used in this study. The fourth section is the practical portion of this study and explains the design and development of both artifacts created in this thesis. The fifth and sixth sections present the results and discusses them. Study concludes with a conclusion section.

2 RELATED WORKS

This section aims to provide a theory basis for the later sections. It should be noted that this is not intended to be a systematic literature review. As a baseline, in Table 1 the keywords and services used to find related works are presented. The results from the searches were scanned and promising texts were picked for further review. No systematic checking of all results was done.

Table 1: The keywords and services used in search of related works for this thesis.

Keyword	No. of results	Accessed	Service
computer science education	5 010 000	7.9.2022	Google Scholar
computer science education exercise	2 440 000	9.9.2022	Google Scholar
programming exercises open platform	108 000	9.9.2022	Google Scholar
forms of plagiarism	140 000	27.10.2022	Google Scholar
forms of plagiarism programming assignments	19 800	27.10.2022	Google Scholar
plagiarism in student assignments	61 000	27.10.2022	Google Scholar
forms of plagiarism assignment	50 700	28.10.2022	Google Scholar
cs1	2 599	7.9.2022	LUT Primo
computer science 1	2 481 714	7.9.2022	LUT Primo
computer science education	1 307 013	7.9.2022	LUT Primo
computer science education	29 265	9.9.2022	ACM Digital Library
programming learning system	306 129	30.9.2022	ACM Digital Library

2.1 Programming education and assessment

Although it has been questioned by the likes of Joughin (2010), the paper of Snyder (1970) and others like it on the assessment of students remains influential. They conclude that students allocate more time to those assignments or topics that are assessed and study or do assignments that are not assessed less, if at all. Cox and Clark (1998) add that the students adapt their learning strategies to pass their courses. This is why it is still important to assess and grade the assignments that students return completed to the course instructor, because the students focus more on the things that are assessed. However, if the number of students attending the course is high, the amount of work for the instructor to assess all the tasks is also high. Ala-Mutka (2005b) also notes that if the amount of tutors on the course that provide more personal guidance is increased, it can lead to inconsistencies in teaching and grading.

Ben-Ari (2001) notes in their paper that students might build an incorrect understanding of computer programming and programming structures. Students as novice programmers rarely plan their program in detail before they begin or test it during the making of it. For these and the aforementioned reasons, it is important to assess and give feedback to the students about their program.

Ala-Mutka (2005b) as well as Lister and Leaney (2003) all reference a study by McCracken et al. (2001) over assessment strategies, who found that students performed much more poorly than anticipated in a programming course. The study is notably due to the fact that it has 10 authors from 8 universities in 5 different countries. The study gives credit to the claim that there is something wrong with how programming is being taught. Among the findings of the study is also that a smaller percentage of students performed well. Lister and Leaney summarized the study by saying that the problem of current teaching is that the students should be able to program but weaker students cannot and are given a pass anyway. If the focus is put more on the weaker students, those that have stronger motivation or skills are left unchallenged to learn more leading to a drop in their performance on the course.

To combat the problem, Lister and Leaney (2003) suggest applying Bloom's Taxonomy of Educational Objectives. Bloom's taxonomy describes six different levels where each successive level depends on the behaviours acquired from the earlier levels (Bloom et al. 1956). These levels are *Knowledge, Comprehension, Application, Analysis, Synthesis* and *Evaluation*. The model of Lister and Leaney contains three grades that apply the principles of Bloom's taxonomy. The first grade is simply a pass, at the knowledge and comprehension levels of the taxonomy. The next level adds application and analysis levels, and the last grade contains all six levels of the taxonomy.

The first grade of Lister and Leaney's paper emphasized the ”literacy” of programming, i.e. that the students can read the code and the syntax, while also being able to make small programs of their own. Their assignments included pseudo-code instructions that the students needed to translate into working Java code. They also emphasized that the code must work completely, as mastering the Java syntax was considered non-trivial. The second grade had more traditional programming assignments, where the instructions were more abstract and lacked the step-by-step pseudo-code that the first grade assignments had. The third and final grade had a practical work that had a list of requirements but the students were free to choose their idea of what the program was supposed to do. These were then peer reviewed by other students based on pre-determined criteria.

The authors emphasized that their goal was not to give students a long hours of mundane work or reward students for prior knowledge of programming. The authors also defend their choice of peer reviewed works by saying that they believe students to be better motivated to

learn about good design when exposed to bad design. In addition they wished to decrease the emphasis on writing code and instead understand it and understand good design principles.

The assessment of programming skills without writing code is not a new idea. For example, Cox and Clark (1998) used an adaptation of Bloom's taxonomy in the form of the RECAP model, developed by Imrie (1984). This model uses two tiers of learning, *shallow learning*, with Bloom's first three levels and *deep learning*, which combines the last three levels of Bloom's taxonomy. Cox and Clark use these tiers to organize types of questions. The first tier focuses on knowledge and comprehensions so it has question types such as *Definition, Language syntax* and *The output of a program or program fragment*. The second tier focuses on and tests problem-solving, with question types like *What does this program do?, Under which conditions will this component perform correctly?* and *Which is the best solution to a problem?*. The authors emphasize that these types of quizzes should not be used as a summative assessment to determine a grade, but rather give nominal marks to emphasize the attempt itself rather than the performance.

Assessing most computer programs is often based on comparing inputs and outputs to what is expected. Simply comparing a model solution to the students solution on the code level is a poor method of assessment, as there are often many possible ways of achieving the result. Hence, simple code level comparisons can not be done automatically. While there can be ways of making solutions to an assignment that are more correct than others, if they produce the same output as the example solution, it is often correct enough. Assessing the functionality and design of each submission by the students by hand is time-consuming and not straight-forward even for experienced course personnel (Ala-Mutka 2005b).

2.2 Assessment Systems

Computer-Assisted Assessment (CAA) systems can be a great help in both large and small courses. CAA in this context means actions that support or implement assessment. In programming, CAA tools generally help by providing a place for submissions and parse and/or execute the program to study its outputs and behaviour. CAA can also be the only option for a course, if the amount of participants is high, and the instructor requires the students to submit their works multiple times over the duration of the course and receive feedback for them individually. (Ala-Mutka 2005b)

CAA systems have evolved for many decades now, introducing new languages, better static analysis and even sophisticated runtime analysis (Paiva et al. 2022). There have been a number of studies mapping the current landscape on e-learning tools and auto-graders for programming exercises, such as from Ala-Mutka (2005a), Douce et al. (2005), Paiva et al. (2022), Ihantola et al. (2010) and Caiza and Álamo Ramiro (2013). Of these, Paiva et al.

(2022) have the most up-to-date and the most comprehensive look at the current CAA landscape. While these studies map out several CAA tools that help instructors grade and students to submit their programming exercises or exams, such as *CourseMarker* and *VIOPE*, only a few mention some type of capability to handle a large set of assignments in other ways than simply storing them for use in auto-grading, or only storing the test cases.

One such system is presented in an article by Tung et al. (2013). The authors present a system of their creation designed to cover most phases of assignments, from their creation to detecting plagiarism in student submissions. This system, called *PLWeb*, has two main parts: the server and an Integrated Development Environment (IDE). The IDE is run locally on the student's computer, which sends data to the server about the assignments and the student's code, and retrieves new assignments from the server. Instructors can monitor the student's performance in the assignments from a web interface.

Other systems, although not as sophisticated as *PLWeb*, are the *CloudCoder* and *EMSEL* (Exercise Management System for e-learning), presented in papers by Papancea et al. (2013) and Balaa (2016) respectively. Both *CloudCoder* and *EMSEL* also present themselves as a platform for managing short programming assignments, though they seem to lack plagiarism detection tools that the *PLWeb* has. According to their website, the development of *CloudCoder* has also been stopped in early 2022 (Spacco et al. 2022). *EMSEL* was presented in the study as an Minimum Viable Product (MVP), with limited support for languages, only mentioning HyperText Markup Language (HTML) and JavaScript as currently supported, though mentioning possible development of Extensible Markup Language (XML), PHP and C++ support (Balaa 2016).

The *PLWeb* system's assessment rules are defined with XML files, as opposed to *CloudCoder*, as well as many other systems like *VIOPE*, *CodeGrade* and *Artemis* (Tung et al. 2013; Papancea et al. 2013; Nikula et al. 2011; *CodeGrade* 2021; Technical University of Munich 2023). These systems have some type of Graphical User Interface (GUI) that the teacher can use to make assessment rules. *VIOPE*, *PLWeb*, *Artemis* and *CodeGrade* also mention the capability to test that the student has used specific features. *EMSEL* seems to use Rich Text Format (RTF) files to store its assignments and rules.

2.3 Plagiarism in programming courses

Plagiarism in the academic world has been defined as "the practice of using other's ideas and texts and claiming them as one's own original authorship without acknowledging the source" (Sharma 2010; Chuda et al. 2012). Addressing plagiarism, and cheating in general, is important, because like Dick et al. (2002) write in their article, students who cheat may not achieve the competence that their profession requires and it harms the academic community

in general. The lack of competence may result in damages to society and the reputation of the profession, institution or the degree, while the damage to the academic community can be harm to the educational environment as more resources are needed to combat cheating and fellow students, who may be at an unfair disadvantage compared to the people who cheat.

Plagiarism is unfortunately a common problem among universities and colleges and is nothing new. Belter and Pré (2009) refer to three different studies by Angell (2006), Roig and Caso (2005) and Whitley (1998), where it was consistently shown that at least half of college students admit to have cheated at some point during their academic careers. McCabe (2005) lists among the most common forms of plagiarism such forms like purchasing papers for submission as original work, copying material verbatim without proper quotation, inadequately paraphrasing, using material without proper citation, and collaborating on an assignment without being authorized to do so.

Students usually commit plagiarism for two main reasons, intentionally and knowing the consequences or unintentionally, usually being uninformed of the seriousness of it or the consequences of it (Belter and Pré 2009). So in addition to methods of detecting plagiarism in student works, there is a need for also informing the students about plagiarism and the consequences of committing it.

In programming courses, plagiarism can manifest in a few different ways according to Joy and Luck (1999):

- A student collaborates closely with another student, in the belief that this is acceptable
- A student with poor success in studies copies, edits and submits a colleague's program or a part of it with or without their permission, hoping that this will go unnoticed
- A poorly motivated (but not necessarily with poor success in studies) student copies and edits a colleague's program in an effort to minimize the work needed for the assignment.

As the authors note, the first case is a grey area of acceptability. It is usually preferred for the students to collaborate to aid in learning. However, it can be difficult to determine what is the boundary between plagiarism and encouraged cooperation. In the context of programming, plagiarising the idea of someone else can be defined as plagiarising the idea of an algorithm, meaning that while they do not copy a section of the program directly, they can copy the logic of how the result can be achieved. This however is less of an issue in basic Computer Science 1 (CS1) and Computer Science 2 (CS2) level courses, as the assignments can be quite simple, meaning that there might only be one or two good ways of achieving the expected result. Like Chuda et al. (2012) write, the bigger issue in those courses is determining what is plagiarism and what is not, when the scope and variability of the assignments is limited.

In this study we are interested in the last two forms that Joy and Luck (1999) present in their paper. The task of copying assignment solutions has the potential of becoming easier if the assignments are the same across multiple implementations of the course. This increases the number of students who have those solutions to the assignments and with that increases the chance of students finding someone who is willing to give the solution (Dick et al. 2002).

Belter and Pré (2009) write in their paper that the students who are more likely to commit unintentional plagiarism are the students who perform more poorly compared to others. They continue to write that an effective way to decrease at least unintentional plagiarism is to simply teach the students the correct way to cite and quote others.

Previously, plagiarism specifically in LUT programming courses has been looked into in research by Hynninen et al. (2017) in their research paper. While this paper left out the year-to-year sharing of assignment answers, it proves that LUT is not immune to plagiarism and in fact produces an alarming finding that almost 53% of students may have been involved in plagiarism in 2016. The study however does not give indication as to the number of assignments they were involved in, only that they may have been involved in one or more assignments, so the actual amount of plagiarism in likely lower than this result may indicate at first glance.

3 RESEARCH METHOD

This study, as told in Section 1.2, aims to build a tool to aid in the management of and in the automation of the selection of programming assignments as well as creating new assignments and variations of old assignments. This section will establish the method of creation and research in order to begin the process of creating these artifacts.

Design Science Research (DSR) is used as the base for creating the aforementioned artifacts. DSR is a research paradigm for creating and evaluating Information Technology (IT) artifacts intended to solve identified problems, thereby contributing new knowledge to the body of scientific evidence. These created artifacts are both useful and fundamental in understanding the problem (Hevner, March, et al. 2004; Hevner and Chatterjee 2010). The two main design processes in DSR are *build* and *evaluate*, while the four types of artifacts it can produce are *constructs, models, methods* and *instantiations* (March and Smith 1995). The two processes, *build* and *evaluate*, are usually iterated over many times to create the final artifact (Markus et al. 2002). To the problem that is addressed, these artifacts are built in an attempt to solve them and then evaluated in how well they solve the problem and are a fit for the organization (Hevner, March, et al. 2004).

The four artifacts that DSR can create represent different solutions to different problems. *Constructs* help understand the problems and solutions by providing the language how they should be defined and communicated. *Models* extend constructs to represent something in the real world. They aid in understanding the problem and solution by presenting, usually visually, the connections between parts of the problems and the solutions. *Methods* on the other hand define processes with which to solve problems. Finally, *instantiations* show how the previous three can be implemented in a working system. They also demonstrate the feasibility and effectiveness of the models and methods they contain. (March and Smith 1995; Hevner, March, et al. 2004)

Peffers et al. (2007) advise in their paper to build upon the prior research done in the field. The previous sections of this study have aimed to do exactly that and have provided the foundation of knowledge from prior research upon which to conduct the DSR made in this study. This is also consistent with the general evaluation pattern's first evaluation as described by Sonnenberg and Brocke (2012). Also, evaluation methods two to four from Sonnenberg and vom Brocke are used, them being logical reasoning, experimenting with prototypes and finally surveys. Simple questionnaires are used to gather data in the testing phase of the assignments, particularly their perceived difficulty. A standard course feedback survey at

the end of the course is also utilized. An interview with experts is conducted at the end to evaluate the tool made in this study. The tool is also tested on the course with the newly made assignments.

To summarize, this study:

- aims to build viable artifacts to important and relevant problems via the available means
- evaluates, demonstrates and explains the utility, quality and efficacy of the artifacts
- describes the function and design of the artifacts so that these artifacts can be verified in a clear and effective manner

With these goals and methods, this study fulfils of the seven guidelines as outlined by Hevner, March, et al. (2004) in their article. These guidelines are:

1. Design as an Artifact
2. Problem Relevance
3. Design Evaluation
4. Research Contributions
5. Research Rigor
6. Design as a Search Process
7. Communication of Research

The first goal is fulfilled by the goal of this thesis itself, as the aim is to build more than one artifact, both the tool and the new assignments. The second guideline is fulfilled in the next chapter and the evaluation is done at the end of this thesis. Research contributions are clearly explained and made available for others to also verify, with the exception of the assignments. Their detailed descriptions will not be published to prevent their effectiveness within the students from being diminished. Available and suitable research methods are used in the creation and design of the artifacts, filling the fifth and sixth guidelines. The last guideline is filled with the detailed explanations in this thesis.

4 DESIGN ARTIFACTS: ASSIGNMENTS AND ASSIGNMENT MANAGER

This section explains the design artifacts that are created in this thesis. The designs of new assignments and the new assignment manager are explained and their development recounted. The justifications for the design choices made are also explained.

4.1 Assignments

In this section, previous assignments found on the C programming course are briefly analyzed in conjunction with research on programming assignments with the goal of understanding what changes in the assignments have yielded better results. The design philosophy is then derived from the results for the new assignments and their variations.

4.1.1 Previous Assignments

Data has been gathered quite consistently on assignments and course results at the basic CS1 and CS2 level courses at LUT. The assignments that are the goal of this study are meant for the CS2 level course that is taught with the C programming language. With this in mind, data was selected from the assignment results of that course. This data was available from 2018 onward, though data from 2019 to 2022 was selected, as the course implementation was very different in 2018 compared to the years selected for this study.

The course had 7 weeks of programming assignments in each implementation, with each week having between 3-5 assignments. Altogether, the course had 31 programming assignments, though the last one, L7-A3 (assignment 3 of lecture 7) was an optional extra assignment.

As seen from Table 2, during the course implementations in the years 2019-2022, the course had between 250-288 students enrolled. When comparing this to Figure 1, it can be seen that around 10% of students did not start the course although they were enrolled.

From Figure 1 we can see that there are significant differences between years in some of the assignments. Some of this difference can be explained by the increased amount of students enrolled each year shown in Table 2, but the differences especially towards the end of the course are not explained by the increased enrollment alone. The overall jump in 2022 can however be explained by the increase in active students that year. The course materials have received significant updates, especially between 2021 and 2022, that are not easily quan-

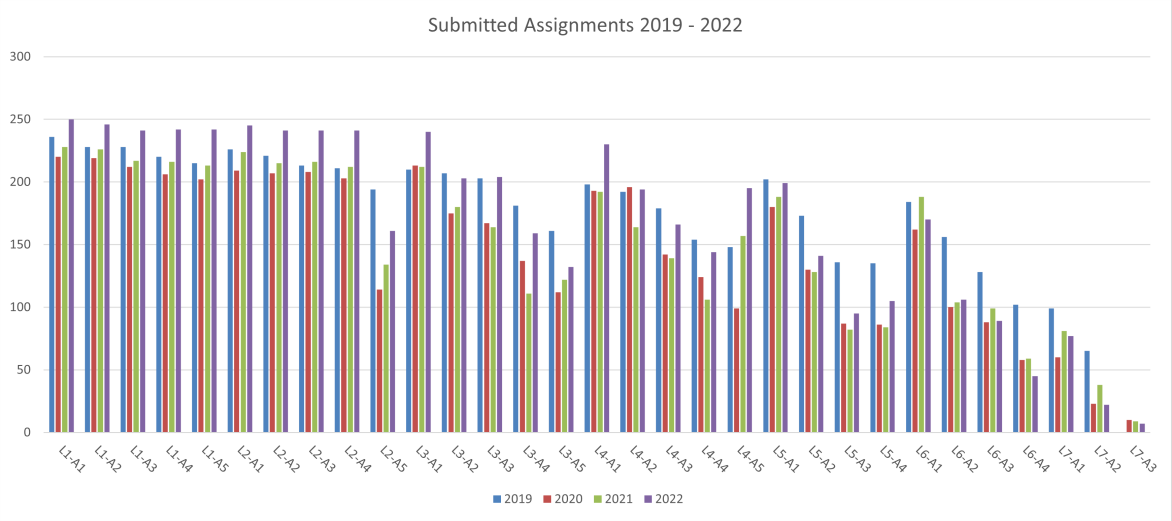


Figure 1: Successful assignment submissions in the C programming course at LUT University between 2019 and 2022.

Table 2: The number of students enrolled in the C programming course at LUT University between 2019 and 2023. Note that the number of active students in 2020 and 2021 are estimates based on completed assignments in week 1.

Year	Enrolled students	Active students
2019	250	226
2020	251	≈220
2021	271	≈230
2022	288	252
2023	266	242

tifiable and separate data on the effects of those updates is not available. The assignments themselves have received updates over the years, but the updates have been minor, such as being more specific on what the student is expected to do or changing the wording to avoid confusion that may have been a problem in previous years.

4.1.2 New Assignments and Variations

For the course implementation in 2023, new assignments were made. Most of these assignments were based on the previous years assignments and were made to be variations of the same assignment by modifying the original by changing data sets, texts, program execution sequence and the assignment instructions. Some of the new assignments were however completely new. These were still designed to teach mostly the same structures and themes that were taught in that week, but with new topics. In order to assess the success and suitability of the new assignments, data from submissions was gathered. A questionnaire relating to the themes of the assignments was also put forward each week.

