Lappeenranta-Lahti University of Technology LUT
School of Engineering Science
Software Engineering

Petteri Mäkelä

SELECTION PROCESS OF AN AUTOMATED ASSESSMENT SYSTEM AT LUT UNIVERSITY

Examiners: University Lecturer Erno Vanhala
Associate Professor Antti Knutas

ABSTRACT

Lappeenranta-Lahti University of Technology
School of Engineering Science
Software Engineering
Petteri Mäkelä

Selection Process of an Automated Assessment System at LUT University

Master's Thesis 2021
85 pages, 4 figures, 14 tables, 5 appendices

Examiners: University Lecturer Erno Vanhala
Associate Professor Antti Knutas
Keywords: Automated Assessment, Programming, Grading, Mass Course, MOOC

At LUT University automated assessment is used to ease teachers work in the software engineering degree programme's CS1 courses. For these courses, the system in use works well to automate the assessment of over 500 students' weekly submissions. Automated assessment is what makes practical programming tasks possible in a mass course such as this one. In recent years, the student intake amounts at LUT University have increased drastically, especially in the field of software engineering. Due to this, automation in assessment must be expanded further to enable practical programming tasks as a part of teaching. As such, the department of software engineering is looking for a system that would work well for advanced programming courses.

In this thesis, the feature evaluation and selection process of an automated assessment system in university context is carried out. The topics of the thesis can be divided in two categories: the environment of automated assessment systems, as well as the requirements gathering process that is carried out. Both are needed for a successful selection of the tool. In the end, a list of tens of possible tools will be narrowed down to just a few. Based on the needs of the relevant stakeholders, the department of software engineering make their decision on what tool suits them the best.

Tiivistelmä

Lappeenrannan-Lahden teknillinen yliopisto LUT
School of Engineering Science
Tietotekniikan koulutusohjelma
Petteri Mäkelä

Automaattisen tarkastustyökalun valintaprosessi LUT-yliopistossa
Diplomityö 2021

85 sivua, 4 kuvaa, 14 taulukkoa, 5 liitettä

Työn tarkastajat: Yliopisto-opettaja Erno Vanhala
Apulaisprofessori Antti Knutas

Avainsanat: Automaattinen tarkastus, Ohjelmointi, Arvostelu, Massakurssi

Automaattista tarkastusta hyödynnetään opettajien työtaakan pienentämiseen LUT-yliopiston tietotekniikan peruskursseilla. Näillä kursseilla työkalut tekevät tehtävänsä yli 500 opiskelijan viikottaisten palautusten tarkistuksen automatisoinnissa. Automaattinen tarkastus on avainasemassa näillä kursseilla mahdollistaen käytännöllisten ohjelmointitehtävien toteuttamisen. Viime vuosina LUT-yliopiston sisäänottomäärät ovat nousseet merkittävästi, varsinkin tietotekniikan koulutusohjelmassa. Tämän takia automaattista tarkastusta on laajennettava myös muille kursseille, jotta käytännöllinen ohjelmointi on mahdollista pitää osana opetusta. Tietotekniikan laitos etsii nyt työkalua, joka voisi parantaa tarkastusta etenkin ohjelmoinnin jatkokursseilla.

Tässä diplomityössä käydään läpi automaattisen tarkastuksen työkalujen valintaprosessi. Työn aiheet voi jakaa kahteen eri kategoriaan: automaattisten tarkastustyökalujen ympäristöön, sekä vaatimusmäärittelyprosessiin. Molemmat osa-alueet ovat tärkeitä työkalun onnistuneeseen valintaan. Kymmenien työkalujen lista rajataan sisältämään vain kaikista sopivimmat ratkaisut. Tärkeiden sidosryhmien tarpeiden pohjalta tietotekniikan laitos tekee valintansa työkalusta, joka sopii heidän tarkoituksiinsa parhaiten.

ACKNOWLEDGEMENTS

Big thanks to all my peers. There's something deeply meditative in sharing your frustrations and victories with people around you. During Covid-19, this honour would go to anyone online on Discord tuning in to my regularly occurring ranting sessions.

As for the thesis itself, parts of this thesis were done in collaboration with our 3-person team.

Aleksi Järventausta did a lot of work with me, including the literature review, examination of the tools involved, as well as creating the demo courses. He will be releasing his own thesis on the implementation of selected system for the department's web applications course.

Erno Vanhala oversaw the work. For the thesis, this included answering my questions on a weekly basis, and sharing advice on how to move forward with different parts of the thesis. In addition, Erno was a key part in communication within the university, as well as to other related stakeholders.

Petteri Mäkelä 01/07/21

TABLE OF CONTENTS

LIST OF SYMBOLS AND ABBREVIATIONS

Automated Assessment	AA
Automated Assessment System	AAS
AbstraktiSyntaksiPuuAnalysaattori	ASPA
Command-Line Interface	CLI
Computer Science 1	CS1
Comma-Separated Values	CSV
Graphical User Interface	GUI
Hypertext Markup Language	HTML
Integrated Development Environment	IDE
LAB University of Applied Sciences	LAB
Learning Management System	LMS
Learning Tools Interoperability	LTI
Lappeenranta-Lahti University of Technology	LUT University
Department of Software Engineering	SWE
The Interactive Material	TIM
Requirements Engineering	RE
Single Sign-On	SSO
Extensible Markup Language	XML
Virtual Programming Lab	VPL

1 INTRODUCTION

In the field of software engineering, automated assessment (AA) has been a common idea all the way from 1960s (Hollingsworth, 1960). Early two thousands was when automated assessment hit big with publications such as Ala-Mutka (2005). There are two things that have been feeding the ever-increasing need for automated assessment: increasing number of students in software engineering classes, and the greater possibility of functioning automated assessment. The tools of automated assessment are improving, but perhaps the most important factor is that the performance of devices surrounding them has increased tremendously. Internet speeds and device hardware are not only performing better now than ever, but they are also readily available for a large audience at low costs.

The intake number of students in the Bachelors' degree programs at Lappeenranta has risen from 27 in 2018, to 129 in 2021 (Vipunen - Education Statistics Finland, 2021). The number for students approved to all study programmes in software engineering has increased from 72 to 183. The increased student intake affects teachers' job in many ways, perhaps the most notable result being increased workload. For reference, in 2013, 25 students were graded on the object-oriented programming course at the university. In 2021, 25 students alone were given full marks for the same course. The department of software engineering (SWE) at LUT University is now looking to extend their use of automated assessment tools in their teaching. While there is a system in place for automated assessment, it does not satisfy the needs of all relevant stakeholders. Namely, another system would be needed to make the assessment of complex programs possible, such as graphical mobile and web applications. For this thesis work, the main goal is to propose systems to find and test systems fit for these courses. This will be done through weighing features of automated assessment on the needs of related stakeholders.