The assignments were kept with the same style of Many Small Programs (MSPs) as earlier implementations had. This was done because research has shown that this has a positive impact on the students learning and view of the course as well as the assignments (Allen et al. 2019). The study by Allen et al. found that students started the MSPs earlier than One Large Program (OLP), completed more of them than necessary and use them to study for exams.

Not including the first two assignments in the first week of the 2023 course, all the assignments were new. As the course had the same structure overall, assignment completion between the years can be compared. The only major difference was change to a new auto-grader the course used. This however allowed us to gather more detailed information on how the students returned their assignments. The students were allowed an unlimited amount of submissions to the auto-grader.

While the basic concepts in the assignments were kept mostly the same, with only slight variation in the number of occurrences, there were a few topics where the number of occurrences jumped significantly. These were file handling, use of `fgets`, standard library `string.h`, usage of error handling and string manipulation. The use of file handling was purposefully inflated in the new assignments, as based on empirical experiences from earlier years this was a difficult topic for students, so chances of practising it were increased. This also partly explains the increase in the use of `fgets`, as the use of `fscanf` for file reading was phased out in the example solutions in favor of `fgets` and also the increase in error handling, as it was determined to be good practice to use in file operations during the course. It was also checked with the auto-grader in each student submission. The tables containing all listed occurrences of topics and their amounts are located in Appendix A and Appendix B.

The new assignments had also more assignments that were continuing from a previous week's assignment. Each week starting from the 2nd lecture, one of the assignments was a continuation of the previous week's assignment. This was usually the 3rd assignment, with the exception of 2nd and 7th lectures, where the main continuing assignment was placed 5th and 2nd assignment respectively. A few others were also continuations from a previous assignment, but these were mostly only a reference to a specific topic, such as the L5-A4, which referred to an assignment in lecture 2.

The difficulty level of the new assignments and their variations were considered to be mostly the same as the previous assignments, with 20 assignments considered minimum level, 6 in basic level and 5 in target level, compared to 20, 5, and 6 in previous assignments respectively. The minimum level is considered to have the topics that are required to learn to pass the course. The basic and target levels have more topics for those seeking a higher grade and

better programming understanding. The breakdown of level per assignment can be seen from Appendix A and Appendix B.

New topics introduced, not previously included in the assignments, were type conversions, American Standard Code for Information Interchange (ASCII)-table and the `ctype.h` standard library. These were included as a way to increase the range of topics as well as because it was seen that these were something that a good number of new assignments could be derived from. Four new assignments with these topics were made for the batch of assignments created in this thesis.

The new assignments also used new sources of data. This was done to bring more variety into the assignments as well as tie them more to the real world. The data sets chosen were either already clean or they were cleaned and condensed to smaller files so the students could test their programs with limited data set where it was easy to manually perceive what the correct answer was when data was handled. These data sets included such things as excerpts from books in the public domain, leaderboard data from sports, anonymized data from previous courses and open data from government databases.

4.2 Mímir - The Programming Assignment Manager

With the addition of new assignments that this thesis creates and others may do in the future, a new problem is created that is also a part of the RQ1 of this thesis: how to manage a large library of programming assignments? This section dives deeper into the purpose and design choices that were made during the planning of Mímir. Requirements from the LUT staff and limitations from the used and available technologies are used to explain aforementioned choices.

4.2.1 Motivations

Hevner, March, et al. (2004) in their rules for DSR state that the artifact created should be an ”unsolved and important business problem”. This tool is created because as Section 2 shows, there are no pre-existing tools that would solve the problem described. While *PLWeb* supports the creation and management of assignments, the system needs to be run locally on the students computer and is in general more of an auto-grader than only a tool for management. *EMSEL* on the other hand does not seem to feature any proper management or search functions for the assignments. Neither also do not appear to support exporting the assignment instructions into a unified paper.

The other tools shown in the previous sections are mostly auto-graders designed to integrate directly onto the learning platform and only intended to store the test rules and provide means

of assessing the students' submissions. The problem presented in this study is not solved by simply storing the assignments. Instead, they need to be easily searchable, viewable and exportable to a unified format. Assessment is also not a part of this specific problem as it is solved by existing solutions.

4.2.2 Scope & Purpose

As implied in previous sections of this study, the purpose of Mímir is to help a programming course instructor manage their library of course assignments and to combat plagiarism by aiding in the selection of these assignments from a pool of similar assignments. This way the assignments can be switched between the course implementations, or even at the instructors discretion, switched between different groups of students on the same course. It is not intended to be in any way a CAA tool meant to aid in submission or assessment of student submissions and hence it will not have any features meant to aid in facilitating them. Instead, it is meant to be an aid for the course instructor before the course or before the release of the assignments. While the tool is not for students, they can be seen as a stakeholder, as well as the teaching assistants for the course.

Mímir is intended to be easy to use and to help in compiling the assignment instructions from the selected assignments, while keeping the visual style and readability consistent across course implementations and lecture weeks during the course. This should help the students focus on the task itself and helps the course instructor to focus on assignment creation and other duties rather than trying to make the visuals of the assignments readable enough. Mímir also helps the instructor by providing an easy method of editing the assignments based on previous feedback or changes in teaching. This way the edits are easy to implement and do not require fiddling with text editors.

4.2.3 Requirements

The tool's requirements were discussed together with the C programming course staff. These requirements were shaped into Functional Requirements (FREQs) found in Table 3 and Non-Functional Requirements (NFREQs) which are found in Table 4. The most significant of these requirements are labeled with a priority of 1, namely the ability to input the assignment data or load it from a file (FREQ-01 and FREQ-02), generate the assignment briefings (FREQ-06) and save what was selected (FREQ-05). The priorities where determined by selecting the core functionalities. The NFREQ-03 means that the program should notify the user of how it is processing data, i.e. what the program is doing.

The requirements for course teaching assistants can be seen as much of the same as students, with only one major difference. While students need the generated set as a unified document

with example outputs, assignment details and possible example data, teaching assistants also need the example solution. For this reason, it should be ensured that the two documents are marked clearly, so the answers to the assignments do not leak accidentally by presenting the students with the wrong file by mistake.

Table 3: The functional requirements of Mímir. Priority has a scale of 1-5, where 1 is top priority and 5 is not a priority.

Requirement ID	Priority	Description
FREQ-01	1	User must be able to input the assignment data (instructions, code and metadata)
FREQ-02	1	User must be able to load assignment data from a file
FREQ-03	1	User must be able to save assignment data
FREQ-04	1	The tool must be able to pick the correct amount of weekly assignments
FREQ-05	1	The tool must remember what assignments or their variations have been selected and when
FREQ-06	1	The tool must be able to generate assignment instruction documents for students
FREQ-07	3	The tool must run the weekly assignments and generate the example run for the instruction document automatically
FREQ-08	2	The tool must generate instruction documents for teaching assistants that includes the example solution
FREQ-09	5	The tool must generate auto-grader submission settings
FREQ-10	4	The tool must be able to use tags in the assignment metadata to select assignments of the same level.

Table 4: The non-functional requirements of Mímir. Priority has a scale of 1-5, where 1 is top priority and 5 is not a priority.

Requirement ID	Priority	Description
NFREQ-01	1	The GUI of the tool shall be easy to understand and intuitive
NFREQ-02	1	The tool shall not be made non-functional by the user
NFREQ-03	3	The processing of data should be informative
NFREQ-04	3	It shall be clear to the user what information is stored where
NFREQ-05	2	The GUI of the tool should be responsive
NFREQ-06	2	The GUI of the tool should be minimalistic

4.2.4 Architecture

Mímir is intended to be a simple tool from the user perspective. The only major external dependency that can not be included in the program itself very easily, as seen in Figure 2, is the `pdflatex` program, which is needed to compile the LaTeX (LaTeX) documents into Portable Document Format (PDF) files. The example solution compiling and running is intended to be voluntary, so if the user chooses not to use the feature, external compilers or

runtimes are not needed. The tool does need to have access to the storage on the device the tool is run on, to import and export assignment and assignment set data.

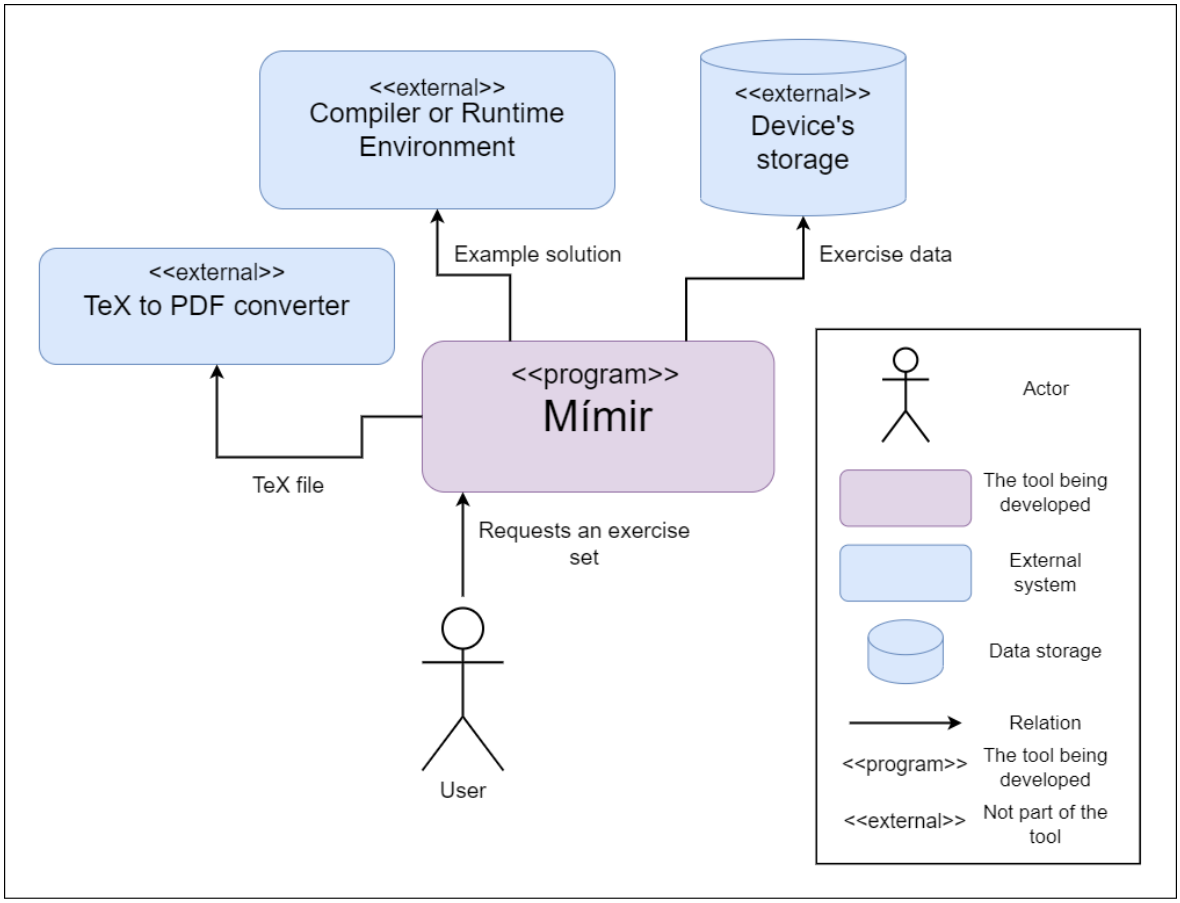


Figure 2: The context diagram of Mímir

For simplicity and easy maintenance, Python was chosen as the language the tool will be built on. The documents were chosen to be first generated into LaTeX documents, because LaTeX features a robust and programmatically easy way to compile uniform and clear documents from different types of data. While Python does have some libraries for generating LaTeX into a PDF, such as the `PyLaTeX` and `pdflatex` modules, these are simply a wrapper for calling the original `pdflatex` program from the commandline using Python's `subprocess` module (Fennema 2020; mbello 2019). For this reason and for simplicity, it was decided that a wrapper of our own should be written, and use the `subprocess` module directly when interfacing with the commandline `pdflatex` program.

While the idea of the tool is simple enough, the complexity and the amount of different data that is required to compile the final assignment instruction papers means that a commandline tool would require too much manual labor to be effective in reducing the amount of work required to put together the weekly assignments. This meant that a GUI was required. GUI

libraries that were mainly looked at were PySimpleGUI, Qt, Kivy and DearPyGUI. These libraries had their strengths, like simplicity for PySimpleGUI or robustness from Qt, but most fell short in either the clarity of documentation, like Qt or Kivy, or being visually too simple like PySimpleGUI. That is why DearPyGUI was eventually selected to be the User Interface (UI) handler component that can be seen from Figure 3.

Each component in Figure 3 has an important role in the tool. As mentioned above, the window handler component handles the tasks related to the UI of the tool. The main component simply acts as a starting point for the program, calling the initializer and the window handler components. The initializer, as its name suggests, handles the initalization of the environment, such as runtime constants, cache folder on disk and other environment variables. The data handler facilitates most of the Input/Output (I/O) operations, such as the retrieval of assignment data and storing that on disk. I/O operation that it does not handle is the generation of the PDF documents, as that is handled through the `pdflatex` program via the external service handler component. The assignment set generator is used to select assignments for generated sets. The briefing generator will then use that set and its related data to generate the LaTeX file which the PDF is generated from as mentioned above.

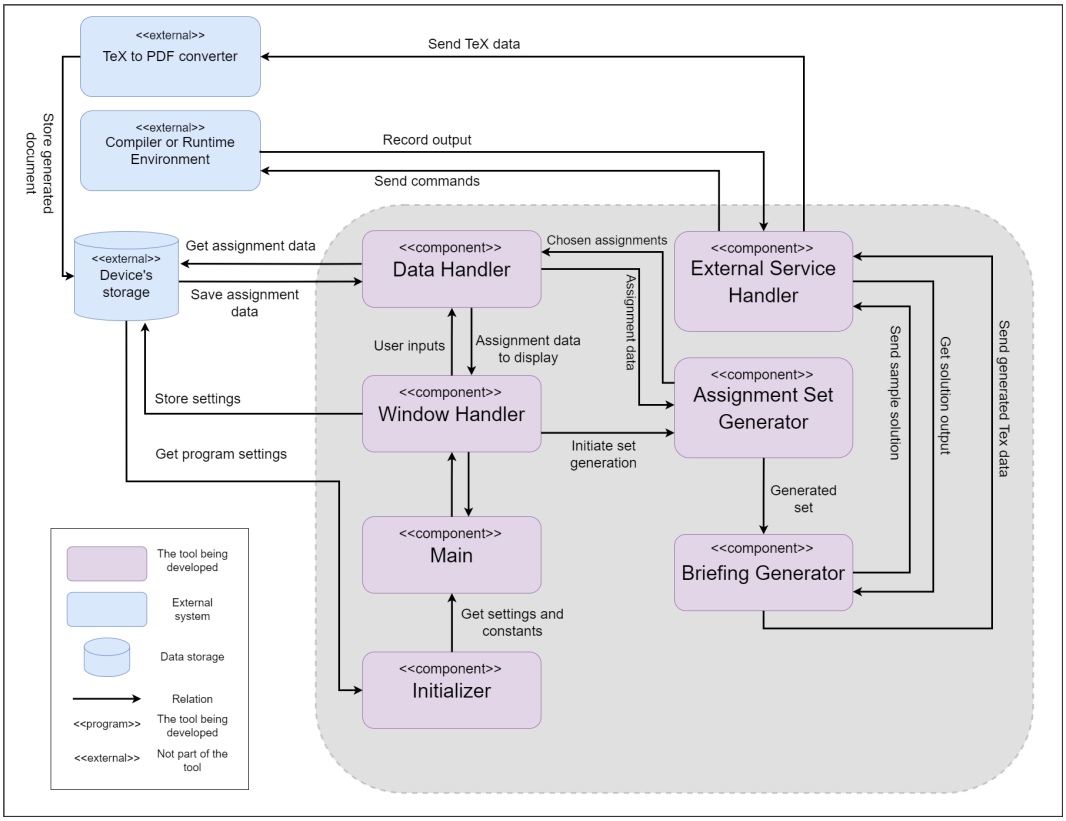


Figure 3: The functional view of Mímir.

To facilitate a fast, efficient and reliable search for a large amount assignments that would or might be stored in the program, a library called `whoosh` was selected to be used. This

allows the indexing of the assignments into a central index. This index will only hold the data necessary to find the rest of the files, so an entry there should only contain the most important fields and those that would be most likely to be search from, in addition a path to the JavaScript Object Notation (JSON) file that will contain the rest of the data.

The program will have a window to input all the assignment information, like title, lecture number, instructions and example files. The data is then stored to a JSON file. The structure of the stored JSON file can be seen from Figure 4. The ”main level” of the file contains the general assignment information that is not specific to a variation or a single example run. This includes the assignment Identification (ID), that was chosen to be generated via a SHA-256 hash of the rest of the JSON file. This allowed for a unique ID for each assignment.

The variations of the same assignment are then listed as a list of assignment objects, each having their own versions of instructions and paths to the files. As example runs are tied to a variation and there can be multiple example runs per variation, they are stored inside a list that is inside the variation. An example run object stores the inputs and outputs that are tied to the run. Finally, each variation also stores the course implementations that it has been used in, such as ”Spring 2023”. As there can be multiple values for some of the keys, they are stored in a list as seen from Figure 4. The assignments can suit multiple places in the week, there can be several code and data files, and with a high probability an assignment will have more than one input.

As to not re-invent the wheel, the final look of the finished PDF was designed to match the old document design in regards to the main elements. Tweaks were of course made, such as separating the inputs to their own block from the example output to allow for easier testing of students’ own programs on their computers. Other tweaks included adding colour highlights to keywords in instruction text and code in the teaching assistant versions. Fonts were also changed from the original to be more modern, but the font selected for the input and output blocks was still monospaced to allow for easier reading of the output format and especially whitespace.

To allow for changing some of the document characteristics, such as using other packages, changing font sizes, header and footer design and margins, a JSON file containing the aforementioned attributes and others was created. This file included, in addition to the aforementioned, options for document class, fonts and hyphenation. It also included an ”other” key, where LaTeX commands that needed to be run before the actual content could be placed. For example, during development, when the `minted` package was added, a command specifying the background colour was placed under the ”other” key.

```
{
    "course_id" : "CTxxAxxxx",
    "course_name" : "Course Name",
    "assignment_id" : "X",
    "exp_lecture" : 0,
    "exp_assignment_no" : [0, 1],
    "tags" : [],
    "title" : "",
    "next, last" : [],
    "code_language" : "c",
    "instruction_language" : "FI",
    "variations" : [
        {
            "variation_id" : "A",
            "instructions" : "Tee C-ohjelma, joka ... ",
            "example_runs" : [
                {
                    "generate" : true,
                    "inputs" : [0,0],
                    "cmd_inputs" : ["", ""],
                    "output": "",
                    "outputfiles" : ["", ""]
                }
            ],
            "codefiles" : ["", ""],
            "datafiles" : ["", ""],
            "used_in" : []
        }
    ]
}
```

Figure 4: The structure of the JSON file that contains the assignment information

Huang et al. (2022) propose and implement an exercise recommendation system with four stages of selection. The fours stages presented are *Data preprocessing, Candidate Generation, Diversity Promoting* and *Scope Restriction*. The authors do not explain the data prossesing stage at all. In the candidate generation stage, the authors state that it uses *"an enhanced random walk algorithm on an educational tripartite graph that is composed of a set of KCs*[knowlegde concepts]*, exercises and class materials"*. Diversity promoting is meant to remove candidates with high lexical similarities and the last stage is meant to filter candidates that are beyond the scope of the course they are meant to be used on.

As the aforementioned model is intended to be more of a assignment generator using Artificial Intelligence (AI) rather than a simple selector and due to the fact that it is highly complex, Mímir will not have this kind of selection process for the assignments. Rather, the tool will

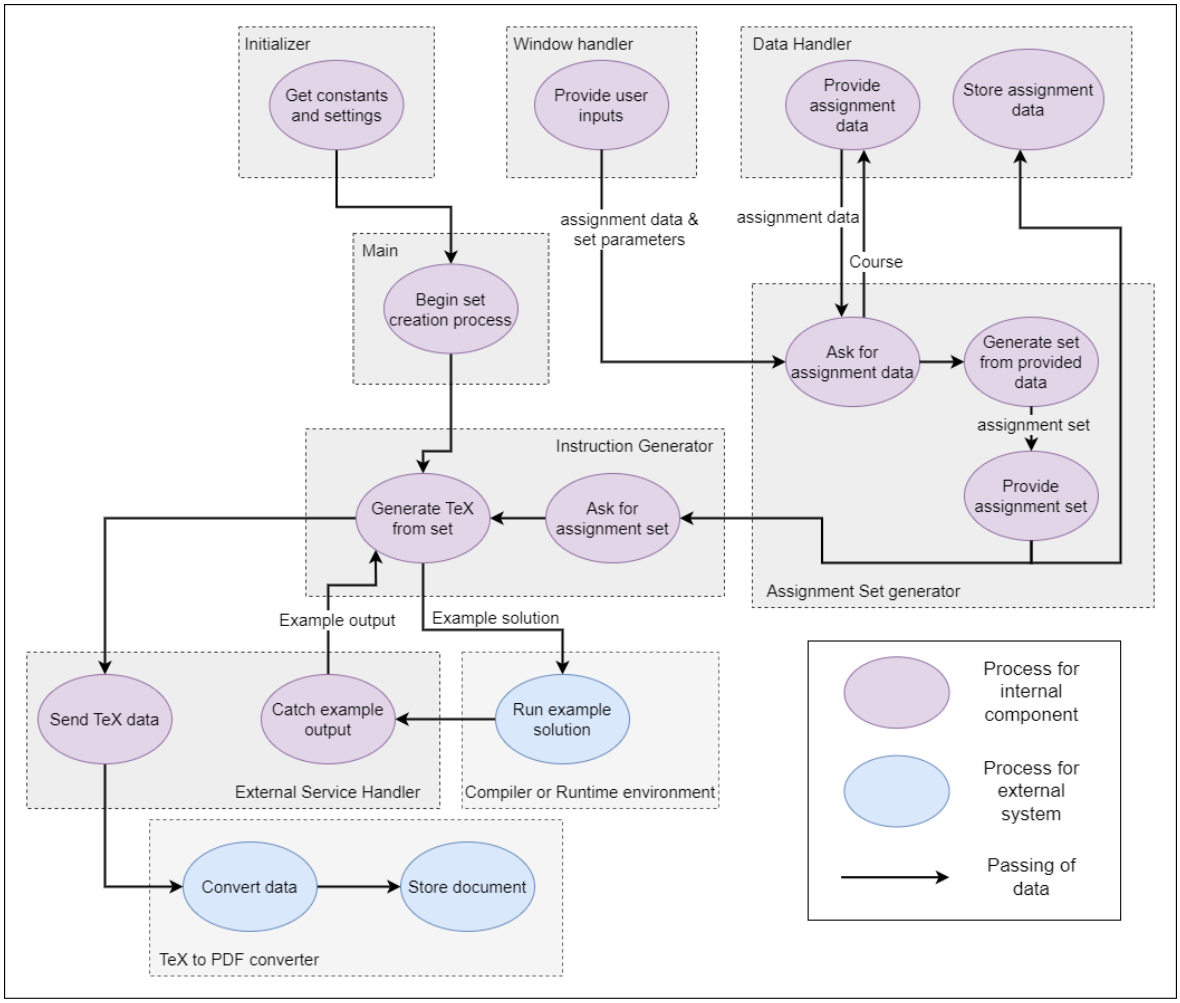


Figure 5: The Information view of Mímir

use a two-stage selection process for the assignments. First, the candidates locations within each weeks paper is determined. As seen from Figure 4, assignments can have multiple locations on the instruction paper that they are suitable for. After the assignments have been sorted, they are assigned priorities. For example, successive continuous assignments need to have a priority over non-continuous assignments. A higher priority is also given to those assignments that were not selected in previous selections, i.e. if the assignment was chosen in the last implementation of the course, it will have a decreased chance of getting selected again. The longer an assignment has not been selected, the higher its chance of getting selected grows. This provides a solution to the problem of students sharing answers from the previous implementations with the same assignments. The instructor will have a chance of manually overriding the selections though, in case the instructor wants to choose another assignment for any reason.

Because it was thought that in an international university such as LUT, there might be others than those speaking Finnish who might use the tool, its UI was made to be easily translatable. All the texts used in the UI were stored in a JSON file, with each text key containing its values in both English and Finnish. This also allows for any language to be added afterwards, when the program takes its texts always from the loaded JSON file. An example of this file can be seen from Figure 6.

```
{
    "ui_main": {
        "FI": "Päävalikko",
        "ENG" : "Main"
    },
    "assignments" : {
        "FI" : "Tehtävät",
        "ENG" : "Assignments"
    },
    "ex_run" : {
        "FI" : "Esimerkkiajo",
        "ENG" : "Example run"
    }
}
```

Figure 6: An example of the JSON file containing the texts used in the UI of Mímir.

4.2.5 Visual Design

The name for the tool was chosen to be Mímir, which was inspired by Norse mythology. In Norse mythology, Mímir is a deity known for his wisdom and said to be the wisest of them (Cunningham 2022). In Figure 7 the designed logo for the program can be seen. The logo was inspired by the logos of the subroutines of the fictitious AI named *GAIA* in the *Horizon*-series of videogames developed by Guerilla Games.

Figure 7: Logo of the Mímir tool.

Before the development on the UI implementation was started, the UI was designed using *Figma* (2023). Requirements in Tables 3 and 4 were taken into account when designing the

UI. In addition, the selected UI library's features were taken advantage of, as it allowed collapsible headers and internal windows to be created and used. This allowed all the necessary components to be inside one main window.

In Figures 8 and 9 some examples of the UI design can be seen. The fields in Figure 8 were designed to be dynamically updated and saved, as the UI library allowed this. The fields in Figure 9 were not intended to do this however, as the data is better to save all at once into the JSON file that stores it. This also prevents the program from creating files that ultimately contain nothing, for example when the user clicks to add an assignment but then cancels immediately or after inputting only a test value. These dummy files cannot then create mixups or odd behaviour in the assignment set generation. The logo in Figure 8 was later in development changed to the one shown in Figure 7, because the text below the logo itself created too much empty vertical space in the *Course information* section of the UI.

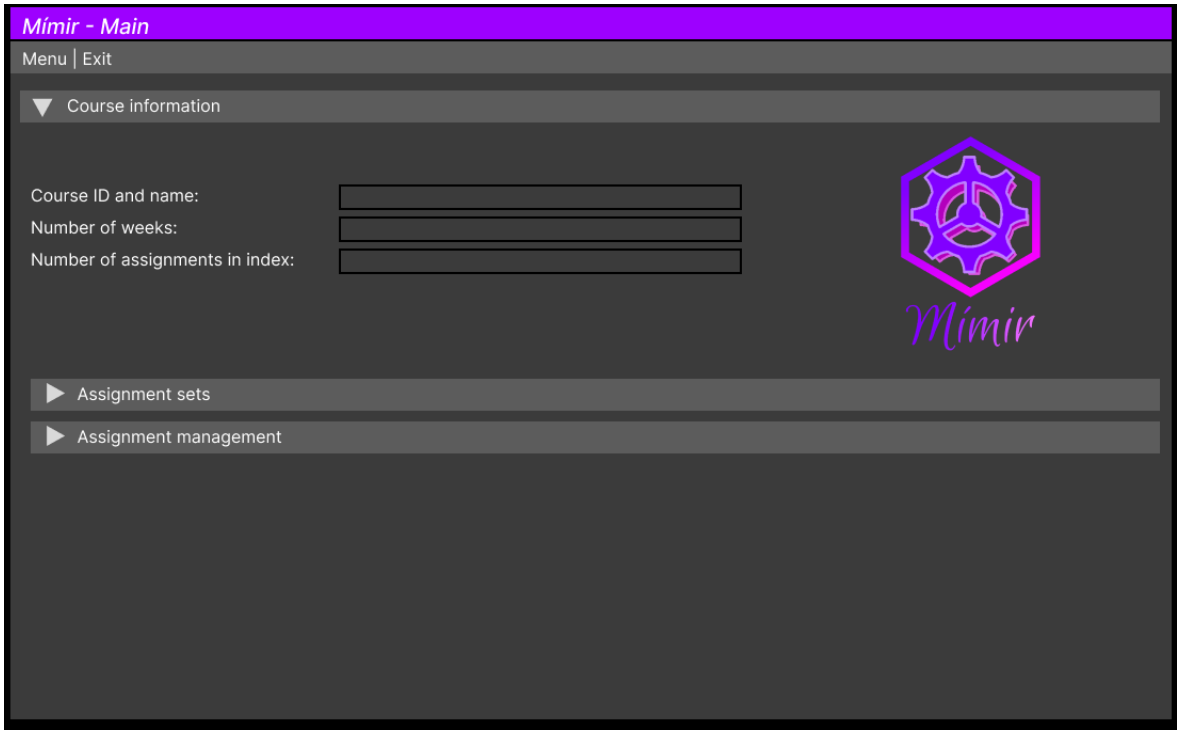


Figure 8: The main menu and landing page mock-ups of Mímir.

The question marks seen in Figure 9 are popups that display a help text for the user. This text gives the user a tooltip as to what to input in the field and what the value is used for. These texts are stored the same way as other GUI texts seen in Figure 6.

The goal for designing the final assignment instruction paper was to keep the main elements but improve the clarity and readability of it. Previously, the paper only had a background

colour in the example run and the inputs were not separated from the output. The new design had the inputs also in a separate container on the page, as seen in Figure 10.

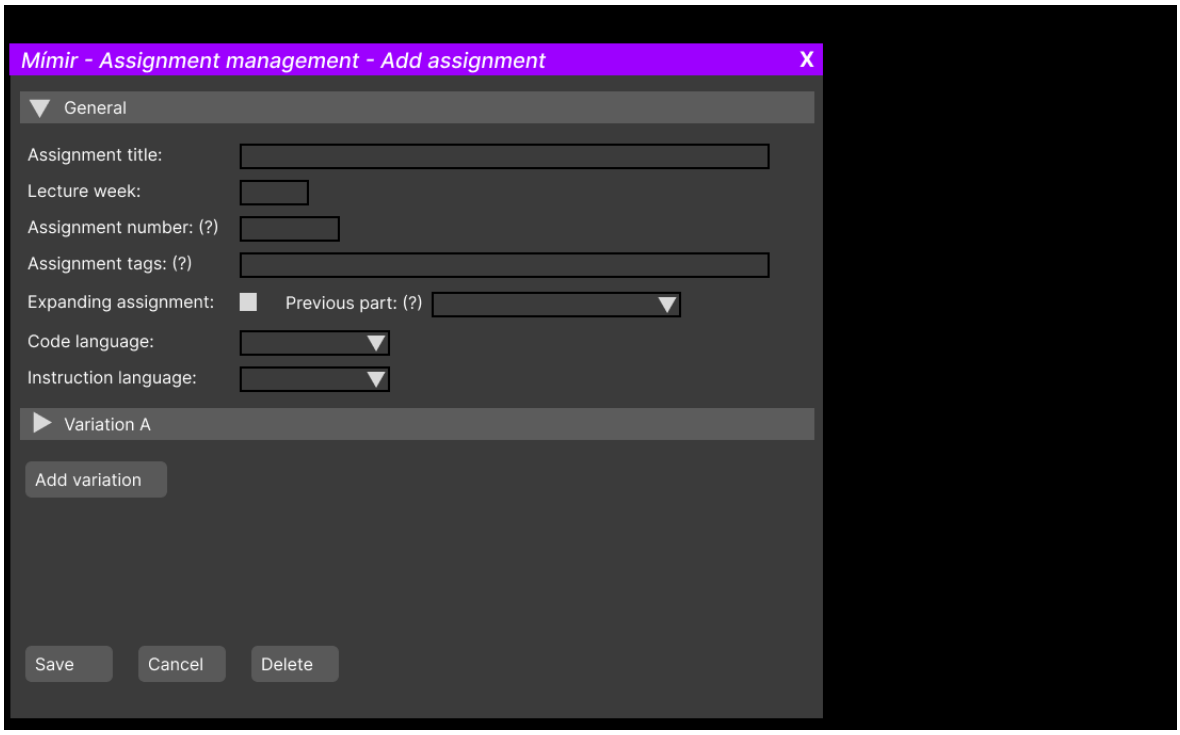


Figure 9: A mock-up of a window for inputting the assignment data.

4.2.6 Development

The development of the tool begun with making Proof-of-Concept (PoC) versions of the TeX generator and the `pdflatex` caller. The PoC versions were intended to demonstrate that it was possible and feasible to build the tool that was being designed. The PoC also helped later design work, helping to shape for example the data format and the TeX document settings. The PoC versions did not include a GUI, which was added much later in the development of the final program.

The development was started a few months before the start of the 2023 C-programming course, at the beginning of November 2022. A stable alpha version that allowed us to generate the instruction TeX files from JSON files containing assignment data was ready around the half way point of the course in early February 2023. The JSON files were still made manually at this point, as the development slowed down during the course. This was due to the creation of the assignments themselves. A buffer of around half of the required amount of new assignments was ready before the course, but the second half needed to be made during

the course which slowed development work on the tool. The PoC versions were used for the program, although they received significant upgrades.

The GUI was managed to be quite accurately translated from a plan into a functional GUI. However, some limitations were encountered during development. The ”Add Assignment” window, seen in Figure 9, was supposed to contain all the variations and their example runs, and the user would have been able to dynamically add them to the same window under a new collapsible header. This proved to be a bad idea, as this slowed down the program considerably when more than 3 variations or example runs were added. This is likely due to rendering being slowed down significantly when the program needed to draw new components dynamically. This theory is supported by the fact that once the components had been loaded, the program returned to its normal responsiveness. Due to this problem, it was decided that new variations and their example runs will be on a separate window, which can be opened from the ”Add Assignment” window. This does have the disadvantage of not showing all the data in one window but provides a far better user experience than the program slowing down after a few variations or example runs were added.

The previous part selection of an expanding assignment was thought to be a dropdown menu, as seen in the UI mockup in Figure 9, but during development it was realized that once the number of assignments and their variations would grow to possibly hundreds, it would be very impractical to select the previous part from a dropdown. Hence, the dropdown was switched to a listbox, with additional buttons to add, delete or show the previous part of the current assignment.

4.2.7 Testing

The program code was originally thought to have been tested via unit tests. However, this was later scrapped, as the UI and its helper functions that relied on the UI took up most of the program code. It was determined that unit tests would not be practical to use for testing the UI. Time constraints prevented non-UI components to be tested via unit tests. Hence, only manual testing was used to produce the MVP, but the manual testing was made extensive, using different data sets and orders of operations to determine whether functions in the tool worked individually and together.

L2 Assignments

- Reading files

Read course book chapter 2 and return the assignments to the autograder.

Contents

L2A1: File Reading

Make a C-program to read the contents of a file and print it to the terminal. Use `fopen()` to open the file and `fgets()` to read the contents.

Example from input file 'Inputfile.txt':

```
This is the first line.
And this is the 2nd line.
```

Example run 1

Inputs:

```
Inputfile.txt
```

Output:

```
Input filename: Inputfile.txt
File contents:
This is the first line.
And this is the 2nd line.
```

Example solution 'ExampleCode.c':

```c
/*20230329, Rami Saarivuori, L02T1 Variation A*/
/*For study use only, LUT (c) 2023*/
#include<stdio.h>
#include<stdlib.h>
#define SIZE 255

int main(void) {
    char Line[SIZE];
    char Filename[SIZE];
    FILE *ReadFile;
    printf("Input filename: ");
    scanf("%s", Filename);
    if ((ReadFile = fopen(Filename, "r")) == NULL) {
```


Figure 10: An example of a generated assignment instruction PDF. Note that this is only a demo and the assignment in the example is not actually in use as a course assignment.

5 RESULTS

The assignments created with this thesis were used in the Spring 2023 implementation of the *"Principles of C Programming"* course at LUT. An early alpha version of Mímir was tested during that course implementation as well, to manage the assignments created and from the fourth course week onward, to compile the instruction papers from the assignments. To gather data on the suitability of the assignments and the instruction papers, multiple questionnaires were conducted on the course. This chapter presents all the results as well as the questions used in the questionnaires.

5.1 Mímir

This section documents the finished version of the tool as well as interviews conducted. These interviews were conducted to evaluate the tool with outside experts.

5.1.1 Finished version

The version associated with this thesis (`v0.9.6`), as well as development history can be found from the tool's GitHub repository (Saarivuori 2023). Figures 11 and 12 show example screenshots from the finished version of the tool. The finished UI differs slightly from the mockups shown in the previous chapter, due to problems encountered and refinement that was necessary to be made for at least an adequate user experience when using the tool. It should be noted that while an earlier version of the tool was used during the Spring 2023 C programming course, the style and structure of the instruction paper stayed the same from that version onward. The differences between these two versions is that the UI and the assignment selection algorithm were finished in the later version, as the assignment selection and data input were done manually in the earlier version.

5.1.2 Expert interviews

To evaluate the usefulness and the potential from a course instructor's perspective, experts were invited to a meeting where a demo of the tool was presented. Six experts participated in this meeting. All six experts had a background or experience from teaching programming focused courses at university level. During the meeting, the experts were asked background questions before the demo, and questions regarding the tool were asked after the demo. The interview was conducted in Finnish and the results and questions are translated. In Table 5 the questions asked from the experts can be seen. Questions 1-5 were asked before the demo and questions 6-7 after the demo.