1.1 Goals and delimitations

The main goal of the thesis is to create a proposition for the university, in which the potential systems are ranked by their suitability. To achieve this, several sub-goals are defined. First, the reasons for revisiting the code grading tools of LUT University are reported. After the motivation is clear, necessary requirements are gathered from the main stakeholders. The last thing before a proposition can be made, is to collect a comprehensive list of tools that could be used for the task. These items are displayed here as a list for a quick reference point:

1. Report the reasons to revisit the code grading tools of LUT University.
2. Identify the different stakeholders of the system and their needs of the solution.
3. Gather necessary requirements of the system to be implemented.
4. Collect a comprehensive list of tools that could be used for the task.
5. Create a proposition for the university, that ranks the systems by their suitability.

1.1.1 Scope, restrictions, and risks

The motivation for the thesis comes from the software engineering department at LUT University. Research is focused on finding an automated assessment system (AAS) to incorporate in SWE's courses. Research is not made on whether increased effort in AA is the only way to achieve better results in teaching. Further research could be made on alternative teaching and assessment methods, such as peer-reviewing, or possibly lowering the importance of practical programming tasks.

As is apparent by research goals, the actual deployment of the system is not included in this thesis. This is covered by another thesis (Järventausta, N.D.) that is being made on the subject. The integration, and usage for course work is also excluded from this thesis. The system is implemented for a real-world problem, and as such, real-world restrictions apply. Work started in January 2021, and selection of the tools was to be made by the start of May 2021. Some of the reporting and research would have to be pushed to reflections made after the work process, as there was still time for that after the selection.

The LUT University is a major stakeholder in work. This introduces risks for research, as maintaining objectivity may be difficult with many interested parties that are involved. Research is concluded also on a solution-first basis. For the thesis this means, that while research and reporting the work is important, the implementation of the system is the most valued aspect of the process.

1.1.2 Research questions

From the goals of thesis work, the following research questions have been derived:

1. What are the requirements of an automated assessment system implemented for the stakeholders of the university?
2. What tools are available for automated assessment?
3. How do the tools available conform to the requirements presented by the stakeholders of the university?

The aim for the research questions is to line both the purpose and the goals of research. The foremost task for research must be the requirements gathering of the system to be implemented. After clear requirements have been stated, tools to fit these requirements can be gathered to include those that fit the stakeholders' needs the best. Finally, with tools to conform to the requirements present, the tools can be ranked based on their suitability. This will create an overview on what requirements can be satisfied by tools and which cannot.

1.2 Research methods

Case study (Runeson and Höst, 2008) was first considered as the go-to research method for the problem. However, as according to Runeson and Höst, "a case study is purely observational while action research is focused on and involved in the change process", action research seemed like a better fit. Action research is a two-stage process, including a diagnostic stage and a therapeutic stage (Baskerville, 1999). In the first phase, analysis of the current social situation is made. In the latter phase, a change is introduced, and their effects are studied. This two-stage process fits well for the problem at hand. In the first stage,

automated assessment systems, and their uses in the university are considered. In the second stage, the proposed systems are demonstrated for the end-users, and their responses are analysed. While action research is a good option for a research of this kind, the timeframe used for research may not allow for proper conduction of action research. The best case would be to select a system for the university, and only proceed to the second stage of action research upon implementation. This would however delay the results of the therapeutic stage by months, and as such, research cannot be conducted this way. Still, action research methodology finds its place at the base of research.

For this thesis, these research methods are followed for the parts where they are applicable. Action research is closely linked to the entirety of the work. In the action stage, a requirement engineering process is carried out. This is namely chapters 3 and 4. After the action stage, a form of therapeutic reflection is done following case study principles, chapters 5 and 6.

1.3 Structure of the thesis

Introduction

In this introduction, the areas of interest for this thesis have been covered. Goals and research questions were determined that will form a basis for the rest of the thesis. Applicable research methods were briefly introduced, and their place in the thesis was determined. Finally, limitations and risks identified were considered, along with problem scope.

Related Research

The second chapter of this thesis is related research. Majority of related research is focused on the area of automatic assessment. First, related research on the development processes of such systems is reviewed. After this, the area of AA is considered in a larger picture to learn more problem at hand. Finally, as thesis work is closely related to requirements engineering, a short look on that area is also taken. During all this, examples are drawn from the research of similar processes to solidify the underlaying concepts.

System Requirements

As the name suggests, requirements chapter will consider the requirements of the tool. In addition to constraints, and the functional and non-functional attributes that the tool should

have, in this chapter the possible stakeholders are considered for their relevancy. In essence, the chapter introduces the people surrounding the system and their needs of the system. The chapter ends with a quick retrospect on how well the requirements were handled.

Gathered Tools & Tool selection

A comprehensive list of tools is generated through literature and other sources. The list of tools will then be reduced by their conformation to the requirements gathered before. At the end of this chapter, a shortlist of 4 tools is generated, that will be taken to further testing. The final decision of SWE is assisted by demo-courses are made with the tools. The demo-courses aim to give a better understanding of how the tools could be used at our university. The demo courses were made accessible to all interested stakeholders within the university. Once again, comparison is made between the tools. This results in a proposal of the best tool for the problem, based on results found during the requirements engineering process.

Discussion

After the results chapter, final area for the thesis is a discussion part. Here, the research questions provided in chapter 1.1.2 are answered. In this chapter, additional considerations are given by the researcher's personal opinions. Finally, further research on the subject is considered, including work still to be done on the area of automated assessment at LUT University.

At the very end of the thesis, a short summary of everything is provided in a compact form.

2 RELATED RESEARCH

In this chapter, research related to the areas of the thesis is covered. Most of the chapter will be related to different areas of automated assessment. In chapter 2.1, the literature review process is covered. Chapter 2.2 reviews similar system development processes. Chapter 2.3 takes a deep look on the area of automated assessment, while chapter 2.4 is reserved literature related to requirements engineering. During these chapters, the related systems described will be covered further, pin-pointing their successes and potential problems.

2.1 Literature Review

Related research was gathered through a literature review. Through some general information gathering online, "Automated Assessment" seemed like a relevant search term for the thesis' topics. As the keyword *Automated Assessment* gave far too many results from Google Scholar, this was further limited to *Automated Assessment Programming*. Google Scholar found 100 articles with these terms included in title. This number of results seemed appropriate for size of the literature review to be concluded. The 100-article list was then reduced by their perceived relevancy. Titles, years, and abstracts were considered to determine which of the articles found were most relevant. The articles were ranked by their relevancy, and on a scale of 0-6, all scoring less than 3 were considered not useful for the purposes of the thesis. This resulted in 35 relevant articles. Upon further reading, some articles were deemed unfit based on relevancy for use in thesis work. The rest were documented, and they would shape the Automated Assessment part of this chapter.