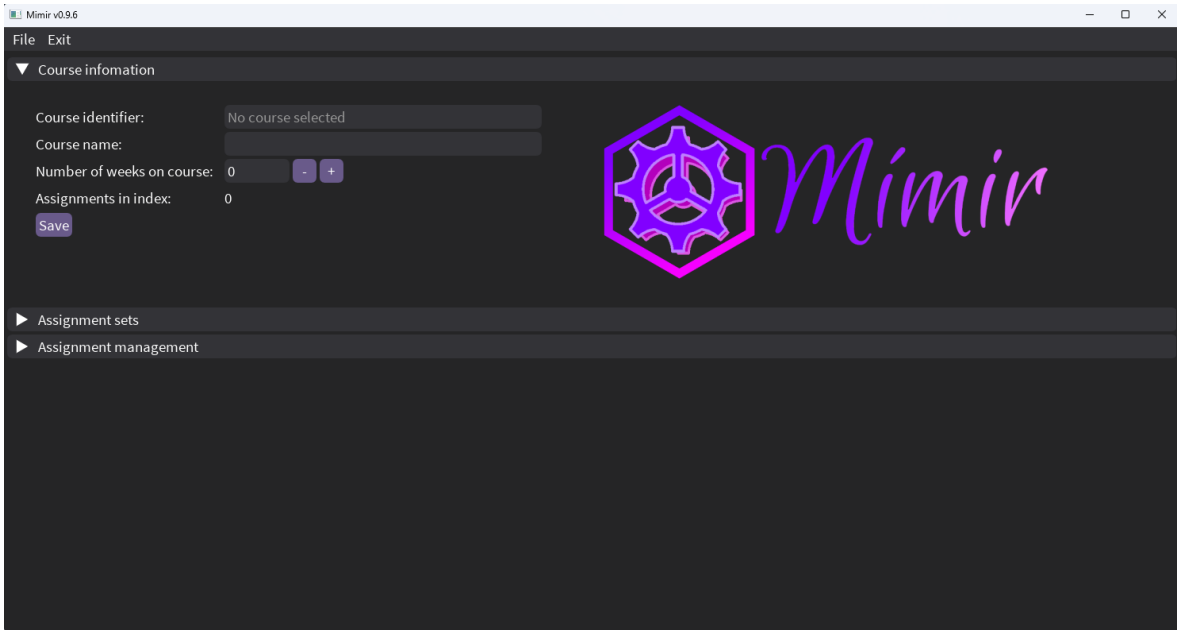


Figure 11: A screenshot from the finished main window UI of the MVP version of the tool

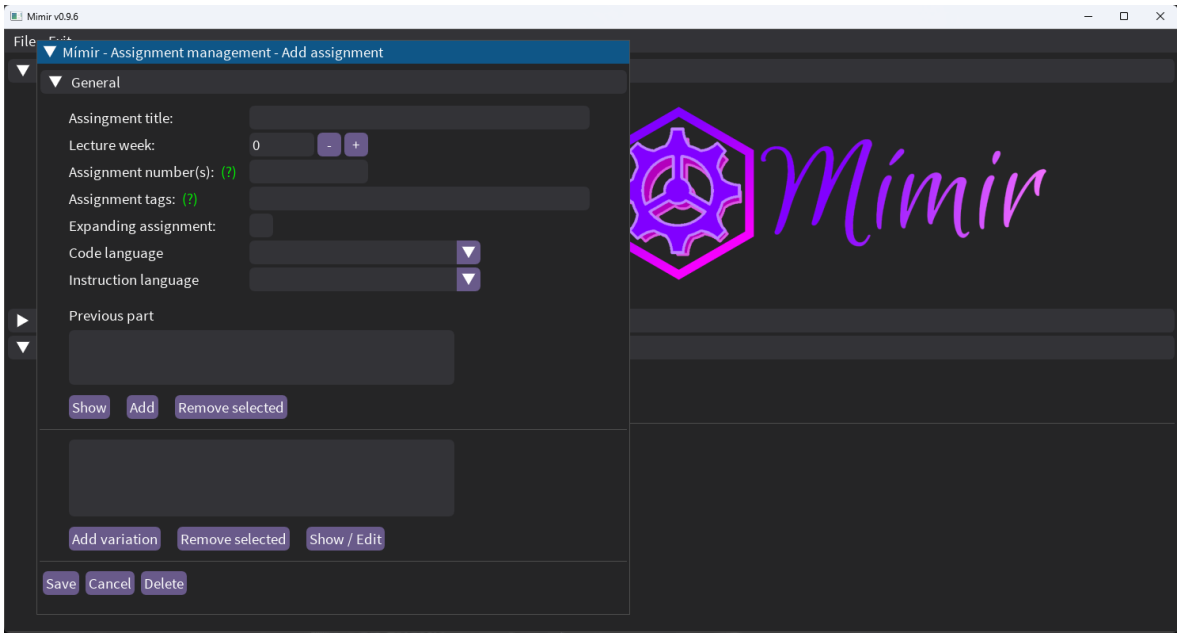


Figure 12: A screenshot from the finished assignment input window UI of the MVP version of the tool

When asked about the assignment instruction papers with questions 2 and 3, all experts said they used templates or simply copied the old instructions to a new page. Half of the experts were using only the web pages on the course platform though and not assignment instructions in PDF format like the other half, making a simple copy of the assignments easier. In regards to the style referred to in question 4, the experts all said that they tried to keep the style

Table 5: The questions that were asked from the experts during the interview.

Question no.	Question
1	Do you have experience in teaching programming?
2	Have you done assignment instruction papers to courses?
3	If yes to Q2, have you used a template or started from scratch each time?
4	Have you paid attention to the style or the consistency of the assignment instruction papers?
5	How much time does the compilation of one instruction paper take if the assignments are already done?
6	What would you change in the tool?
7	Do you think that the tool is suitable for you course(s)?

consistent within their own courses but noted that the style and consistency across teachers was something that the department itself had had discussions about. Expert 1 also noted that if a problem was raised in course feedback, the fix was applied to all places that it was encountered in, such as missing information. Expert 2 noted that he used the same templates as Expert 1.

The experts had more varied answers to question 5. Expert 3 said that the compilation took no time, as he had all the instructions on a web page on the course platform that was easy to copy to another course within the platform. Expert 4 said that if the assignments were done, it usually took about an hour to format the text correctly to the template. Expert 6 was within the same range, stating that it took him about one to two hours to setup the assignment instruction and return pages, if it was necessary to make new assignments. Expert 2 said that as he had a completely new course, it could take anywhere between half to a full working day to make the instruction papers and configure the auto-grader for those assignments. Expert 5 was in agreement with Expert 2.

After the demo was done, questions 6 and 7 were asked. The experts raised many potential use cases for the tool, some that were not thought of in the original plan for the tool. Expert 5 saw that the tool had a potential use case in exam management, as those were changed more frequently, as well as needing multiple variants per exam. The expert said that the frequency with which he changed the course assignments was not enough to warrant the use of the tool for that. Expert 2 said that he saw potential for the tool if a bank of assignments was already made, rather than for example in his case where he was doing new assignments for a new course. However, he raised concerns on how much time the adoption of the tool would take if the instructor already had a large bank of assignments ready. Expert 4 was in agreement with the concern and said that while the tool seemed to be quite easy to use and learn, the nested menus could be tedious to navigate through when inputting assignments.

Expert 6 raised the idea of implementing a feature that could export the assignments in HTML format, which would widen the potential user base to those who where using only web pages to display their assignment instructions. This would eliminate the dependency from LaTeX, so the user would not have to install or manage a separate LaTeX installation that is currently necessary in the tool. Regarding the concern of adoption that was raised by Experts 2 and 4, Expert 6 suggested implementing an import feature that could take data in a structured file format, such as a Comma-Separated Values (CSV) file. Expert 6 also had the idea of adding support for images to the tool, suggesting a simple command to place the image in the text by the user and then supplying the image separately to the program by, for example, importing it like the other files. As some of the experts said they used a different format or different names for the different structures in the instruction paper, they suggested that these could be able to be configured by the instructor to their liking.

The overall sentiment was that all of the experts saw potential for the tool. Many of them also saw more potential if the tool could save and manage the test cases used for the assignments in the auto-grader that is used in programming courses at the university. They also concluded that it would not solve all problems but was a step in the right direction.

5.2 Assignment Instruction Papers

This section presents the results from the assignment instructions. Results from both the visual style of the instruction papers that were created with Mímir, and how well the students understood the text of the assignment instructions, are shown.

5.2.1 Visual Style

At the end of the aforementioned course, students were asked to evaluate the visual style of the assignment instruction papers that were generated using an early version of Mímir. The first three weeks of the course used an old style instruction paper design that was manually made. Week 4 used an alpha version with some features that were added in week 5, hence it has not been included in the comparison. From Table 6 we can see the questions that were asked from students. The results are listed in Figure 13. From the results we can see that students overall had positive opinions about the readability of the design. In questions 2 and 6-10 the answers are overwhelmingly positive, with an average of 82.8% answering on the positive side of the answer scale. Overall sentiment in comparison with the old style is also positive, with 33.8% of students considering it somewhat or completely better than the old style in Question 1. It should be noted though that most were still indifferent to this, as 43% of students answered ”Neither agree nor disagree”. There were very little negative answers, with only one completely disagreeing with the statement in Q1 and Q11. Most negative

answers hence were ”Somewhat disagree”, of which there were consistently less than 10, with the exception of Q1, which had 11 of the aforementioned answers.

Table 6: Questions that were asked from the students at the end of the 2023 course regarding the visual style of the instruction papers generated by Mímir.

Question No.	Description
Q1	The appearance of the instruction paper is clearer on course weeks 5-7 than on 1-3
Q2	Overall look is clear and readable
Q3	Table of contents is useful
Q4	Subroutines separated into bullet points help conceptualize the program
Q5	Separated inputs help test my own program
Q6	The fonts used are readable
Q7	The font sizes used are good enough
Q8	Program example run, inputs and examples from input and result data are distinct from each other
Q9	The colored highlights on keywords help read the instructions
Q10	The background colours in inputs and example runs help with readability
Q11	Coloured highlights are better than only bolded highlights

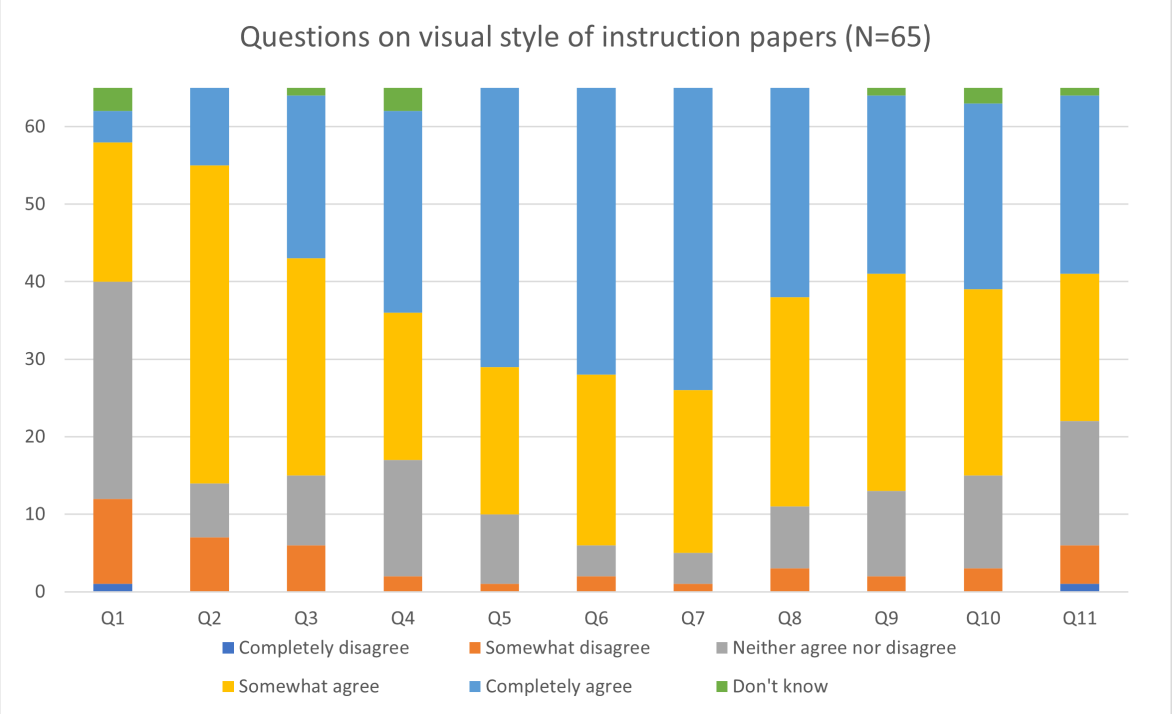


Figure 13: The results from the questions presented in Table 6.

5.2.2 Instruction Formatting

As can be seen from Figure 14, the majority of students found that the instructions to the assignments were either ”Quite Easy” or ”Very Easy”. The amount of students that found

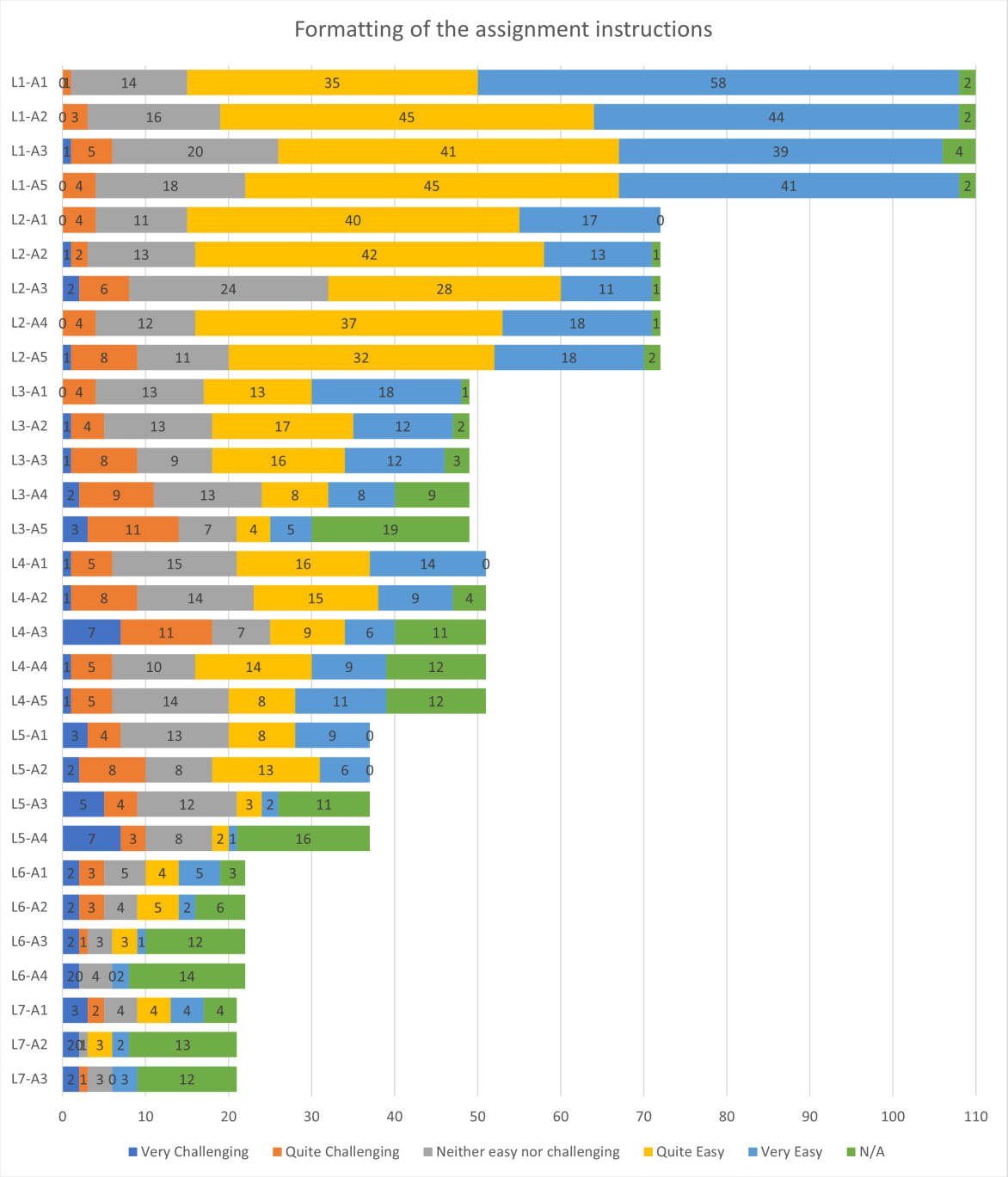


Figure 14: Results from asking how challenging the formatting of the assignment instructions were.

them challenging on some level is very low on the first few weeks, but rise slightly towards the end of the course. Their absolute amounts stay low though, with the amount of students who found them easy or neither easy nor challenging decreased steadily as the number of answers also decreased. ”N/A” answers also saw an increase towards the end of the course. There are only small spikes in difficulty that can be seen in L2-A3, L3-A5, L4-A4 and L5-A4.

5.3 Assignment Submissions

In the same course, new assignments created in this thesis were used. From Figure 15 it can be seen that while the first two weeks the successful submission rate was consistently slightly above the average of previous years, it began fluctuating more starting from the third week compared to the average of previous years. Some assignments did considerably better, such as L2-A5 and L7-A3, but there were a considerable amount of assignments were the successful submission rate was much lower than the average of previous years. Such examples are L3-A5, L4-A3 and L5-A4. Overall, the average submissions dropped from 160.7 per assignment in 2019-2022 to 151.5 in 2023. However, if we instead focus on comparing year-to-year instead of averaging the previous years together, 2023 is in line with 2021, where the average submissions per assignment was 151.9 and higher than 2020, were the number was 146.5. 2019 and 2022 had averages of 174.4 and 170.1, respectively

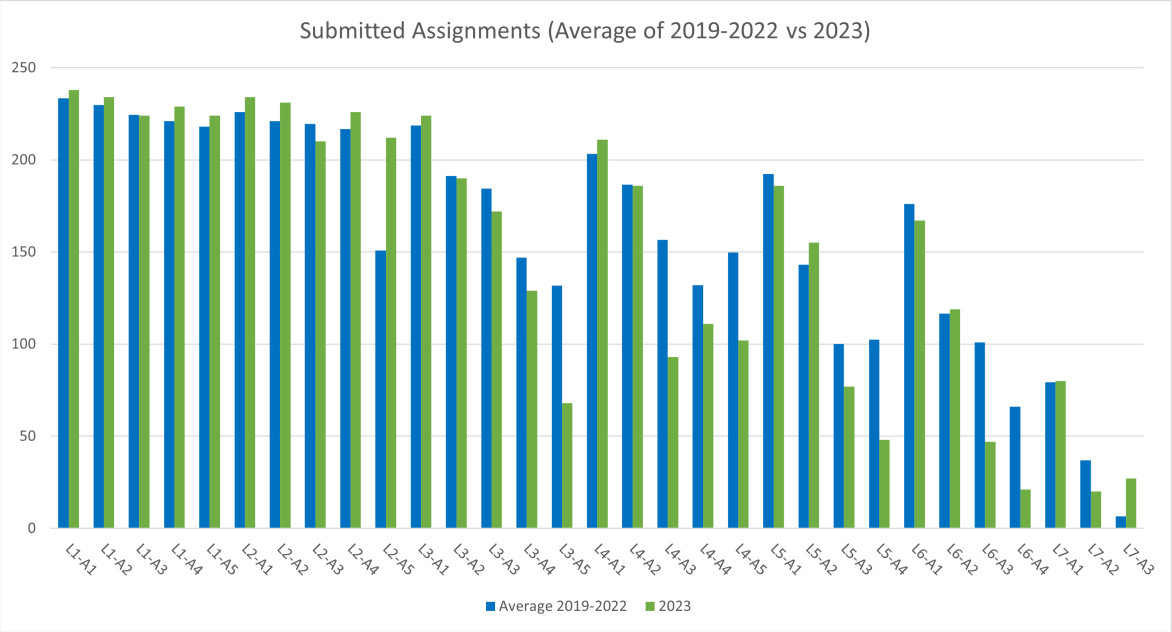


Figure 15: Successful assignment submissions. The values of 2019 to 2022 have been averaged and are compared to 2023.