The basic literature review provided an overview on the automated assessment area of the thesis. It did however not cover the area of requirements engineering adequately. As such, additional resources on RE processes are described here. Literature on the subject was searched through Scopus. While Google Scholar was good for finding a large array of literature, Scopus allowed a better search functionality to pinpoint the most relevant ones. Search was made for any literature with the phrase *Requirements Engineering* on its title. First, all related papers were searched, which returned 4035 results. On second search, results

were limited to papers published during years 2010-2020, which returned 2361 papers. From both results, papers were sorted by highest cited, and top 20 papers were looked at for their abstract and title for relevance. 9 articles, books, and other documents were found useful, and overall, they covered many of the problem aspects.

2.2 Implementations of automated assessment systems

One of the most prevalent themes of literature review was the abundance of research made on system implementation in different university contexts. All the papers discussed below state the development of their own system that would make their university's assessment processes more efficient. Research listed here were filtered from a larger number of papers. These were found to be the most reflective, giving a theory basis for their decisions based on the system. Still, many of the systems listed here fall short on many aspects of AA, which gives us an opportunity to learn from their mistakes. In this chapter, the motivation of developing original systems is emphasized, while in the following chapters this list is used as a reference for backing up the good and bad cases of automated assessment.

Edgar was developed at the university of Zagreb. Motivation for developing it, was that even if there are many publications available, the choice of software turned out to be very limited. In their literature review, it was noted that instead of making a new system, existing ones should instead be developed. The plea for making new systems was however also noted, as existing systems have many problems, notably their availability, lifespan, distribution, suitability, lack of examples, and explicit explanation on how they work. Little work was done to tailor the system to the university's needs, the feature set of the AAS was taken from Abelló et al's (2016) requirements for e-assessment system, Noonan's (2006) grading system phases, and Wang et al's (2004) conception of a web-based assessment and test analysis system. (Mekterović *et al.*, 2020)

Siette has been used since 2000s as an assessment tool to many different areas of teaching (Rios, 1998) (Conejo, Guzmán and Trella, 2016). Web based technologies started to pick up on the late 90s, which was when Siette was developed to take advantage of this for teaching. At that time, the field of Intelligent Tutoring Systems for assessment, which is where Siette

took its inspiration from. Originally, it was used in conduction with a European project called TREE, where Siette would generate questions on identification and classification of trees. In recent years it has been developed to better fit complex programming tasks, even while input & output testing for program code has been a feature of Siette for a longer time. (Conejo, Barros and Bertoa, 2019)

SAUCE, a System for Automated Code Evaluation, was developed for the use in distributed systems programming at Johannes Gutenberg University. While the project has since been deprecated, it provided an outlook for what a system could look like in the context of advanced programming courses. Motivation for this, was that current system could not be used for distributed system evaluation. Yet, literature review on systems available isn't included in this paper either. (Schlarb, Hundt and Schmidt, 2015)

At Christopher Newport University, USA, **Webwolf** was developed to make assessment of web applications possible. Namely, it would allow the inspection of HTML elements, as well as some functional properties, such as links on web pages, to be assessed. Some relevant publications were identified for automated assessment of web applications. In addition, major testing frameworks, systems for programmatic browser control, and website assessment tools were described. While a comparison between tools available is healthy, there is little information on why commercial solutions were not considered for their needs. (Siochi and Hardy, 2015).

In Pieterse's article on automated assessment features, they described system in use at the University of Pretoria, South Africa. Motivation in developing the **FitchFork** assessment system were threefold: Programming language support, bandwidth used for systems outside of the university, and the possibility of having students work on system development. In the case of FitchFork, the reasons behind deciding on creating their own system seem realistic. (Pieterse, 2013)

AutoCOREctor was developed at the technical university of Madrid during COVID pandemic. It was based on another system already developed and in use at the university. The ease of further developing a pre-existing system and the difficulties of adopting other tools for their specific usage served as a rationale for development. (Barra *et al.*, 2020)

Daradoumis et al. (2019) included an extensive literature review on work that has been done to identify automated assessment tools. In the end, they determined that for complex programming tasks, there were no good opinions available. For their distributed programming course, they developed a tool called **DSLab**. The research was focused on the effects that their tool had on the students' learning experience, as they had also found out the gap of research related to addressing the effectiveness of AA tools.

ASPA (Abstrakti SyntaksiPuu Analysaattori) is another grading tool in use at LUT University. It was made as part of thesis work (Luukkainen, 2020), which aimed implement a suitable tool to support grading of course projects and electronic examinations. ASPA creates static analysis of student submissions, and it's used alongside a dynamic assessment tool. What this provides, is a way to enhance the existing tools' capability of creating more comprehensive feedback. While some existing systems were considered, ultimately the reason for creating the tool was the choice of language. As the course is still in Finnish, feedback for students should also be given in Finnish – something that was not part of other system designs.

2.3 Typical Features of Automated Assessment

Automated Assessment in teaching is the act of automating the grading process of students' submissions. As mentioned in the introduction, automation of the process has been a rising idea all the way since computers were released. This is especially true for the field of software engineering and other related fields, as assignments are often programs that can be rather easily tested automatically. Features that automated assessment could provide are considered here for future reference. The features stated here will serve as a basis for requirements gathering as they help to give an estimate on what a functional AAS requires.

2.3.1 Static and Dynamic Assessment

Static and dynamic assessment is a common categorization of AAS's made by many authors (Liang *et al.*, 2009) (Ala-Mutka, 2005) (Staubitz *et al.*, 2015). Static methods refer to assessment that has done one the program code, whereas dynamic assessment is made in

conjunction with program execution. A comprehensive automated assessment system should consider both categories. The only way to know if a solution works, is by dynamic assessment. Static methods are not a requirement for automated assessment. However, features such as keyword usage and plagiarism detection complement dynamic assessment well.

2.3.2 Static Assessment

Static assessment refers to assessment done without executing the code. It is required for making sure that the program to be run does not contain any errors that would be a problem during execution. Static methods also allow for many other useful methods, that are not achievable by functional assessment. Examples of static assessment are coding style, programming errors, software metrics, design, and keyword & plagiarism detection (Ala-Mutka, 2005).

Recently there has been new mechanisms for static assessment of programs (Daradoumis *et al.*, 2019). These include for example edit distance (Barker-Plummer, Dale and Cox, 2021), control flow graph similarity measurements (Vujosevic Janicic, Tošić and Kuncak, 2013), as well as Euclidian and Levenshtein word distance (Štajduhar and Mauša, 2015). Edit distance could be used for assessing the errors in problem formalization, which is a subject of software engineering, as well as many other fields. Similarity of control flow graph is derived from functional testing. The idea is, that in addition to running the functional tests, a model is generated based on the flow of the program. This can then be checked against teacher's model solution. It could also be used for plagiarism detection, which was mentioned in the article's future work section. In Stajduhar and Mausa's paper many of the methods (Euclidian, Levenshtein, and edit distances) were used for bettering the assessment of SQL grading. Essentially, the aim of the papers mentioned here, is to detect errors in a corpus of student submissions.

ASPA uses static assessment for a variety of functionalities. It was found that the standardization of feedback shows promise for assisting students with their programming assignments. During testing of the ASPA prototype, most common types of errors found in students' code were:

1. Return statement at the middle of the function.
2. Missing exception handling from the file operation.
3. Missing exception type.
4. Missing return at the end of the function.
5. Missing header comments. (Luukkainen, 2020)

These functionalities underline the room for static assessment in addition to dynamic methods. Static assessment however seems to be less common. In the research documentation referenced in chapter 2.3.1, the rest of the systems did not include static methods in their assessment methodology.

Security is not enforced in the tools via static methods to find potentially malicious programs before executing. SAUCE relies on Unix permissions, sandboxing, resource limits, and firewall solutions to keep malicious and erroneous use at bay. For security, many of the systems rely on sandboxing first and foremost instead of incorporating static methods for assessment. Sandboxing reduces risks of student submitted code to very low amounts. Yet, without any static methods, a question is raised whether the malicious users of the system could ever be caught.

As for plagiarism, many of the tools referenced did not touch on it or claimed that it was an issue for future. FitchFork used a script to calculate the MD5 hash for each archive uploaded, producing the number of exact copies within student submissions. Nearly 70% of students were found to be involved in re-uploading identical articles made by other students. Later, the student submissions were decided to be put through MOSS (MOSS, 2021), a system for static plagiarism detection (Schleimer, Wilkerson and Aiken, 2003). Similarly, SAUCE compared student submissions by “computing the Jaccard index over n-grams in the source, which are presented to teacher in a list or as a dendrogram gained by agglomerative clustering”. In layman’s terms, the similarities found in student submissions could be modelled for the teacher. These cases are great examples of the place for static methods in automated assessment systems.

2.3.3 Dynamic Features

Code functionality and code efficiency are what's measured through dynamic assessment (Ala-Mutka, 2005). The only way to make sure that the code does what it is asked, is by running it and checking its' outcomes. Code efficiency is an example of other types of assessment that can be measured while concluding dynamic assessment. This can be for example comparison of the code's runtime or memory usage against a model solution. Depending on the toolset used, the reasons leading to failure – i.e., the error stack – can yield many answers to why the execution of the program had failed. To sum it, dynamic methods can be viewed as the bread and butter of automated assessment. This could also be seen in the system articles listed before. Testing of a submission is done through assessing the programs state at different points of the execution and making assertions on what it should have done. For the purposes of this thesis, the two most relevant methods are **I/O** (Input/Output) and **unit testing.**

In **I/O testing**, the focus is on what the starting point and ending point of a program are. With given inputs, the program should produce a given output. To illustrate this, a basic I/O test case is shown in Figure 1. The program illustrated will square any number given to it as an input, noted by a rectangle in the middle. Given 3, the answer will be 9, since $3^2 = 9$. Given the number 10, the program would output 100. This output will typically be printed to standard output of the given system, which can then be shown in a terminal or integrated development environment (IDE) depending on the development environment.

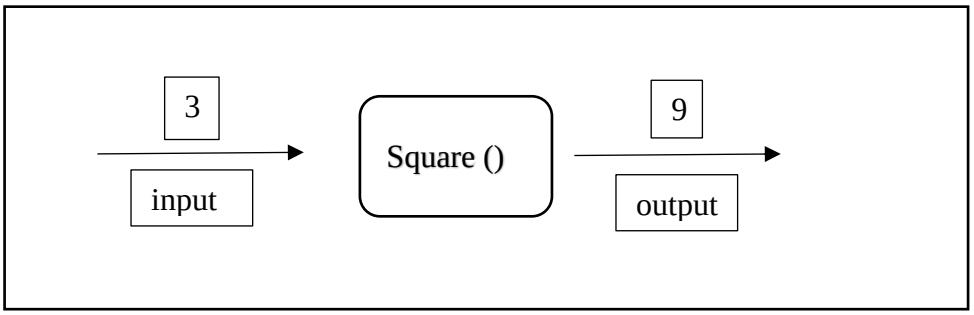

Figure 1: I/O testing

I/O testing can be extended to many different use cases. In CS1 courses, a common task is to formulate an output based on the input. The output can be for example the input squared, a whole string representation of the program flow including a menu, or a csv (comma-

separated values) file. For advanced programming courses, the usage of I/O can be quite limited, as the output is often not a string or a number, but rather a graphical representation of the program's current state. Some courses, such as the web applications course rely on outputs that can be assessed as string such as HTML (Hyper Text Markup Language. However, assessing HTML as a string does not do justice for the program, because it is made to be a graphical representation. This is further extended in graphical user interfaces (GUI) such as those created for mobile in JAVA, where the system's state is hard to represent as a single number, string, or a file.

This is where **unit testing** can be applied. Instead of matching the outputs given by the system, unit testing aims to figure out the system's internal state by testing the units it is comprised of. When code is given by students, typical unit testing may prove difficult. In use cases such as CS1 course in Python, unit testing would be usable only if students' functions would be exported from files in a structured manner. They could then be tested as units. For use cases related to GUIs, unit testing can be done for the components that these interfaces are comprised of. In Figure 2 a component of a simple HTML webpage is shown. It contains a picture of the GUI, an HTML snippet matching that, and assertions that could be comprised based on the HTML. The units in the webpage are for example the whole wiki-item, just the header or the text, or perhaps a collection of the items.

Figure 2: Web component with HTML and assertions.

Testing the functionality of webpages was a key element of many of the documents referenced in chapter 2.1. Webwolf was created to test web applications by using Selenium for programmatic browser control, and JUnit for creating the test cases. AutoCOREctor would use Mocha (Mocha, 2021), Zombie (Zombie, 2021) and Chai (Chai, 2021) libraries to make assertions on student submitted web applications. SAUCE uses many different dynamic methods in its testing of distributed systems – the validation testing however is considered only through I/O of the program. A common way of providing a better usability in testing, was by generating structured outputs instead of only an output string. FitchFork would produce XML (Extensible Markup Language) files of C++ programs to assess, whereas Edgar used SQL and Mongo runners to produce a database record set or a JSON file for assessment. Measurements of code efficiency were also present in many of the tools documented. Execution time of the program was used as a key factor in the SAUCE system for example.