The overall trend in the assignment completion, despite the fluctuations compared to earlier years, was still very much in line with previous course implementations. After the third week, students became more selective in what assignments they try or complete. Figure 16 shows that the amount of students who even start the assignments begins to drops sharply at the start of the third week. On and after the third week, the line of students starting the assignment begins to resemble a camel-hump pattern, where the first and the second assignments of the week are the most popular, while the subsequent assignments fall below half of the number of students starting them compared to the first two assignments. During the course, the drop

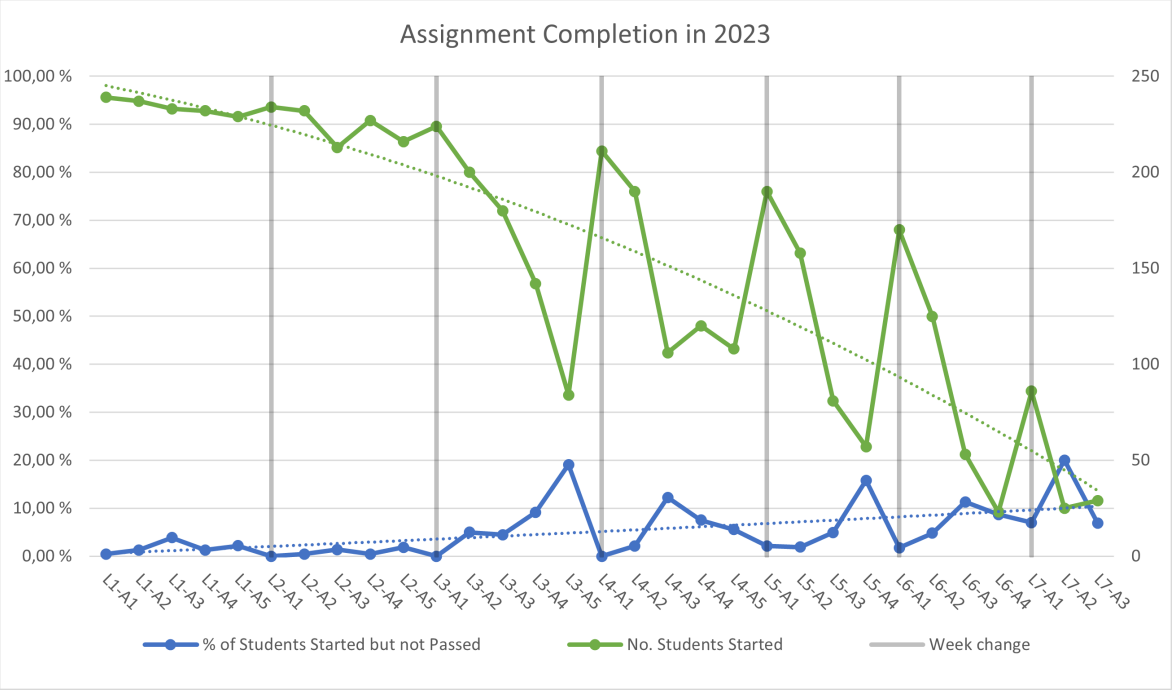


Figure 16: Assignment completion with new assignments in the 2023 course.

in how many start the first assignment is not as sharp from week 1 to 6 as it is from week 6 to 7. Between weeks one and six, the number only drops 69 students from 239 to 170, while the drop is 84 students in just one week from week six to week seven.

Some assignments had less activity than what could be expected based on earlier years, especially L3-A5. In Figure 16 it can be seen that only around 35% of students (84 students) even started the assignment, and of those, 19% of them (16 students) did not finish it. This was a noticeable spike in assignment completion, because while the later assignments also have a small amount of students starting them, they are in line with earlier years and the overall trend of less students actively participating in the last weeks of the course. This can also be seen from Figure 15, as the amount of students completing this assignment was over half less than the average of previous years. Though interestingly, the difficulty of this assignment cannot be seen from Figure 17, as there is no distinct spike in the amount of submissions per student. A similar assignment was L5-A4, which had a similar topic like L3-A5. The number of students who started and those who did not end up completing it also had visible spikes or drops that deviate from the overall trend. L5-A4 does have more elevated levels of submission as seen from Figure 17, though not by a big amount.

The number of submissions per student varies a lot over the course, as can be seen from Figure 17. There were multiple spikes, but they do not seem to follow a pattern all the time. A small pattern can be found between assignments L4-A3, L5-A3, L6-A3 and L7-A2. These

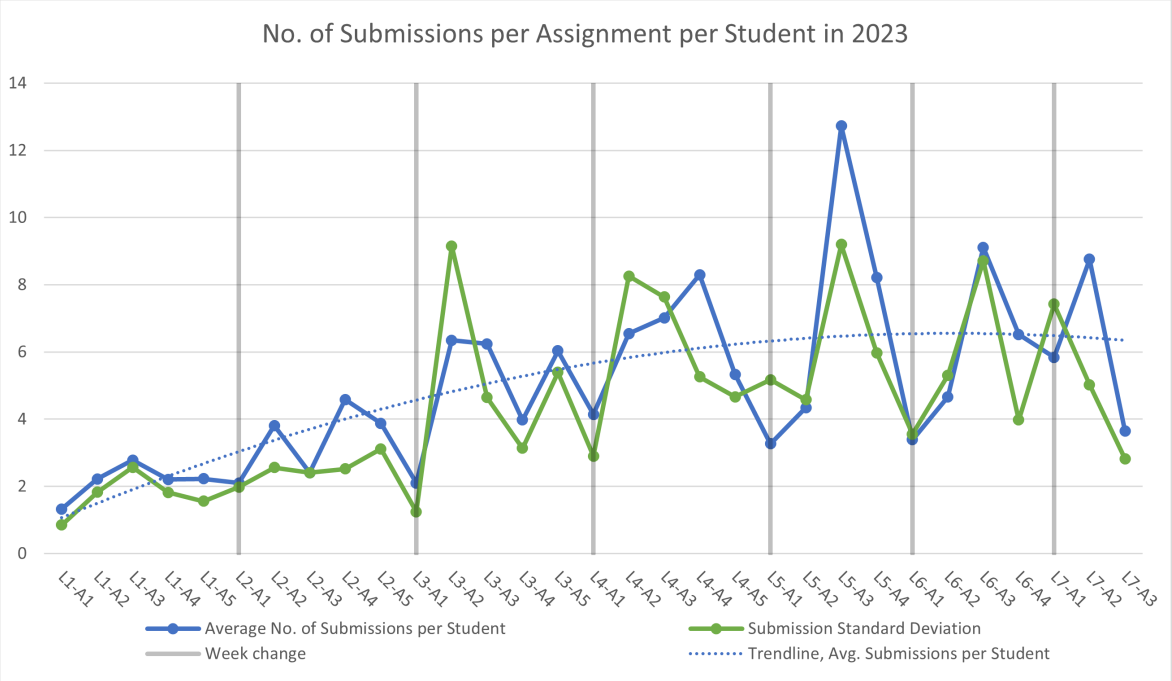


Figure 17: Average submissions per assignment and the standard deviation of the number of submissions per assignment per student in the 2023 course using new assignments.

assignments all were dependent on the previous assignment. Week 4 can also be seen as having elevated number of submissions throughout the week, with the exception of the first assignment that week, which had a considerably lower amount of submissions per student than others that week. L1-A1 had the lowest, but L3-A1 and L2-A1 were not far behind, all having less or around 2 submissions per student on average, with the standard deviation also around or below 2. These numbers cannot be compared to previous years however, as no data of this kind exists from previous years.

The overall trend of assignment completion does not majorly differ from previous years as can be seen by comparing Figures 1 and 15. The students are very active in the first three weeks, after which they become more selective in what assignments they try to do. As can be seen from the figures, the first and second assignments of the week are usually popular, as they are usually the easiest from that week, teaching the basics of the new concepts introduced in the lecture that week. It should be also noted that the minimum number of assignments that need to be completed to pass the course is 15, so those who are only looking to pass with the bare minimum seem to not put extra effort into the assignments.

5.4 Difficulty of Topics and Assignment Goals

The perceived difficulty in assignments does not translate into any significant drops in assignment understanding though. Each week during the course, students were asked to answer the question ”The aim of assignment X was understandable”, where X was the number of an assignment between 1 and 5, though in lecture 5 and 6 it was between 1-4 and in lecture 7 between 1-3. This questionnaire was given to the students the following week. Figure 18 shows the results from each week compiled. From the figure it can be seen that the answers are mostly positive throughout the course, though the level of agreement with the statement drops quite consistently as the course progressed. Each assignment got an average of over 3.5, except three assignments which scored just below that. It should be noted that the students could also answer ”N/A”, which meant that they did not do that assignment. While the amount of these do increase in the last two weeks of the course, the figure is in percentages, obfuscating the true numbers in a quick glance. The largest ”N/A” answers are L3-A5, L5-A4, L6-A4 and L7-A3 with 12, 14, 11 and 10 students respectively. The amount of students who answered these questionnaires each week can be seen under the title of Figure 18.

In the same questionnaire as the aforementioned instruction understanding question, each week students were asked to rate the topics in that weeks assignments from 1-5, where 1 was ”Very Easy” and 5 ”Very Challenging”. Students were also able to respond ”N/A”, which was meant to mean that the student did not do that assignment. These questionnaires were conducted to gather more data from the students themselves as to what topics they perceived to be difficult. Getting this data from the students is necessary as the submission rates only tell one side of the story. It also helps understand what the difficult topics are so they can be taken into account in teaching. The results from these questionnaires are outlined in the following subsections.

5.4.1 Assignment Goal Understanding

As it can be seen from Figure 18, the assignments had an overwhelmingly positive number of answers to the question whether the instructions and goal of the assignments were understandable. As noted in the per-week questionnaires, the first two assignments of the week do show as lower counts of ”N/A” answers here too. The amount of those answers is also a bit smaller than in the topic specific questions. There were very little disagreeing answers throughout the course, with the largest number in L2-A5. The ”Neither agree nor disagree” answers stayed surprisingly low as well, with the largest amount in L4-A3. The amounts of students who answered the questionnaires are the same as in the topic specific questions, but are also listed in the top of Figure 18.

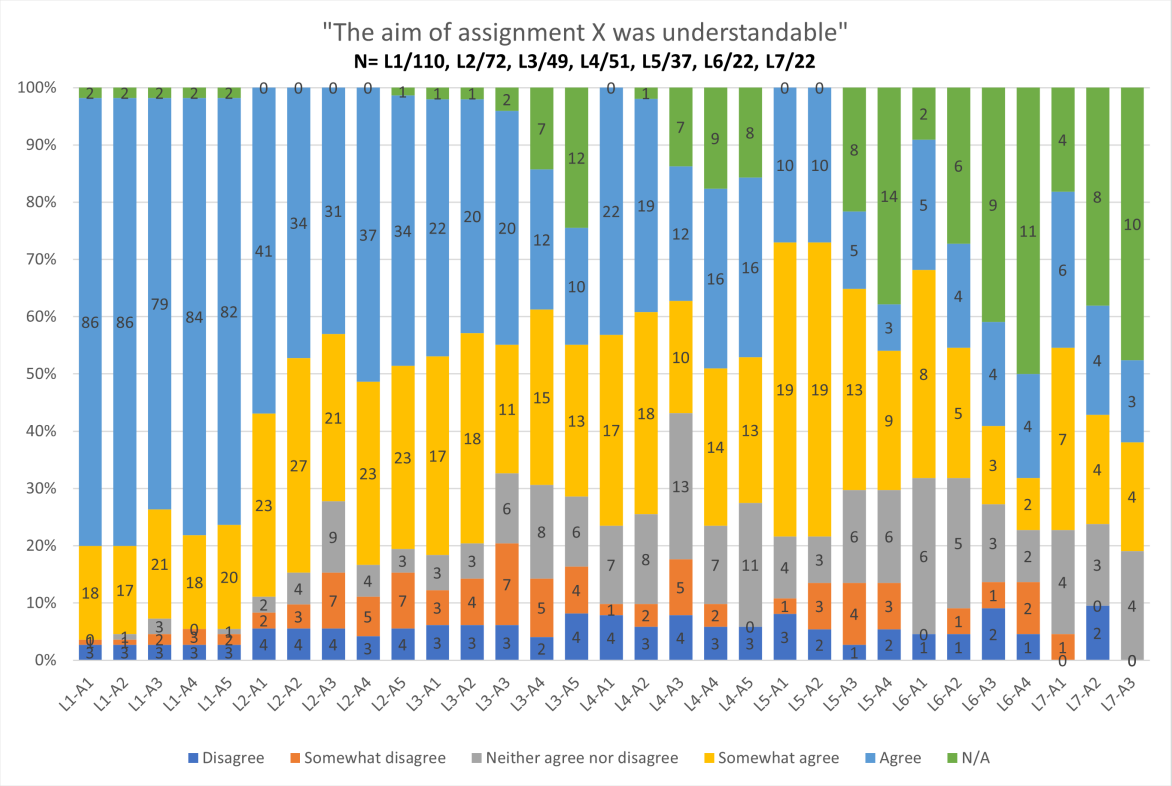


Figure 18: Results from a weekly question of how understandable the assignment goals were.

5.4.2 Week 1

The first week's answers do not have much surprise, with a significant majority of the answers being either Very Easy or Quite Easy, and only a handful responding that the topic was difficult as seen from Figure 19. These were the addition operators (++) in L1-A3 and character array in L1-A4. The amount of answers in the middle of the scale stayed between 9-22 students, with an average of 14.7. A few students also wrote about L1-A3 in the open feedback section, mostly noting that the instructions were a bit difficult to understand. One such answer is *"The design of the L1-A3 assignment was so difficult to understand at the beginning that it took a while to understand the assignment as a whole. When I calmly read through the assignment several times, I got to the bottom of what was really required to do."* There were also answers that praised the first weeks assignments for their simplicity, such as *"It's good that the tasks are easy and small at first. Experiences of success help to get excited, and insurmountable problems cannot arise at the beginning."*

5.4.3 Week 2

The second week saw small increases in "Very Challenging" answers, though the "Very Easy" and "Quite Easy" answers still had the significant majority, as seen from Figure 20. As

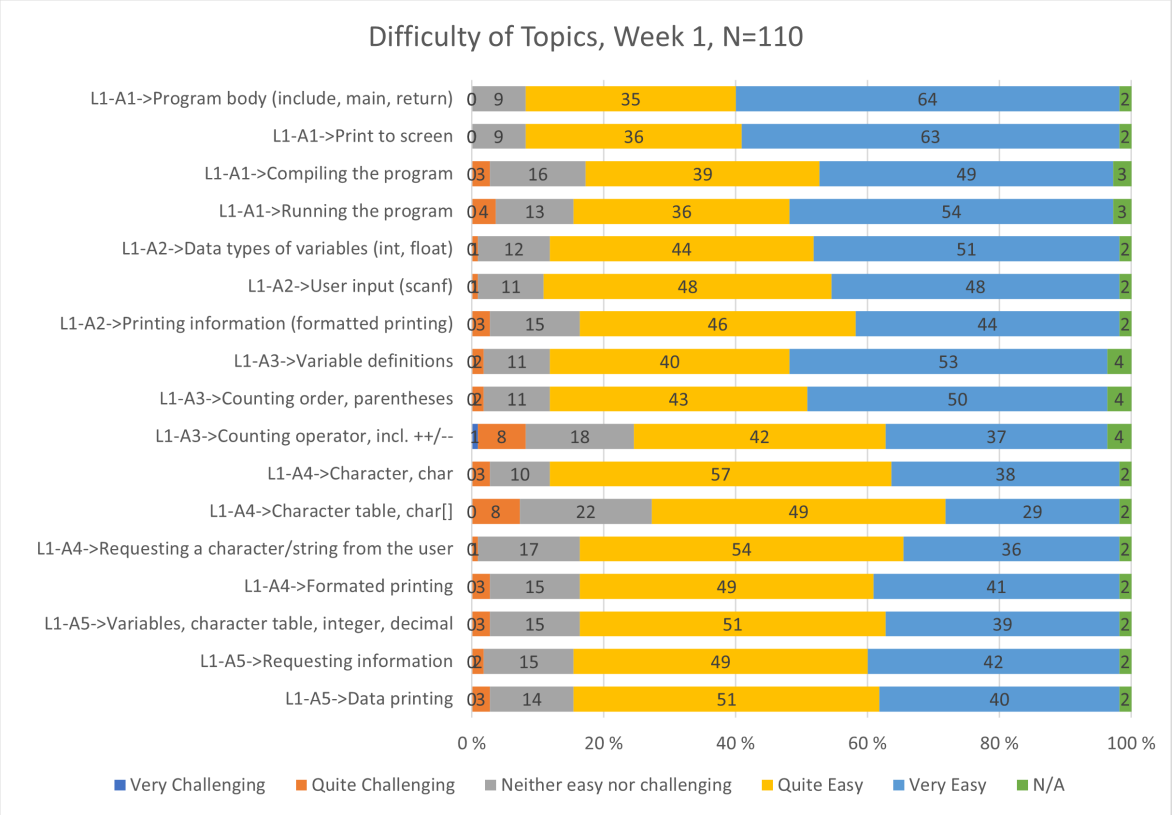


Figure 19: Difficulty of assignment topics, week 1.

mentioned earlier, L2-A3 and the ASCII-table were seen as difficult, and this questionnaire also corroborates that fact, as it saw an increase in negative answers and a small decline in very positive answers. The other topics in the same assignments also had these effects, though on a lesser scale. On this week, the `for`-loop also had some very negative answers, though there were only 2 out of 72 students. The answers on the middle of the scale on the second week stayed between 6-24 this time, with an average of 12.5, a decrease from the previous week. Most open feedback answers related to a problem with the auto-grader used on the course on the second week, rather than any difficulty in the assignments themselves.

5.4.4 Week 3

In the open feedback section of the third week, there were a few comments about there being a major jump in difficulty of the assignments compared to the second week. This does show from the data as well, as the third week, presented in Figure 21, begun to see an increase in ”N/A” answers as well as a large jump in negative ones. The apparent difficulty does show in the submissions as well. In Figure 16, there is a major downward trend in the third week from the first assignment to the fifth assignment of that week. Though if this is compared to the previous years, seen in Figure 15, it can be seen that the first assignment has more

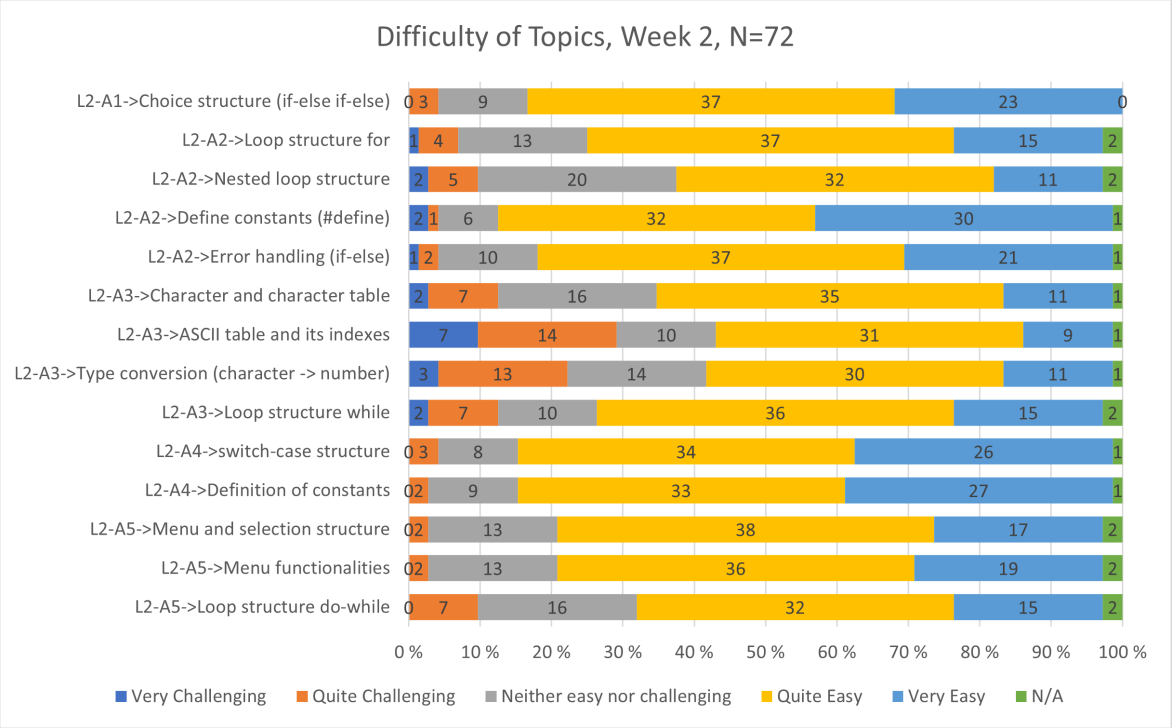


Figure 20: Difficulty of assignment topics, week 2.

submissions, the second about the same amount and only from the third assignment onward there are less submissions than in previous years. The topics of the first two assignments on the third week still follow the pattern of the previous weeks, but from the third assignment onward, the amount of ”N/A” and negative answers begin to increase rapidly. The question on pointers was the first to receive more negative than positive answers. Very close to having more negative than positive was also the topic of return values from functions. The answers in the middle of the scale on the third week were between 6-14, with an average of 9.7, a decrease again from the previous week.