2.3.4 Other Features

There are several features that fall outside of the categorization between dynamic / static assessment. While the two make up for the assessment process of a system, it is not everything that is required by these systems. For the development of Edgar, features covered were categorized further to 9 categories, as depicted in Figure 3. During research of Edgar, it was also discovered that many of these features are neglected in present systems. While all of these aspects may not be needed by an automated assessment system, extra features provided by the systems can be seen as valuable. The benefits along with results gained by using the systems prior mentioned will be discussed later. In this chapter, extra features provided by the systems are described.

Course Administration	System monitoring	Exam logging and monitoring
Problem mitigation	Analysis and visualization	Data import and export
Content authoring	Rich question types and grading facilities	User-friendly online testing

Figure 3:

Features of comprehensive automated programming assessment systems based on model applied in Edgar (Mekterović *et al.*, 2020)

Feedback for the student is a major feature that falls out of static and dynamic methods. The feedback needs to be there but creating good quality feedback is a hard thing to achieve. Providing binary feedback – whether the solution is correct or not – leads to lengthy bug-fixing process among most students (Kyrilov and Noelle, 2016). ASPA in its entirety is focused on the feedback given to students which was seen as beneficial, as was described in Chapter 2.3.2. The basic form of feedback given to students is to relay the relevant compiler error message for students. As an example, WebWolf would provide its feedback with a message of success / failure, and the relevant compiler message. With some work, the compiler messages can be given to the user in a more informative fashion, such as those given by Edgar, where information on the test cases would be given in addition to other information. Finally, novel features can be seen in feedback as well. DSLab which was used for in assessment of distributed systems, could give very clear visuals on how the distribution of the system was handled.

There are many more additional features that automated assessment systems could provide. Analytics are an important feature for many of the systems covered – whether it's students' time usage within the systems, or measurements of the programs created by students. A comprehensive list of these would be too large of a task for the scope of this literature review. For now, it is important to state that while assessment is carried through static and dynamic

methods, there are many more important features of automated assessment systems, that may prove beneficial for their users.

2.3.5 Benefits of automated assessment

The much sought-after benefits of automated assessment are clear. Like in other fields where computer-aided work is required, the key benefits are what these machines do the best. These benefits include objectivity, consistency, speed, and 24h availability (Ala-Mutka, 2005). In mass-courses, these features are what make teaching possible in the first place. WebWolf reported that the assessment times of student submissions were reduced to less than 35 seconds per submission, from the manual time of 17 minutes on average.

It is also clear that many of the features are sought-after for students as well. These features are the most prominent on introductory programming courses. In these courses, the complexity of programs to assess remains relatively simple. For the CS1 course, automated assessment systems provide positive results on students learning, even if the quality of the assessment is a key factor in this (Skalka and Drlik, 2020). Not only is automated assessment good for introductory courses, but its benefits also hold in the advanced courses as well. Systems like AutoCOREctor are helpful for students, as was perceived by students themselves in a web applications course in 2020 (Barra *et al.*, 2020). On a scale of 1-5, the students would like to have tools like autoCOREctor in other course (score 4.7), and automated assessment systems like autoCOREctor were preferred over manual methods (score 4.6) (Barra *et al.*, 2020). The web applications and introductory programming courses are examples where, at least at LUT University, they are made to teach practical programming skills. While programming skills can be taught in other ways, if the goal of the course is to teach practical programming, then the best way to achieve this is by practical programming tasks (Ala-Mutka, 2005).

2.3.6 Potential pitfalls

A statement was made in the last chapter, saying that automated assessment provided benefits, given that it is of high quality. Here, the many potential quality-reducing problems are considered. These are namely: Challenging effort, Increased Effort, Suppressing

creativity, Plagiarism, Higher skills expectations, Malicious programs and Limited capability (Pieterse, 2013). In Table 1, these potential problems are described further. On the left column, these problems are stated, with the right column giving a short description on the matter. Many of these problems are further explained in the following sections.

Table 1: Potential pitfalls of automated assessment (Pieterse, 2013)

Term	***Description***
Challenging Effort	Tasks created for automated assessment require more from the teacher than many other assignment types, and as such correlate with the challenge in effort needed.
Increased effort	The workload in grading tasks is shifted to design and implementation of the assessments. Depending on the number of students, this may increase or decrease the overall effort.
Plagiarism	There are several reasons for the abundance of plagiarism in AA. If there is anything to be gained by plagiarising work from others, the AAS needs to be able to deal with such issues.
Malicious programs	Intentional or not, students' submissions may prove harmful to an AAS. From infinite loops to data mining, there are many security issues related to automated assessment.
Higher skills expectations	In addition to AAS's proving harder for teachers to use, it also requires a lot from the students. The system may prove challenging for its users, creating additional concerns.
Suppressing creativity	AAS requires tasks with precise answers. Depending on the type of assessment, this can greatly affect students' freedom with their implementation.
Limited capability	There are limitations to all AAS's. System should be used for what its capable of, while other types of assessment should be prioritized for where they are needed.

All the papers discussed showed the differences in effort regarding automated assessment. While Webwolf took the assessment procedure from minutes to mere seconds per submission, it also introduced JUnit styled testing framework for teachers to handle. Built on top of JUnit and Selenium, while Webwolf did provide teachers with a set of functions to make assessment easier, it still showcases how effort moves from the manual grading process to the assessment design. Increased & challenging effort can also be seen in the implementation of these systems, often requiring proper maintenance of the system. An example of this is the modular architecture used by Edgar, which comprised of 3 different runners, two frontend modules, a core system, and 7 additional modules. As a contrast, manual assessment requires only a few modules - a submission library, assessment environment, and assessment logging system.

Many of the tools stated, that plagiarism was unavoidable. As was noted earlier, in the testing of FitchFork, many students were found to copy answers from other students. Other than plagiarism, the automated assessment tools are vulnerable to other types of cheating. A basic way to cheat a system, is by giving an answer that fits the input/output tests in a way that is not the expected solution (Kratzke, 2019). If the task is to create a program that squares a number, you would expect the students to create such program. However, if the student knows what the test cases for the task are, they can simply print the answer without doing any maths in the program itself. In Figure 2, this type of cheating is illustrated in Python code. In the top, the correct solution for the program is presented, where the program squares any number given to it. On the bottom, the program outputs string values based on the input given. The main takeaway from this, is that this basic form of cheating can be very widespread, and it can skip a large amount of logic required by the program.