5.4.5 Week 4

Figure 22 shows the results from the topics of week 4. Again as seen in the results of the previous weeks, the first two assignments have very little negative answers, though the proportion of ”Very Easy” answers saw a decline in favor of ”Quite Easy”. The third assignment seems to continue a trend from earlier weeks where the topics of this assignment received more negative answers than any other topics that week. The fourth week was also an exception where the use of `struct` was asked twice, once in questions relating to A1 and again in A3. In the answers relating to A3, the amount of ”N/A” answers was much higher, as well as the negative answers. Negative answers also were greater in number than positive.

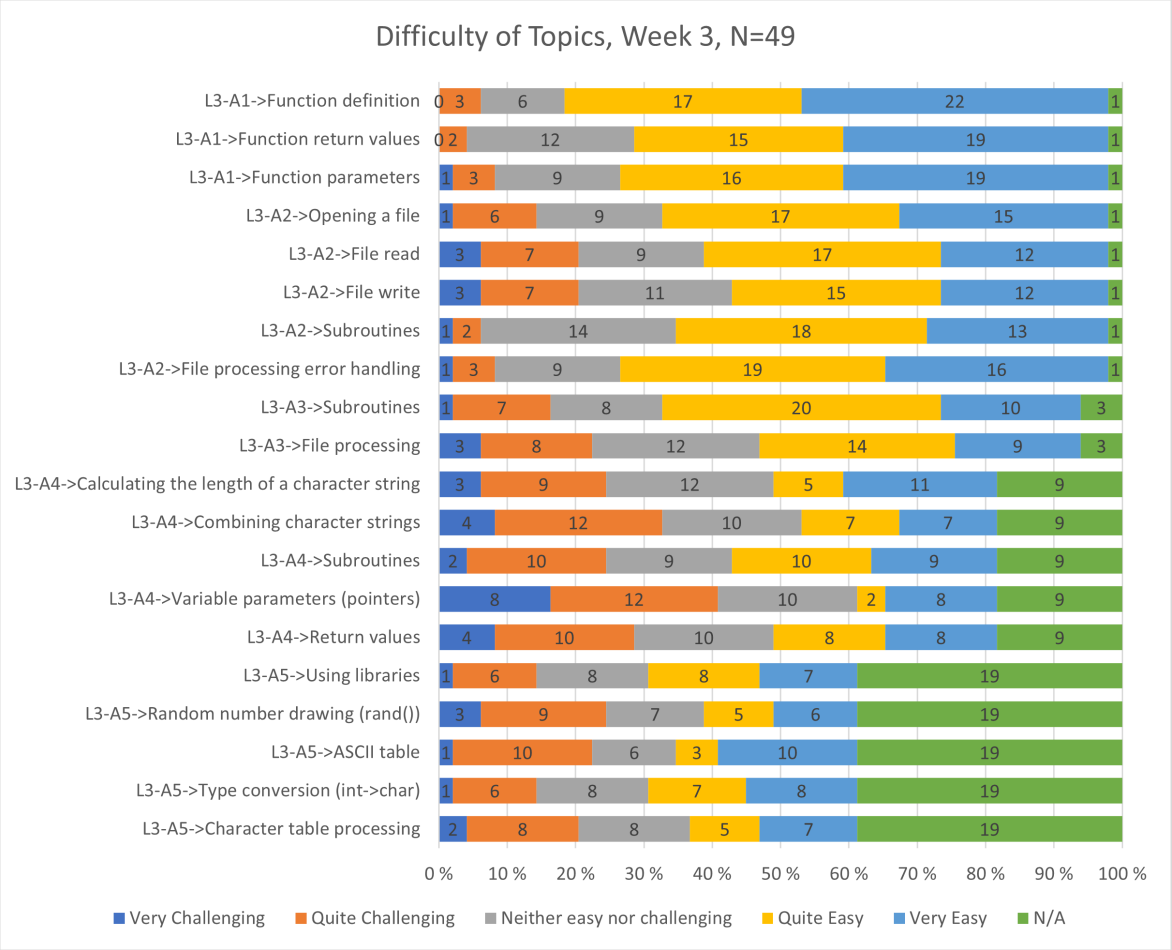


Figure 21: Difficulty of assignment topics, week 3.

The amount of people answering the middle of the scale stayed relatively same as week 3, between 6 and 15, with the average being 9.8.

5.4.6 Week 5

The fifth week was the first to have only four assignments. This week, for the first time, saw an increase of negative answers across the board, even in the first two assignments which can be seen from Figure 23. Notably, the amount of ”N/A” answers on the fourth assignment count almost half of the responses. The last two assignments saw also a notable increase in the proportion of very negative answers compared to positive answers. They only outweigh the positive answers on three occasions, ”Functions, their parameters and return values”, ”Dynamic memory control” and the formatting of the instructions in L5-A4.

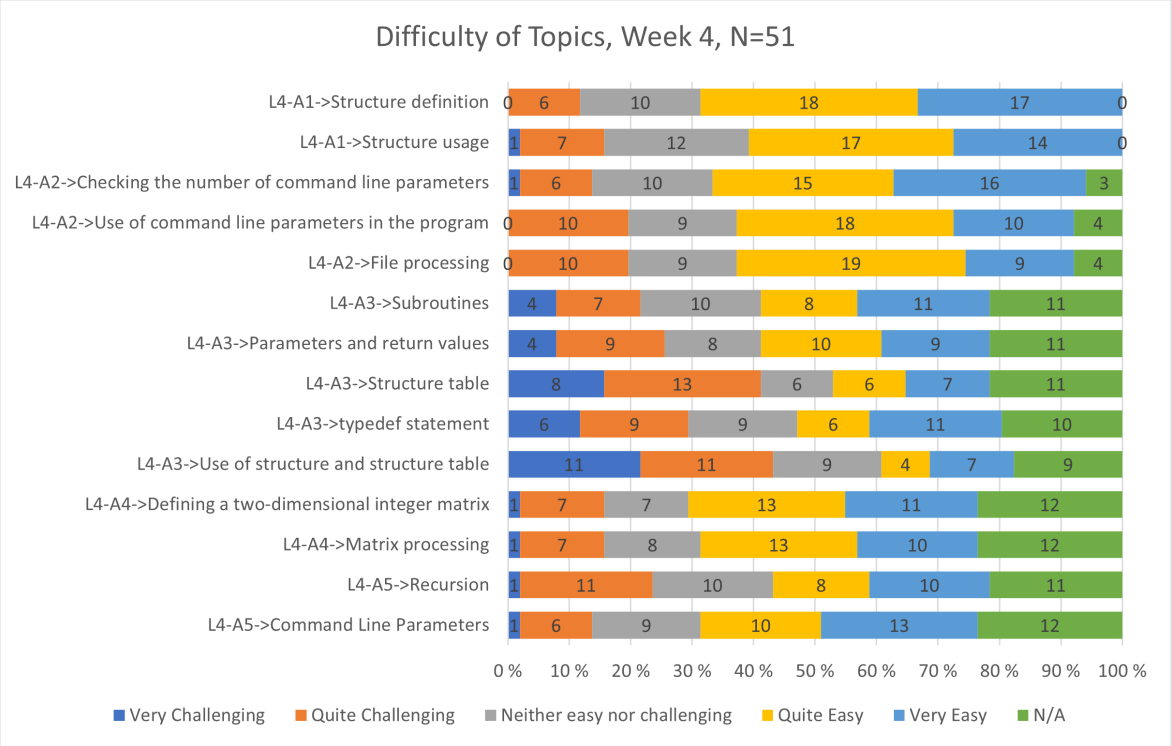


Figure 22: Difficulty of assignment topics, week 4.

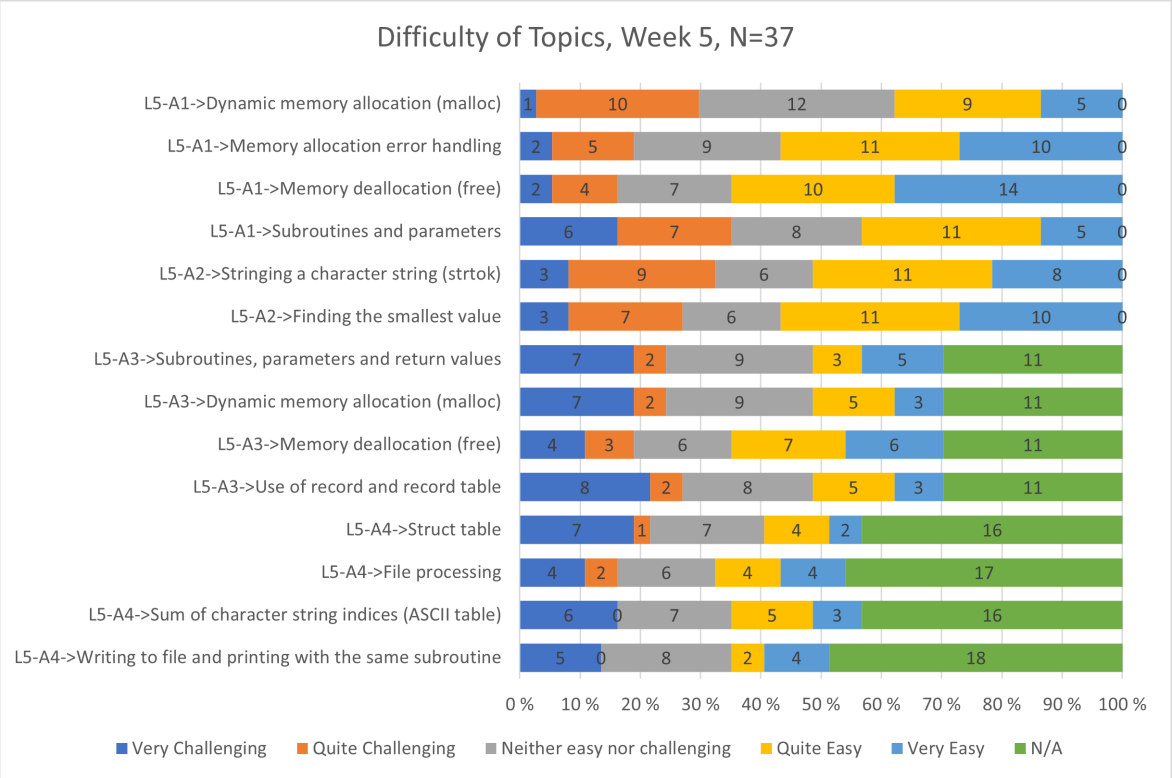


Figure 23: Difficulty of assignment topics, week 5.

5.4.7 Weeks 6 & 7

The number of answers to the topic questions on the sixth week were only 22 students out of the average of 92 and on the seventh week, 22 out of 46 students that started the assignments on those weeks. Hence, the answers to these questionnaires cannot be considered as accurate as the previous weeks, though their accuracy also decreases as the number of answers decreases. The answers to the topics on the sixth week, seen in Figure 24, are quite tied overall between positive and negative answers, with the positive having slightly more answers on the first and second assignments. The number of answers in the middle of the scale are quite low though, only between 1-6 answers with an average of 3.1. For the last assignment, the answers are almost all tied. This assignment had the topic as "The use of `struct tm`". The seventh week, seen in Figure 25, is quite similar to the sixth week, with the exception of the middle of the scale, which had even less answers than the sixth week, between 0-4 answers. On the seventh week, the second and third assignment had almost 60% of answers be "N/A". Another difference between the sixth and seventh week is also the fact that there were a bit less "Very Challenging" answers, with some opting for the "Quite Challenging" answer.

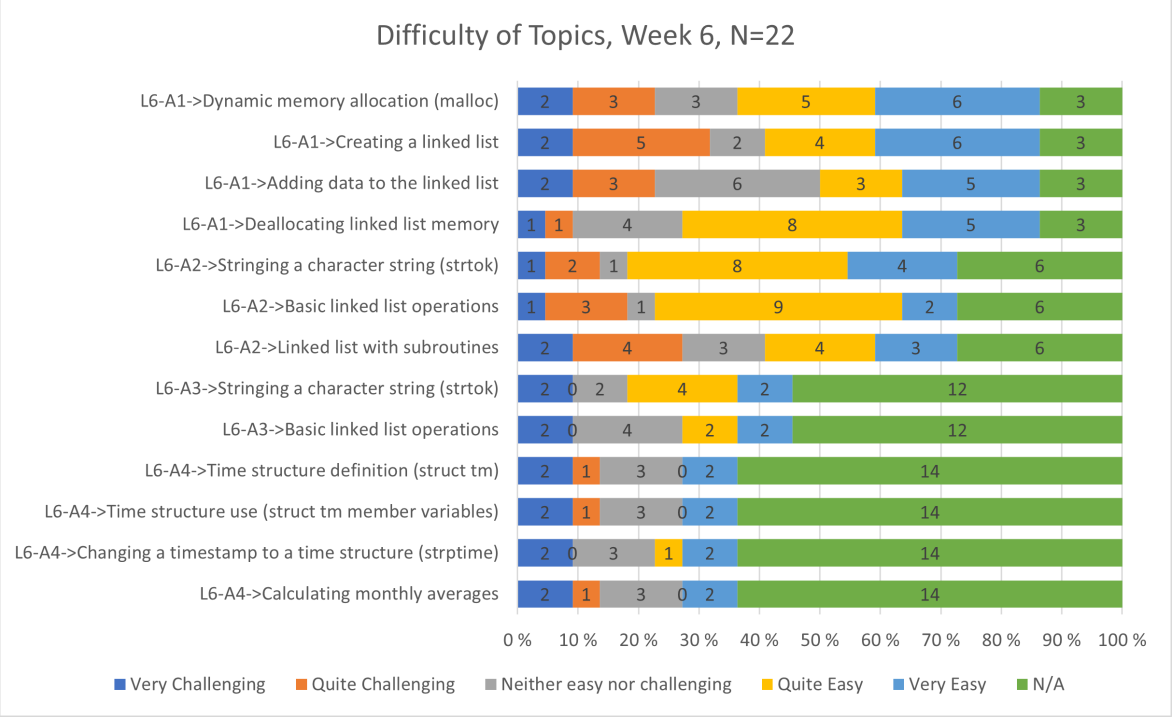


Figure 24: Difficulty of assignment topics, week 6.

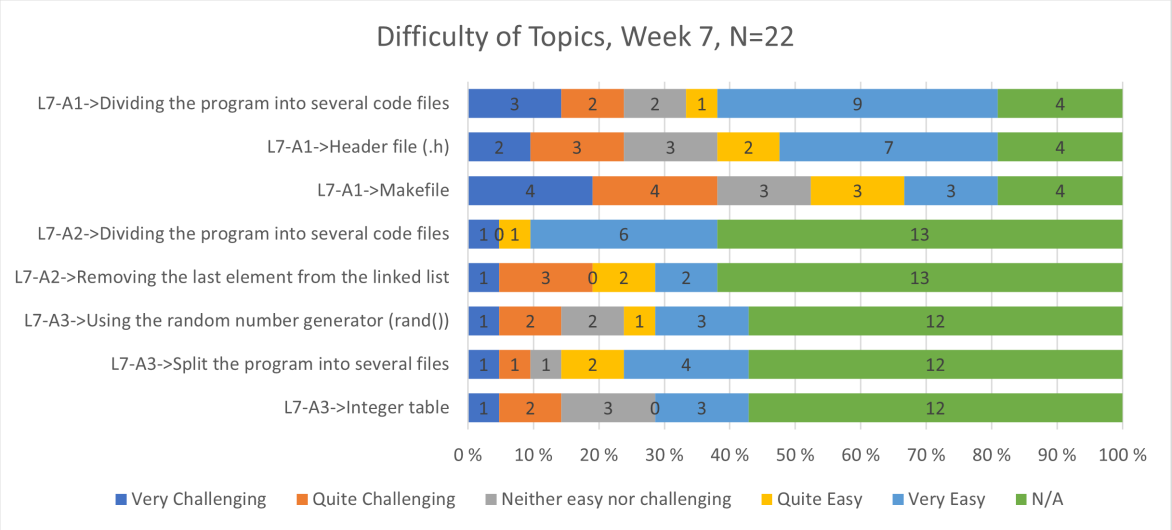


Figure 25: Difficulty of assignment topics, week 7.

5.5 Time estimations

When comparing the time spent during the course each week in Figure 26 it can be seen that the overall trend of time spent during the course was noticeably higher than in the previous year. It is important to note though that the students were asked for an *overall* estimation of the time spent with the course for each week, which includes lectures, weekly assignments and self-study. The final work was released in lecture 7, hence its number is not strictly comparable to other weeks. The trend of an increase in difficulty of the concepts on the first weeks to the reduction in the number of assignments in weeks 5-7 can be clearly seen from the figure though.

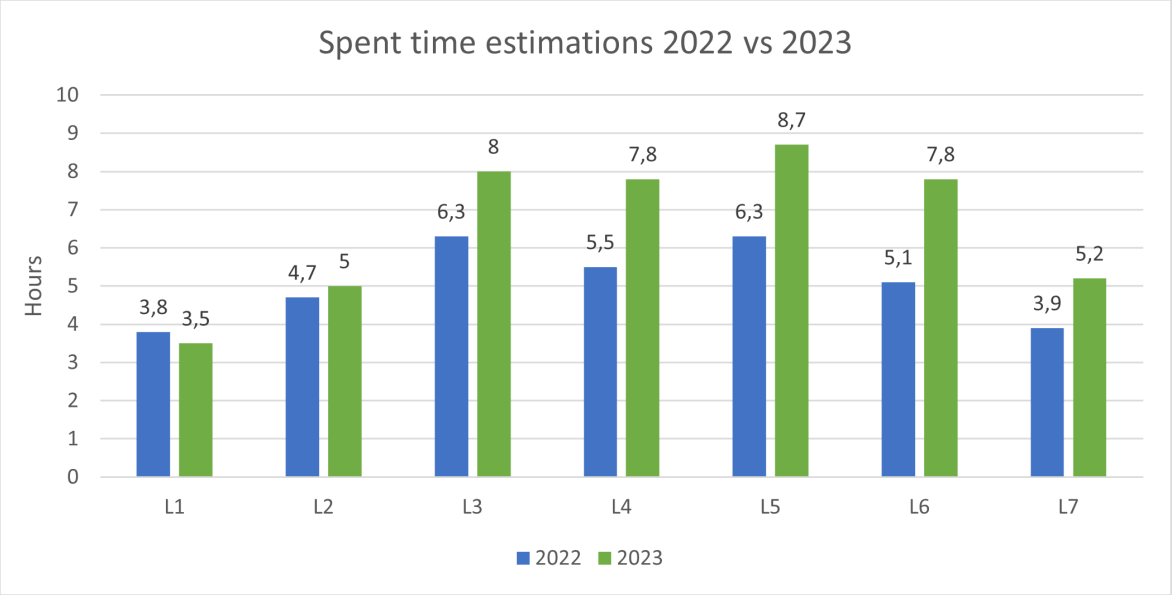


Figure 26: The average of estimations of time spent with the course students gave for each week of the course in 2022 and 2023.

5.6 Course Feedback

At the end of the course, in addition to the technical questionnaire that had the questions listed in Table 6, another questionnaire was presented to the students. This was a standard course feedback that is used in all courses at LUT. 26,4% of students (64 students) answered this questionnaire out of the 242 that started the course. Most of the questions are out of scope for this study, like questions on course motivation and teaching methods, but the open-ended questions have some answers relating to the assignments on the course. The first of these, titled ”What aspects of the course most need improvement?” received 42 non-empty answers. Of these 42, 14 answers related directly to assignments and are hence applicable to this study. The answers were translated from Finnish to English. Some examples of these are:

- ”Each assignment should be started ’from a clean slate’. The assignment instructions had a lot of errors.”
- ”In my opinion, the instructions to assignments were a bit hard to understand.”
- ”The assignments could have example solutions.”
- ”It would be great if even more topics were applied to the tasks that are civilizing or that one could identify with. For example, in the python course, we learned about the price of electricity in the practical work, and in the weekly assignments, e.g. car operating costs. Subjects that educate seem to bring added value to doing tasks.”