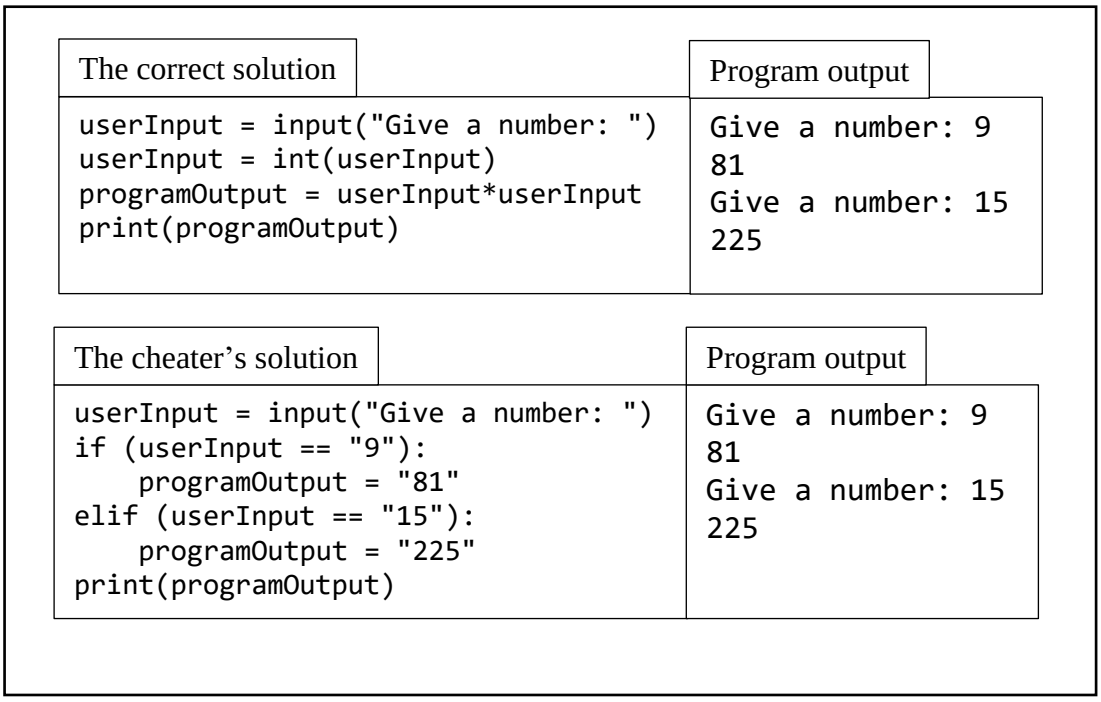

Figure 4: I/O Cheating example

Kratzke (2019) further explains that, if the tool in question is not secure enough, there is a possibility for other types of cheating, such as injecting points by using input / output streams. Here, instead of outputting the "correct" answer, the program could put something in the output stream to indicate the system, that the program should be graded with full

marks. There might be various ways of taking advantage of an automated assessment system's way of processing student submissions, depending on the tool.

From the students' standpoint, higher skills expectations and suppressing creativity are real obstacles. While the papers did not mention these, it can be seen where the problems raise. Higher skills expectations come with the use of the system. Not only do students need to learn how to create a solution for the task at hand, but they will also need learn how to use the system linked. Depending on the system design, this has a potential to be a large issue. Additionally, suppression of creativity stems from the rigorous testing the systems include. Depending on the system design, submissions may show some leniency with how to approach this, or they may require students to submit pixel-perfect solutions. This issue somewhat ties to the limited capability of the systems. The system should only be used for what it is capable of. If the course is about web application design, the students should be given a possibility of actually designing solutions, instead of focusing on matching the specification precisely.

In the previous chapter, it was noted that many attributes of automated assessment are preferred by students over manual assessment. Especially courses where the assignments are straightforward, the system can be very well received, such as autoCOREctor when used on a CS1 course. For tasks outside of CS1 courses, such as on a more advanced object-oriented programming course, the benefits of automated assessment remain unclear (Skalka and Drlik, 2020). Pieterse (2013) summarizes the potential problems automated assessment well: "Applying automated assessment is in itself not a solution. Automated assessment can only add value to a MOOC if it is of high quality. Providing high quality automatic assessment can be very challenging and demands increased effort from the instructor." High quality systems make many different types of testing available for the teacher to use. It still requires a lot from the teacher to figure out where automated assessment can be best used on their courses, and where they may do more harm than good.

2.3.7 Automated Assessment Conclusion

While there are several problems that automated assessment brings, the basic need for a system remains the same. There are simply far too many student assignments for the teachers

to assess, yet practical programming through the assignments is required. Even if automated assessment brings its own problems, the benefits are clear. When handled well, AAS's enable students to learn through practical programming which prepares students best for practical tasks in the future.

2.4 Requirements Engineering Process

While the typical features of automated assessment make for a good baseline, the needs of the stakeholders must be considered. This includes the stakeholder's perspective of what makes an automated assessment system good, and what the staff of the university want for the feature set. In addition, there are certain limitations, such as budget and integrations to other services, that are a must for the university. To account for all these things, a look on the requirements engineering (RE) process is done here.

The book Requirements Engineering (Hull, Jackson and Dick, 2011) is followed for the requirements engineering process. It functions as a practitioner's guide, providing much needed insight on the requirements engineering process. This process can be split to two phases: the problem domain, and the solution domain. In the problem domain, the needs of the system are developed into requirements of the system through stakeholders. In the solution domain, the requirements gained are implemented through a system and its related subsystems. The practical work covered in this thesis mostly stays in the problem domain, as creating an entirely own system is not in the needs of the university.

IEEE (2017) defines a requirement as "a statement statement that translates or expresses a need and its associated constraints and conditions". While the functional requirements have been discussed thoroughly in the past chapters, there remains something to be said about the non-functional requirements as well. No matter how well the assessment system functions, if it is not usable, it will not grant a positive impact for anyone involved. Non-functional attributes are further explained by (Chung and Do Prado Leite, 2009, p.), where it is argued that the emphasis in requirements is lop-sided towards functional attributes. Chung and Do Prado Leite also make a robust definition of non-functional requirement. This definition varies greatly in literature. For the purposes of this thesis, non-functional requirements are

all requirements not related to functionality of the program, the "-ilities" and "-ities". For the assessment system, this might include requirements such as usability, security, and maintainability.

Requirements engineering process comes with its own potential pitfalls. Most common RE problems include incomplete and / or hidden requirements, communication flaws between project team and customer, and moving targets, among others (Fernández *et al.*, 2017). These are not by any means new observations. Yet, the prevalence of these problems found in survey's such as Fernández et al.'s underlines the importance of a solid requirements engineering process. Most cited causes for these problems were reported to be lack of time and lack of experience of RE team members, with many more common occurrence being reported alongside these (Fernández *et al.*, 2017). To answer these risks and obstacles, different frameworks and techniques have been suggested (Asnar, Giorgini and Mylopoulos, 2011) (Van Lamsweerde, 2000). For the purposes of this thesis, the requirements engineering process is not expected to come with great risks and obstacles. Many parts of the process may be handled by the SWE department – hidden away from the scope of this thesis. Should any problems uncover, these articles among others provide different ways of proceeding with the process.

3 REQUIREMENTS FOR THE TOOL

In this chapter, the requirements of the new tool are specified. These are gathered from many sources. At the end of each chapter, the requirements, constraints, and stakeholders related are listed, with the abbreviations of:

R referring to functional Requirement,
QR referring to Quality Requirement
C referring to Constraint, and
S for Stakeholder.

First, the automated assessment environment at LUT is considered to learn from the needs of automated assessment in past use cases of the university. Second, the stakeholders and their needs for the system are considered. An emphasis is made on the stakeholders of LUT University, for which a questionnaire is presented. This chapter is concluded by stating the most important requirements and constraints gathered.

3.1 Automated assessment environment at LUT

Moodle

At LUT, the main LMS (Learning Management System) in use is Moodle (Moodle, 2021). LUT University uses Moodle as the base for all courses, for which different systems may integrate to. Moodle provides some automated ways of assessment such as quizzes. Most of the assessment is semi-automated since in one way or another, Moodle tries to make the assessment process as simple as possible. Programming tasks however are something that Moodle does not touch upon.

Viope

Viope (*Viope*, 2021) has been in use at LUT University since early 2000's. The long-lasting relation with their services has been good for both parties involved. Viope is currently in major use for many introductory programming courses, including CS1, C-programming, and Java courses. However, in its current form, this service cannot provide assessment for

complex programming tasks. Courses dealing with technologies such as web and mobile applications need another tool. **ASPA**, which was described in chapter 2.2 and further in chapter 2.3, has been developed to be potentially used alongside Viope to provide better feedback for students.

Autograder

As the student numbers at LUT University kept rising, an automated tool for web applications course was developed. This tool was Autograder (Knutas *et al.*, 2019), and it was used for two instances of a web applications course. Autograder provided assessment through a testing framework called Cypress (Cypress, 2021). The maintenance procedures, such as creating new test cases for the assessment proved an issue. On the latter instance, the system could not handle the number of students submitting assignments. This is what ultimately lead to the motivation of seeking new assessment tools for the department of software engineering.

The automated assessment environment gives the following constraints for the new tool:

C1 – The system must be compatible with Moodle

C2 – The system must be able to assess complex programs

C3 – The system must be reliable for mass courses of LUT University

3.2 Stakeholders

A stakeholder is an individual, group of people, organisation or other entity that has a direct or indirect interest (or stake) in a system (Hull, Jackson and Dick, 2011). The number of stakeholders for a given system can be very broad, and all of them should be considered during the requirements engineering process. Stakeholders of the system were identified during the early stages of the requirements engineering process. Most of the information of potential stakeholders were gained during talks with SWE and other universities outside of LUT. The list that was formed was considered for its completeness by comparing it with Hull et al.'s extended list of 14 different stakeholder types. In Table 2: Stakeholders of the system, the stakeholders of the system are listed. The way of noting the stakeholder, as well as their group has also been considered. The groups considered are End-user, Maintenance,

Management, and Other. These groups will later be discussed for their needs as well as the means of gathering their relevant requirements of the system.

Table 2: Stakeholders of the system

Stakeholder	***Group***	***Way of Acquisition***
SWE Staff	End-user	Initial talks with SWE.
Students of SWE Courses	End-user	Initial talks with SWE.
Information Services	Maintenance	Initial talks with SWE.
SWE Management	Management	Initial talks with SWE.
Digital Learning Team	Management	Contact by Digital Learning Team on questionnaire.
Other teaching staff of LUT-Group & their students	End-user	Possibility of cooperation with LAB was found out during talks with SWE. Other interested parties of the university were found during the process.
Other Finnish universities	Other	Talks with other universities during the process.
Training personnel of the system	Maintenance	Addition with the consideration of Hull's list.
Usability experts	Other	Addition with the consideration of Hull's list.
Regulatory authorities	Management	Addition with the consideration of Hull's list.
Opinion leaders	Other	Addition with the consideration of Hull's list.
Product Owners	Maintenance	An own consideration during consideration.

End-User

Stakeholder groups: SWE staff, students of SWE courses, other teaching staff of LUT-Group & their students.

The groups classified as End-users are those that will be using the system daily to perform their daily tasks. This includes both the teaching personnel, who will be using the system to assess students, as well as the students submitting their assignment solutions. Teaching environments of other groups inside LUT-Group are also considered, even if they may be affected by the system only in the future.

Management

Stakeholder groups: SWE management, regulatory authorities, digital learning team

These stakeholders manage and govern the use of the system. The most crucial stakeholder of this group is the SWE management personnel. They have several important actions in the

requirements engineering process, such as giving the motivation for the work, as well as making the final decision on the system. Digital Learning Team is a group within LUT, that manages the digital learning environments of the university, such as Moodle. They have a vested interest in all digital systems used to support learning at the university, and as such they have been separated from other regulatory authorities. The other regulatory authorities include for example management groups of LUT University outside of the department. These have a lesser importance, as they are connected mostly through the SWE management.

Maintenance

Stakeholder groups: Information services, training personnel of the system, product owners

Maintenance groups of the system handle with how the system is maintained during and after implementation. Information services of LUT University handle the maintenance of LMS's it connects to, user access to the system, security, and hosting of the system, among many activities. The training personnel will make sure that all teachers that need the system are able to use the system properly. This will be the community surrounding the system, whether it is dedicated support personnel, the system's developers, the advanced users of the system at LUT University, or its users wherever it operates at. To make it easier for users of the system to get a hold of the training personnel, a product owner for the system is appointed. During the work of this thesis, that product owner will be the team involved.

Other

Stakeholder groups: Other Finnish universities, usability experts, opinion leaders

The other stakeholders of the system cover a large variety of people, both internal withing LUT University, and external. Other Finnish universities are tied closely to the automated assessment environment in Finland, and their needs are considered later. Usability experts and opinion leaders are important figures that can stand out in any of the groups. Usability experts can be very useful for their knowledge on the subject, and they are found for example in the groups that have used similar systems before. Opinion leaders are those who have strong opinions themselves but are also capable of affecting other people's opinions of the system. These are also found in various groups including university teachers, students, as well as those stakeholders not directly affected by the system.

3.2.1 Role of stakeholders in requirements engineering process

While there are a great number of stakeholders that will be affected by the work, their role in the process is kept low. The most important stakeholder group for the scope of this thesis are the end-users, namely the teachers that would use the system in the future. During a typical requirement engineering process, the needs of all stakeholders should be considered, documented, and approved by themselves. This process could be closely followed, but for the purposes of this thesis some work in requirements gathering is omitted. There are a few important reasons for this:

Time and effort

The work for this thesis has a time constraint, and there is no large development team involved. Work needs to be placed on the important matters. In requirements gathering this means, that many stakeholder needs are derived from literature or from the needs of other relevant stakeholders.