- ”Learning the memory control was left to the course project because memory leaks were not tested in the weekly assignments.”
- ”The assignment instructions were unclear at times – The course could potentially cover more than just reading files, analyzing, etc. For example, coding a simple game could be refreshing.”
- ”The assignments are too broad.”
- ”The difficulty of the weekly assignments are too high in my opinion.”
- ”Some assignment instructions were unclear”
- ”The assignments on the C course were too long at the end of the course.”

As a counter-question to the previous, a question titled ”What was best about this course?”. There were 40 non-empty answers in total, of which 9 are directly related to assignments. The answers are, again, translated from Finnish to English. Some examples of the answers:

- ”Examples to the assignments.”
- ”The assignments were good and there were enough of them.”
- ”An adequate number of assignments.”
- ”The assignments were good in my opinion.”
- ”The assignments.”
- ”Scheduled weekly assignments.”

By comparing these two lists and the amount of answers, it can be seen that while there are more answers to the question of what could be improved upon relating the assignments, they are much more scattered in the subject of the complaint. The students that answered the counter-questions are quite unified thinking that there are an adequate number of assignments on the course and that they were good overall.

5.7 Summary

In summary, Mímir was found by the experts to have potential. They said that while it did not solve all problems they had encountered with assignments, it was a step in the right direction. The experts also had plenty of suggestions for future development.

The students generally found that the assignments were easy on some level, but there were assignments and topics that stood out as being somewhat more challenging than intended.

This can be seen in Figures 15 and 16 as large dips in assignment submissions compared to earlier years. Topic difficulty questionnaires raised the ASCII-table, variable parameters and the use of structures as well as functions as topics that were found more difficult than others. This can be seen from Figures 20, 21, 22 and 23. There was a small but noticeable jump in difficulty between weeks 2 and 3, that was also noted by the students in open feedback. The majority understood assignment goals though, and the formatting of instructions were seen by the majority at least ”Quite Easy”.

Assignment submission overall despite these was generally along the trend similar to previous years. Time estimations rose in the later weeks, though the curve stayed similar. The number of submissions per student cannot be compared to previous years due to lack of data, but this course implementation’s data shows that some assignments that continued from the previous week got more submission attempts per student than others. This suggests that the students had some issues with these, though that does not appear to show that prominently in other gathered data. The responses to the course feedback varied more on the negative side than on the positive.

6 DISCUSSION & FUTURE WORK

In this section possible explanations for the results presented in the last section are presented and discussed. The section is divided into subsections discussing the assignments and the tool separately. Future targets for research on the assignments and development areas and ideas on the tool are also discussed.

6.1 Assignments

This section analyzes the results presented in the previous section. It tries to find possible explanations to the results seen, as well as present future development areas and ideas. It also answers the RQs related to the assignments.

6.1.1 Assignment Topics

Ala-Mutka (2005b) states in their thesis that according to research, usually students do not have problem with the programming language structures and concepts, but rather planning the program and using the more complex structures. They note that students usually approach the program ”line-by-line” rather than in logical parts of the program. This is one possible explanation to some of the problems that the students have had with the assignments.

The new topics discussed in Section 4.1.2 seemed to be difficult in the assignments, and especially L3-A5 as seen from Figure 16. This difficulty is also directly said by some students, as seen from Figures 20, 21 and 23. The same topics used in that assignment, the ASCII-table and type conversions, could also be found from L2-A3, which does also have a small dip in assignment completion. L2-A3 interestingly also has a dip in average number of submissions per students. This suggests that the students spent more time testing their program on their own computer rather than submitting it to the auto-grader for testing. From the weekly topic difficulty questionnaire in Figure 20 however, it can be seen that especially the ASCII-table and type conversion still had mostly positive answers, meaning they were considered easy. In addition, the answers to questions relating to understanding or the formatting of the instructions do not significantly differ from the other assignments. Only notable difference is the amount of ”N/A” answers. Although in Week 2 it can be said with more certainty that these new topics were not too difficult, as it can be seen from Week 5 in Figure 23 that the amount of students answering the question was low and these topics do not significantly differ from the other topics on that week. This suggests that the students seemed to have either noticed something in the instructions that turned them off from it immediately without them

even trying to understand the instructions, and only those that completed the assignment answered the questionnaire. This probably skews the answers and hence does not give the most accurate picture of the true difficulty of these topics.

If we compare L3-A5 and L2-A3 to a third assignment, L5-A4, that also had a similar topic as noted in Appendix B, we can see that this assignment does have a notable lack of positive answers in the weekly topic difficulty questionnaire. In it, most of the topics in that assignment have more negative than positive answers. The share of ”N/A” answers from total is still high as like L3-A5. Especially the answers to the formatting of the instructions suggest that the problem specifically in this assignment was not necessarily the topic, but how the topic was presented or explained in the instructions.

It should be noted however that these topics were only explained in the instructions of the assignments and were not a part of the programming guide that the students normally use for all language structures presented on lectures. This probably also plays a part in why these assignment in particular had bigger dips in successful assignment completion. Based on the information gathered, these structures should either be removed from the assignments or explained better in the instructions. Type conversions are a topic that could be covered in the programming guide, but there is little value in explaining how the ASCII-table works, though it could be presented as a concrete example of type conversions. Care should be taken in the future as to not introduce too many new concepts in the assignments without also explaining them in other material, such as lectures and programming guides.

The resulting data gathered from the questionnaires on the difficulty of topics is mostly in line with expectations that more complex topics are seen as more difficult in later weeks as we look at Figures 19 through 25. A few topics do stand out as being more difficult than others. On Week 3 in Figure 21, ”Variable parameters” aka pointer parameters were seen as difficult by more people than those who found it at least somewhat easy. Same is true for ”Combining character strings”, though to a lesser degree than the aforementioned pointers. From empirical evidence it can be said that these have always been somewhat difficult to some students, so it is not surprising to see that some students consider these to be difficult. In week 4 however, from Figure 22 it can be seen that structures and their use seems to jump out as a spike in difficulty. Some part of this may be explained by the fact that L4A3 was a continuing assignment from the previous week, so adding the structure to the existing assignment may also have contributed to the perceived difficulty. This is corroborated by the fact that L4A1 also has structures and their use but those have significantly less negative and more positive answers. The rest of the weeks do not have significant jumps in difficulty, and overall the perceived difficulty increases quite steadily, as is expected as the topics covered get more complex and add to the previous topics.

An interesting comparison can be made between Figure 17 showing submissions per student and the weekly topic difficulty questionnaires in Figures 19 through 25. If we compare the results from L3-A2, we can see that the number of submissions as well as the standard deviation of them is very high, but this does not really show when looking at the difficulty of topics in that assignment from Figure 21. One possible explanation for this and also for some other assignments is that reading the lines from a file introduces line breaks that might not be present in the auto-grader, hence the submission fails when the output is not the same because of the line breaks. This cannot be the only explanation though, since data is handled in many different ways in the assignments throughout the course. Another explanation that is specific to the continuing assignments is that the small or big changes made to the output between weeks as new features are added are not immediately implemented by the students and they go line by line looking for the differences as they add the features.

6.1.2 Instruction Formatting vs Goal Understanding

When comparing Figures 14 and 18, it can be seen that the students seem to understand the goals of what the assignments are meant to teach but sometimes fail to understand what they are exactly supposed to do in the assignment. Some open feedback was also in line with this, like with L1-A3. An interesting point that also seems to corroborate the fact that the goals are better understood is the number of ”N/A” answers between the figures. Both questions were asked in the same questionnaires, so they have the same students answering them, yet in Figure 18 there are less ”N/A” answers than in Figure 14.

6.1.3 Future Areas of Development

The methods and points presented by Lister and Leaney (2003) were not taken fully into account due to administrative and time constraints. However, the course is expected to be redesigned in the near future, which provides an excellent opportunity to bring new types of assignments and new topics onto the course. This can include more things such as pseudo-code instructions, multiple-choice questions where students need to comprehend the function of a snippet of code and more complex algorithms. Implementing these changes would give something to both the weaker and the stronger students, aiding them in their learning and comprehension of programming and the language used.

Obviously, not all assignments can be made or is appropriate to make them easy to understand. The principles which Lister and Leaney as well as Cox and Clark (1998) presented in their papers could be utilized more in the future, with the first assignments of the week being easier and shorter or even containing some pseudo-code steps on how they’re supposed to work, so even the weaker students understand the concepts better. This could enable them to also at least try the harder assignments in search of better grades or better understanding.

Of course more skilled students cannot be forgotten. If the first assignments of the week are easier with more explanation on how they're supposed to be done, the later assignments could have more abstract instructions with less pseudo-code to challenge the stronger students to learn more. However, as the course already has 31 assignments, the work load needs to be balanced so the students do not face a situation where they do not have time to do all the assignments even if they want to due to other course work or other factors impacting their use of time.

To encourage the students to learn more or do the assignments, like answers to the open course feedback said, the assignments could utilize more topical data or topics that are more connected to the real world. The problem with this usually is though that to properly or easily handle data, structures that are introduced later in the course need to be used, which makes them impossible to use early in the course. However, it could still improve motivation later in the course where the number of assignment submissions are low.

6.1.4 Research Questions

Based on the data gathered with the questions in Table 6 and the results gathered from them and displayed in Figure 13 it can be said as an answer to the RQ2 in Section 1.2 that the programmatically generated instruction paper is slightly more preferred among students compared to the handmade paper. Most are still indifferent as to which one they would prefer to see. The generated paper does reduce the time to compile the paper from the assignments into a publishable document, so it is preferred heavily among course staff. The indifference in answers to Question 1 also suggests the changes to the papers are minimal enough that it does not make a difference to the opinions.

To answer RQ3, it is a clear fact based on Figure 16 alone that changing the assignments does affect the amount of students who submit assignments and complete them successfully. However, the impact of changing the layout of the instruction paper is not clear from the data. This is probably difficult to determine as well, as there are too many variables in a normal course, such as the students themselves, corrections and tweaks made to the assignments and course arrangements. The impact of these variables are hard to mitigate. Though, as there is no open feedback regarding the instruction papers and the answers found in Figure 13 sees students being indifferent about the changes, so based on the available data, changing the style of the assignment instruction paper has minimal effect, if at all, on submission rate or quantity.

6.2 Mímir

This section discusses the current state of the tool. It also gives future areas of development based on the current state. Finally, it answers the RQ related to Mímir.

6.2.1 Current State

The current state of Mímir as of the publication of this thesis can be viewed as the MVP. It has the necessary functions that enable it to be used to manage weekly assignments and select and compile the instruction papers from those assignments. It can maintain a catalogue of assignments with the ability to add, remove and edit assignments. From those assignments, weekly instruction papers can be compiled from either creating all the course weeks at once, or one by one manually. Assignments can be tagged and searched to find a specific assignment from the catalogue to view or edit. Assignments have the ability to store their instructions, example runs, input and result data and other metadata.

6.2.2 Future Areas of Development

While the tool is in a usable state, there are multiple areas that can be improved and expanded in the future. The experts raised some of these in the interview conducted and some were raised during the planning and development phases. For example, in the current version, the assignments are stored locally. While the files are transferable on their own as the file paths are stored with relative paths rather than absolute paths in the JSON-metadata, this would require manual work or setting up a separate *Git* (2023) repository to handle transfer between instructors or devices. A future version could, with this in mind, directly integrate a data sync feature to a Git-based service like GitLab or GitHub. This would provide one possible method of sharing the assignments to another instructor, or back them up to a safer location than just a local copy.

Another integration that could be added to the tool later that the experts in the interview also brought up is an integration to auto-graders, such as CodeGrade. This could provide automatic exporting and compilation of test cases for the assignments, reducing the manual labor required to input all the test cases into the auto-grader. Saving time from inputting the test cases to the auto-grader would enable the instructor to either make more meaningful tests or focus on other areas. However, this feature would require close co-operation or at the very least a structured file import from the side of the auto-grader.

As mentioned in the previous chapters, the experts raised the ideas of importing large sets of previously done assignments through some structured file format, like CSV and exporting them in HTML. Creating the import feature is rather simple. If the column names are fixed, then the file does not even need to follow the same column order. Metadata can then easily

be stored in JSON files. The HTML export feature would require some more work than the import, as a whole new HTML generator needs to be programmed. However, this should not bee too difficult, as a simple text layout without fancy graphics is not a hard task to accomplish.

The current MVP of Mímir only supports smaller weekly assignments. The JSON file that in this version stores the document settings does have a `"Mimir_class"` key, which in the MVP is not used for anything, but could be expanded to feature other types of assignments, like larger course projects. These would require a different document format and some changes to the LaTeX generator, but are an entirely possible area of future development that would bring all course assignment instructions to a consistent layout. The MVP's scope was strictly programming assignments, but it would be possible with a bit of refactoring to include other types of assignments as well.

The suggestion of the experts that the instruction papers would be able to be configured by the instructor to their liking is a rather large change. It would require significant changes to the LaTeX generator as well as the data inputs, if the instructor wanted to add or remove components. With this in mind, this is unlikely to be a large priority in future development.

As it stands, the code highlighting in the instruction papers created with Mímir is done with the main coding language chosen when adding an assignment. This however is slightly problematic, especially with the C language and current assignments. On the example course's lecture week 7, program compiling using Makefile is introduced and used. This creates a problem, where the Makefile is not highlighted properly, as the tool only passes the assignment code language, in this case C, as an argument to the `minted` package that is used to make the highlights. This study chooses not to make any recommendations as to how to mitigate this problem, but this is a feature that should be changed in a future version.

This thesis has shown that a lot of feedback and data can and is gathered to support CS1 and CS2 courses at LUT. Right now, this data is scattered and separate from the assignments. A future target of development could be that the feedback and assignments are put together, so that instructors have a clearer idea of what needs to be improved or emphasized more on the course. One such idea would be to implement an import feature to Mímir to import the feedback data for the program to compile and store to be displayed in conjunction with the assignments the feedback relates to, or simply overall feedback from the course. This feature would most likely need significant improvements to the GUI of the program, but would be definitely doable.

6.2.3 Research Question

Despite its shortcomings and the abundance of future development ideas, it can be said that Mímir is suitable to be used as a programming assignment manager, hence being an answer to the RQ1. While the creation of the assignments is still manual work in regards to the tool, it can be used to add these assignments into a catalogue for an easier management of them. This catalogue can be then used to create sets from them and store and export those for use in courses.

7 CONCLUSION

The available material has shown, that although it cannot be fully verified, there are indicators that some form of of inter-year plagiarism exists with course assignments on LUT programming courses due to the assignments being almost identical. Hence, this study has aimed to combat this problem by creating a number of new assignments as well as an assignment manager. The assignments can be used in conjunction with or by replacing the previous assignments that the C language course has had. These assignments were created to still teach the same basic structures and themes that the previous assignments had, but with new topics. In order to evaluate whether these new assignments were suitable and usable in the future, the students who participated in the course were asked questions in multiple questionnaires throughout the course. The largest data set, the submission quantities of assignments detailed in Figure 15, shows that while there was fluctuation between some assignments compared to previous years, overall the trend stayed the same with students focusing on the easier assignments at the beginning to pass the course.

The topics that were completely new were seen as somewhat difficult. From Figures 16 and 20 it can be seen that there were noticeable spikes on the assignments that had completely new topics. The difficulty was also mentioned in the open feedback of the weeks that those assignments were present in. Looking at the overall scores, most people still found them to be doable, if not easy. This study concludes that new topics that are introduced in the assignments but not in lectures should be explained in more detail and that there should not be too many of them in order to not burden the students.

The tool created during this study, Mímir, is an answer to the problem of a large amount of assignments that need to be organized. The tool was designed and made with the goal of helping the course instructor to easily organize and manage catalogues of programming assignments, with the ability to also create visually clear instruction papers for use on programming courses. In the testing that was done on the Spring 2023 C course, students found that the instruction papers created with Mímir are easy to read and help them understand what the assignment is about. The new separation of inputs, examples from data and results is a good feature according to the students. The change from the old style to the new style generated with the tool did not noticeably affect the submission rates, but it cannot be said for certain, as there were other more prominent factors that affected the submission rates. In addition to the students, the experts that were interviewed saw high potential and many possibilities in the tool that were not originally thought of. The experts also gave noteworthy suggestions that if implemented would enable them to start using the tool.

REFERENCES

Ala-Mutka, Kirsti (June 1, 2005a). “A Survey of Automated Assessment Approaches for Programming Assignments”. In: *Computer Science Education* 15.2. Publisher: Routledge, pp. 83–102. ISSN: 0899-3408. DOI: 10.1080/08993400500150747.

– (Nov. 2005b). “Automatic assessment tools in learning and teaching programming”. Publication / Tampere University of Technology, vol. 559. PhD thesis. ISBN: 9789521514708.

Allen, Joe Michael, Frank Vahid, Alex Edgcomb, Kelly Downey, and Kris Miller (Feb. 22, 2019). “An Analysis of Using Many Small Programs in CS1”. In: *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*. SIGCSE ’19. Minneapolis, MN, USA: Association for Computing Machinery, pp. 585–591. ISBN: 978-1-4503-5890-3. DOI: 10.1145/3287324.3287466.

Angell, Lance (2006). “The relationship of impulsiveness, personal efficacy, and academic motivation to college cheating.” In: *College Student Journal* 40.1.

Balaa, Ziad (Apr. 2016). “Developing an exercise management system for e-learning”. In: *2016 Sixth International Conference on Digital Information Processing and Communications (ICDIPC)*. 2016 Sixth International Conference on Digital Information Processing and Communications (ICDIPC), pp. 93–96. DOI: 10.1109/ICDIPC.2016.7470798.

Belter, Ronald and Athena du Pré (Oct. 13, 2009). “A Strategy to Reduce Plagiarism in an Undergraduate Course”. In: *Teaching of Psychology* 36.4. Publisher: Routledge, pp. 257–261. ISSN: 0098-6283. DOI: 10.1080/00986280903173165.

Ben-Ari, Mordechai (2001). “Constructivism in Computer Science Education”. In: *Journal of Computers in Mathematics and Science Teaching* 20.1, pp. 45–73. ISSN: 0731-9258.

Bloom, Benjamin, Max Engelhart, EJ Furst, Walker Hill, and David Krathwohl (1956). “Handbook I: cognitive domain”. In: *New York: David McKay*.

Bromage, Bruce and Richard Mayer (1986). “Quantitative and qualitative effects of repetition on learning from technical text”. In: *Journal of Educational Psychology* 78. Publisher: American Psychological Association, pp. 271–278. ISSN: 1939-2176. DOI: 10.1037/0022-0663.78.4.271.

Caiza, Julio and José María del Álamo Ramiro (Mar. 2013). “Programming assignments automatic grading: review of tools and implementations”. In: *7th International Technology, Education and Development Conference (INTED2013)*. Num Pages: 10. Valencia, Spain: E.T.S.I. Telecomunicación (UPM), pp. 5691–5700.

Chuda, Daniela, Pavol Navrat, Bianka Kovacova, and Pavel Humay (Feb. 2012). “The Issue of (Software) Plagiarism: A Student View”. In: *IEEE Transactions on Education* 55.1.

Conference Name: IEEE Transactions on Education, pp. 22–28. ISSN: 1557-9638. DOI: 10.1109/TE.2011.2112768.

CodeGrade (2021). *CodeGrade - Virtual assistant for your coding classroom*. URL: https://www.codegrade.com/#how-it-works (visited on 05/05/2023).

Cox, Kevin and David Clark (Apr. 1, 1998). "The use of formative quizzes for deep learning". In: *Computers & Education* 30.3, pp. 157–167. ISSN: 0360-1315. DOI: 10.1016/S0360-1315(97)00054-7.

Cunningham, John (July 25, 2022). *Mimir | Norse mythology | Britannica*. In: *Encyclopedia Britannica*. Norse Mythology. URL: https://www.britannica.com/topic/Mimir (visited on 10/24/2022).

Denny, Paul, Diana Cukierman, and Jonathan Bhaskar (Mar. 19, 2015). "Measuring the effect of inventing practice exercises on learning in an introductory programming course". In: *Proceedings of the 15th Koli Calling Conference on Computing Education Research*. Koli Calling '15. New York, NY, USA: Association for Computing Machinery, pp. 13–22. ISBN: 978-1-4503-4020-5. DOI: 10.1145/2828959.2828967.

Dick, Martin, Judy Sheard, Cathy Bareiss, Janet Carter, Donald Joyce, Trevor Harding, and Cary Laxer (June 24, 2002). "Addressing student cheating: definitions and solutions". In: *ACM SIGCSE Bulletin* 35.2, pp. 172–184. ISSN: 0097-8418. DOI: 10.1145/782941.783000.

Douce, Christopher, David Livingstone, and James Orwell (Sept. 1, 2005). "Automatic test-based assessment of programming: A review". In: *Journal on Educational Resources in Computing* 5.3, 4–es. ISSN: 1531-4278. DOI: 10.1145/1163405.1163409.

Fennema, Jelte (Aug. 2, 2020). *PyLaTeX*. original-date: 2014-01-15T14:52:10Z. URL: https://github.com/JelteF/PyLaTeX (visited on 10/11/2022).

Figma (2023). *Figma: the collaborative interface design tool.* Figma. URL: https://www.figma.com/ (visited on 03/29/2023).

Git (Apr. 12, 2023). *Git - fast, scalable, distributed revision control system*. original-date: 2008-07-23T14:21:26Z. URL: https://github.com/git/git (visited on 04/12/2023).

Hevner, Alan and Samir Chatterjee (2010). *Design Research in Information Systems: Theory and Practice*. Vol. 22. Integrated Series in Information Systems. Boston, MA: Springer US. ISBN: 978-1-4419-5652-1, 978-1-4419-5653-8. DOI: 10.1007/978-1-4419-5653-8.

Hevner, Alan, Salvatore March, Jinsoo Park, and Sudha Ram (Mar. 2004). "Design Science in Information Systems Research". In: *MIS Quarterly* 28.1. Publisher: MIS Quarterly, pp. 75–105. ISSN: 02767783. DOI: 10.2307/25148625.

Huang, Shuyan, Qiongqiong Liu, Jiahao Chen, Xiangen Hu, Zitao Liu, and Weiqi Luo (2022). "A Design of a Simple Yet Effective Exercise Recommendation System in K-12 Online Learning". In: *Artificial Intelligence in Education. Posters and Late Breaking Results, Workshops and Tutorials, Industry and Innovation Tracks, Practitioners' and Doctoral*

Consortium. Ed. by Maria Mercedes Rodrigo, Noburu Matsuda, Alexandra I. Cristea, and Vania Dimitrova. Lecture Notes in Computer Science. Cham: Springer International Publishing, pp. 208–212. ISBN: 978-3-031-11647-6. DOI: 10.1007/978-3-031-11647-6_36.

Hynninen, Timo, Antti Knutas, and Jussi Kasurinen (Nov. 16, 2017). "Plagiarism networks: finding instances of copied answers in an online introductory programming environment". In: *Proceedings of the 17th Koli Calling International Conference on Computing Education Research*. Koli Calling '17. New York, NY, USA: Association for Computing Machinery, pp. 187–188. ISBN: 978-1-4503-5301-4. DOI: 10.1145/3141880.3141906.

Ihantola, Petri, Tuukka Ahoniemi, Ville Karavirta, and Otto Seppälä (Oct. 28, 2010). "Review of recent systems for automatic assessment of programming assignments". In: *Proceedings of the 10th Koli Calling International Conference on Computing Education Research*. Koli Calling '10. New York, NY, USA: Association for Computing Machinery, pp. 86–93. ISBN: 978-1-4503-0520-4. DOI: 10.1145/1930464.1930480.

Imrie, Bradford (July 1984). *In Search of Academic Excellence: Samples of Experience*. ERIC Number: ED294467.

Jankowitz, Hugo (Jan. 1, 1988). "Detecting Plagiarism in Student Pascale Programs". In: *The Computer Journal* 31.1, pp. 1–8. ISSN: 0010-4620. DOI: 10.1093/comjnl/31.1.1.

Joughin, Gordon (2010). "The hidden curriculum revisited: a critical review of research into the influence of summative assessment on learning". In: *Assessment & Evaluation in Higher Education* 35.3, pp. 335–345. DOI: 10.1080/02602930903221493.

Joy, Mike and Michael Luck (May 1999). "Plagiarism in programming assignments". In: *IEEE Transactions on Education* 42.2. Conference Name: IEEE Transactions on Education, pp. 129–133. ISSN: 1557-9638. DOI: 10.1109/13.762946.

Lister, Raymond and John Leaney (Jan. 1, 2003). "First year programming: let all the flowers bloom". In: *Proceedings of the fifth Australasian conference on Computing education - Volume 20*. ACE '03. AUS: Australian Computer Society, Inc., pp. 221–230. ISBN: 978-0-909925-98-7.

Mann, Samuel and Zelda Frew (Jan. 1, 2006). "Similarity and originality in code: Plagiarism and normal variation in student assignments". In: 52, pp. 143–150. ISSN: 1-920682-34-1.

March, Salvatore and Gerald Smith (Dec. 1, 1995). "Design and natural science research on information technology". In: *Decision Support Systems* 15.4, pp. 251–266. ISSN: 0167-9236. DOI: 10.1016/0167-9236(94)00041-2.

Markus, M. Lynne, Ann Majchrzak, and Les Gasser (Sept. 2002). "A Design Theory for Systems That Support Emergent Knowledge Processes". In: *MIS Quarterly* 26.3. Publisher: MIS Quarterly, pp. 179–212. ISSN: 02767783.

mbello (Feb. 5, 2019). *pdflatex: Simple wrapper to calling pdflatex*. Version 0.1.3. URL: https://github.com/mbello/pdflatex (visited on 11/10/2022).

McCabe, Donald L (Nov. 29, 2005). “Cheating among college and university students: A North American perspective”. In: *International Journal for Educational Integrity* 1.1. ISSN: 1833-2595. DOI: 10.21913/IJEI.v1i1.14.

McCracken, Michael, Vicki Almstrum, Danny Diaz, Mark Guzdial, Dianne Hagan, Yifat Ben-David Kolikant, Cary Laxer, Lynda Thomas, Ian Utting, and Tadeusz Wilusz (Dec. 2001). “A Multi-National, Multi-Institutional Study of Assessment of Programming Skills of First-Year CS Students”. In: *SIGCSE Bull.* 33.4, pp. 125–180. ISSN: 0097-8418. DOI: 10.1145/572139.572181.

Nikula, Uolevi, Orlena Gotel, and Jussi Kasurinen (Nov. 1, 2011). “A Motivation Guided Holistic Rehabilitation of the First Programming Course”. In: *ACM Transactions on Computing Education* 11.4, 24:1–24:38. DOI: 10.1145/2048931.2048935.

Paiva, José Carlos, José Paulo Leal, and Álvaro Figueira (June 9, 2022). “Automated Assessment in Computer Science Education: A State-of-the-Art Review”. In: *ACM Transactions on Computing Education* 22.3, 34:1–34:40. DOI: 10.1145/3513140.

Papancea, Andrei, Jaime Spacco, and David Hovemeyer (Aug. 12, 2013). “An open platform for managing short programming exercises”. In: *Proceedings of the ninth annual international ACM conference on International computing education research*. ICER ’13. New York, NY, USA: Association for Computing Machinery, pp. 47–52. ISBN: 978-1-4503-2243-0. DOI: 10.1145/2493394.2493401.

Peffers, Ken, Tuure Tuunanen, Marcus Rothenberger, and Samir Chatterjee (Dec. 1, 2007). “A Design Science Research Methodology for Information Systems Research”. In: *Journal of Management Information Systems* 24.3, pp. 45–77. ISSN: 0742-1222. DOI: 10.2753/MIS0742-1222240302.

Roig, Miguel and Marissa Caso (2005). “Lying and cheating: Fraudulent excuse making, cheating, and plagiarism”. In: *The Journal of Psychology* 139.6, pp. 485–494.

Saarivuori, Rami (June 2023). *Mimir*. Version 0.9.6. URL: https://github.com/Appelsiini1/Mimir.

Sharma, Bal Krishna (July 9, 2010). “Plagiarism among University Students: Intentional or Accidental?” In: *Journal of NELTA* 12.1, pp. 134–141. ISSN: 2091-0487. DOI: 10.3126/nelta.v12i1.3440.

Snyder, Benson R. (1970). *The Hidden Curriculum*. Cambridge: Mass., Mit Press.

Sonnenberg, Christian and Jan vom Brocke (2012). “Evaluation Patterns for Design Science Research Artefacts”. In: *Practical Aspects of Design Science*. Communications in Computer and Information Science. Berlin, Heidelberg: Springer, pp. 71–83. ISBN: 978-3-642-33681-2. DOI: 10.1007/978-3-642-33681-2_7.

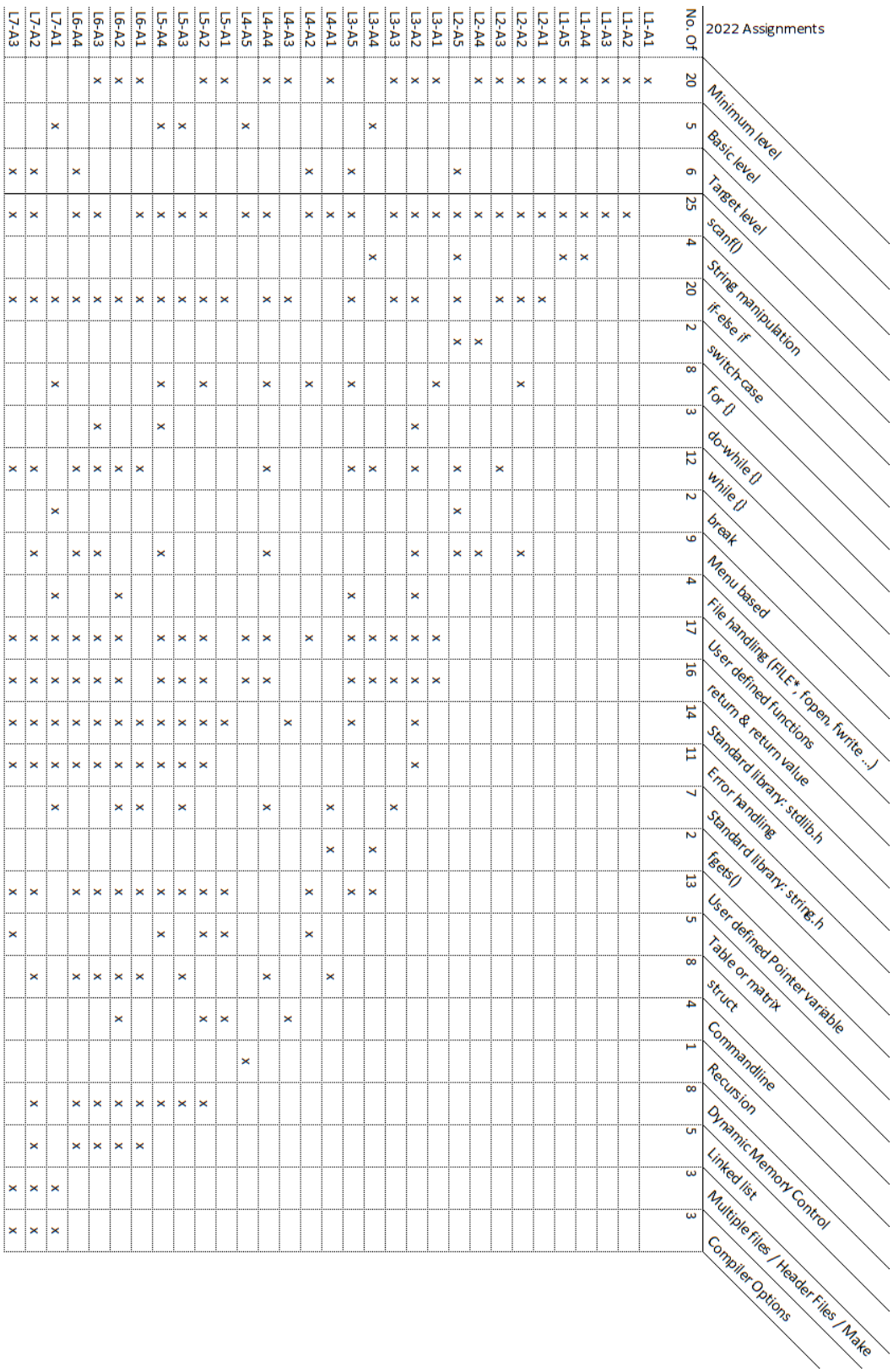
Spacco, Jaime, David Hovemeyer, Andrei Papancea, and Shane Bonner (Mar. 24, 2022). *CloudCoder*. URL: http://cloudcoder.org/ (visited on 05/05/2023).

Technical University of Munich (Apr. 7, 2023). *Artemis: Interactive Learning with Individual Feedback — Artemis documentation*. URL: https://artemis-platform.readthedocs.io/en/latest/ (visited on 09/27/2022).

Tung, Sho-Huan, Tsung-Te Lin, and Yen-Hung Lin (July 1, 2013). "An Exercise Management System for Teaching Programming". In: *Journal of Software* 8.7, pp. 1718–1725. ISSN: 1796-217X. DOI: 10.4304/jsw.8.7.1718-1725.

Whitley, Bernard (1998). "Factors associated with cheating among college students: A review". In: *Research in higher education* 39.3, pp. 235–274.

A APPENDIX 1


2022 Assignments
Minimum level
Basic level
Target level
scanf()
String manipulation
if-else if
switch-case
for {}
do-while {}
while {}
break
Menu based
File handling (FILE*, fopen, fwrite ...)
User defined functions
return & return value
Standard library: stdlib.h
Error handling
Standard library: string.h
fgets()
User defined Pointer variable
Table or matrix
struct
Commandline
Recursion
Dynamic Memory Control
Linked list
Multiple files / Header Files / Make
Compiler Options
No. Of
L1-A1 x x x x x x x x x x x x x x x x x x x x x x x x x
L1-A2 x x x x x x x x x x x x x x x x x x x x x x x x x
L1-A3 x x x x x x x x x x x x x x x x x x x x x x x x x
L1-A4 x x x x x x x x x x x x x x x x x x x x x x x x x
L1-A5 x x x x x x x x x x x x x x x x x x x x x x x x x
L2-A1 x x x x x x x x x x x x x x x x x x x x x x x x x
L2-A2 x x x x x x x x x x x x x x x x x x x x x x x x x
L2-A3 x x x x x x x x x x x x x x x x x x x x x x x x x
L2-A4 x x x x x x x x x x x x x x x x x x x x x x x x x
L2-A5 x x x x x x x x x x x x x x x x x x x x x x x x x
L3-A1 x x x x x x x x x x x x x x x x x x x x x x x x x
L3-A2 x x x x x x x x x x x x x x x x x x x x x x x x x
L3-A3 x x x x x x x x x x x x x x x x x x x x x x x x x
L3-A4 x x x x x x x x x x x x x x x x x x x x x x x x x
L3-A5 x x x x x x x x x x x x x x x x x x x x x x x x x
L4-A1 x x x x x x x x x x x x x x x x x x x x x x x x x
L4-A2 x x x x x x x x x x x x x x x x x x x x x x x x x
L4-A3 x x x x x x x x x x x x x x x x x x x x x x x x x
L4-A4 x x x x x x x x x x x x x x x x x x x x x x x x x
L4-A5 x x x x x x x x x x x x x x x x x x x x x x x x x
L5-A1 x x x x x x x x x x x x x x x x x x x x x x x x x
L5-A2 x x x x x x x x x x x x x x x x x x x x x x x x x
L5-A3 x x x x x x x x x x x x x x x x x x x x x x x x x
L5-A4 x x x x x x x x x x x x x x x x x x x x x x x x x
L6-A1 x x x x x x x x x x x x x x x x x x x x x x x x x
L6-A2 x x x x x x x x x x x x x x x x x x x x x x x x x
L6-A3 x x x x x x x x x x x x x x x x x x x x x x x x x
L6-A4 x x x x x x x x x x x x x x x x x x x x x x x x x
L7-A1 x x x x x x x x x x x x x x x x x x x x x x x x x
L7-A2 x x x x x x x x x x x x x x x x x x x x x x x x x
L7-A3 x x x x x x x x x x x x x x x x x x x x x x x x x
20 5 6 25 4 20 2 8 3 12 2 9 4 17 16 14 11 7 2 13 5 8 4 1 8 5 3 3

B APPENDIX 2

2023 Assignments	Minimum level	Basic level	Target level	scanf()	String manipulation	if-else if	switch-case	for ()	do-while ()	while ()	break	Menu based	File handling (FILE*, fopen, fwrite ...)	User defined functions	return & return value	Standard library: stdlib	Error handling	Standard library: string.h	fgets()	User defined Pointer variable	Table or matrix	struct	Commandline	Recursion	Dynamic Memory Control	Linked list	Multiple files / Header Files / Make	Compiler Options	ASCII-table	Type conversion	Standard library: ctype.h	Standard library: time.h
No. Of	20	6	5	27	10	22	1	5	8	17	1	10	13	19	18	17	18	13	14	16	8	10	4	1	10	6	3	3	4	4	1	1
L1-A1	x																															
L1-A2	x			x																												
L1-A3	x			x																												
L1-A4	x			x	x																											
L1-A5	x			x	x																											
L2-A1	x			x		x																										
L2-A2	x			x		x		x									x															
L2-A3	x			x	x	x				x											x								x	x		
L2-A4	x			x			x				x	x																				
L2-A5	x			x		x			x			x					x															
L3-A1	x			x										x	x																	
L3-A2	x			x		x				x			x	x	x	x	x		x													
L3-A3	x			x		x			x	x		x	x	x	x	x	x	x	x													
L3-A4		x			x	x				x				x	x				x	x												
L3-A5			x	x		x		x		x			x	x	x	x					x								x	x	x	
L4-A1	x			x														x	x			x										
L4-A2	x			x		x				x			x	x		x	x		x	x			x									
L4-A3	x			x		x			x	x		x	x	x	x	x	x	x	x	x	x	x										
L4-A4		x		x				x					x	x	x	x	x	x		x	x											
L4-A5			x	x	x	x			x			x		x	x			x		x			x	x								
L5-A1	x					x								x	x	x	x	x	x	x					x							
L5-A2	x			x	x	x				x			x	x	x	x	x	x	x	x					x							
L5-A3		x		x		x			x	x		x	x	x	x	x	x	x	x	x	x	x	x		x							
L5-A4			x	x	x	x		x	x	x		x	x	x	x	x	x	x	x	x	x	x			x				x	x		
L6-A1	x			x		x				x						x	x			x		x			x	x						
L6-A2	x				x	x				x			x	x	x	x	x	x	x	x		x			x	x						
L6-A3		x		x		x			x	x		x	x	x	x	x	x	x	x	x		x	x		x	x						
L6-A4			x	x	x	x			x	x		x	x	x	x	x	x	x	x	x	x	x			x	x						x
L7-A1		x		x	x	x		x		x			x	x	x	x	x	x	x	x		x			x	x	x	x				
L7-A2			x	x		x				x		x		x	x	x	x			x		x			x	x	x	x	x	x		
L7-A3		x		x		x				x				x	x	x	x			x	x						x	x				