Absence of contribution to system development

The outcome for the requirements engineering process is to decide on an already existing system that will be introduced. These systems may be inadequate to fill the needs of the stakeholders, even if those needs were well documented. As such, it is more important to report the most important constraints of the system and work with the most important stakeholders.

The continued requirements engineering process after this thesis

The system decided on will not be in full use after this thesis. Rather, it is used as a prototype on selected courses, where its usability is fully considered. As of this thesis, the most important constraints of different stakeholders are detrimental, while other constraints and needs can be still considered down the line. This will however impact the selection of the system, and the realization that the system might have to be used for purposes other than described as the result of this thesis.

3.3 End-user requirements

The most requirements for the system are gained from the system's end users. These are namely the students submitting their assignments, and the teachers working with the system's assessment features. For the scope of this thesis, requirements were gathered systematically through the teachers. The reasons for this were discussed in chapter 3.2.1. In essence, it was deemed not worthwhile to gather requirements from students in addition to teachers at this point of the requirements engineering process. It should also be noted that teachers share many of the perspectives as students do. This does not remove the need to consider how the tool is later viewed by the other significant end-user group: the students. As for the requirements gathering methods, a survey was decided upon. It was suggested by the SWE Department, as surveys have been proven to be a decent way to gather information at the university.

3.3.1 Survey

A questionnaire was created for the university staff to get a grasp of the important features of the system. Kitchenhams's & Pfleeger's (Pfleeger and Kitchenham, 2001) instructive articles on the best practises of surveying were used as a reference for the questionnaire created. The definition of survey is "a comprehensive system for collecting information to describe, compare or explain knowledge, attitudes, and behaviour" (Pfleeger and Kitchenham, 2001). The real value of the articles however relies on the description of survey process – a ten step guide from setting the objectives to reporting the results that could be used to convey the survey. Questionnaire specification is included in Appendix 1, which includes many of these steps of the process.

The questionnaire was held for two major reasons of requirements gathering process. First, the questionnaire aims to give a better understanding on what the main users of the system would like from it. Closed questions rank the requirements that we have picked up based on how important they feel to the teachers. Open questions bring more ideas to the table – they aim to answer whether there's something we have missed during the requirements gathering process so far. In addition to typical requirements gathering, the questionnaire however has a second major reason for requirements gathering: stakeholder engagement. It is important

at every step of work to engage stakeholders with the process to alleviate any misunderstandings or other problems. The survey was one of the first interactions with the university staff to create more discussion on the area of automated assessment.

Results of the closed questions of the survey are in Appendix 2. The main results of the closed questions, as well as all yielded open-question answers are pointed out here. The questionnaire had a total of 8 respondents. In retrospect, the number of respondents correlates well with teachers that currently deal with automated assessment systems, so no more could have been hoped for. The validity of the chosen gathering method will be discussed later. Low respondent numbers mostly affect the quality of the data gathered which must be considered in future work, even if the data is not the only thing that the questionnaire provided. Every one of the respondents knew what automated assessment systems were and only two had not used automated assessment systems on their own courses before.

All respondents would use the system for assessing programming tasks. Half or less would apply it for other uses, such as publishing course material or other tasks. LUT University has Moodle for that, so it is reasonable. Most common environment, where the tool could be used was in terminal programming. Half of the respondents would use it for the assessment of mobile and web applications, while 3/8 would use it for database management and mathematics as well. Most important features were functionality, plagiarism detection, and automated feedback. Run time, common errors, and following the model solution were deemed not as important. Feedback had some variety between answers – perhaps because people did not know what kind of feedback this was. Keyword checking also had variety, most likely because it is very important if the respondent knows how an automated assessment system should work. Non-functional features provided interesting results. Most important was the customizability of the system and ease of use for students. Least desired quality was usage of tools that the user is familiar with. Other possible features for the tool were not deemed as important. Only ‘Very Important’ results were for GitHub returns. Question on the feature-set did spark some ideas, which are in the open feedback section below.

Open feedback

Most teachers gave open feedback on the questionnaire and its subjects either on the questionnaire, or through e-mail. Here are all the answers with differing needs:

On features

- A student-search functionality
- Peer-review / code review for student
- Choosing assessment types for different solutions
- Support for additional frameworks, such as NumPy / React.js
- Support for Stata
- Bash scripts / Assessment for Linux course – terminal usage through Moodle

On questionnaire design

- The questionnaire was difficult to understand at points
- There could have been more options for choices

On selection process and tool design

- Some mathematics courses have an automated assessment system in use
- The system should do at least one language well to not make the experience for all languages worse – e.g., providing only general i/o assessment
- Choice should be made based on what is available, and what the University can afford
- The teacher should not have to be a coder for the system
- Tools should be considered whether they are categorized based on assignment or assessment types (Web / Introductory / etc.)

3.3.2 Requirements of the End-Users

The base requirements of the end-users have now been gathered. Literature review provided many insights that were used as the baseline for the questionnaire. The questionnaire pin-pointed the exact requirements that are the most sought after for the stakeholders of SWE department's teachers. Answers to survey question 2 "The assessment system should consider the statements listed", that had an average score of 3 or higher marked them important or very important to most respondents. These mark the base of the requirements, as stated below:

R1 – The system should include a feature for detecting plagiarism

R2 – The system should allow unlimited submissions by students

R3 – The system should be able to evaluate how student programs handle resources

R4 – The system should give automated feedback for submissions

R5 – The system should have a possibility for manual feedback of submission

R6 – The system should have a possibility for manual assessment of submission

Based on question 6 considering the environments of the system, half, or more than half of the respondents would use the system in environments as stated by the following requirements:

R7 – The system must be usable for the assessment of terminal programs

R8 – The system should be usable for the assessment of web applications

R9 – They system should be usable for the assessment of mobile applications

Finally, as per additional considerations of question 9, the following requirement can be stated:

R10 – The system should allow submissions through a common version control

Quality requirements of the system were discussed during literature review as well. They formed a baseline for the questionnaire. Based on question 5, “Properties of the system”, the following quality requirements could be stated:

QR1 – The system should be easy to use for the students

QR2 – The system should be customizable for different tasks

Additional requirements could be drawn from the open feedback gathered through the questionnaire. Decisions had to be made on whether given feedback was worthwhile for the requirements of the system. The most prevalent feedback given are stated here as functional and non-functional requirements for the system:

R11 – The system should provide support for external tools

QR3 – The system should provide rich assessment properties

3.4 Requirements of other stakeholder groups

In addition to end users, the other stakeholder groups have requirements as well. The requirements stated here have been gathered through talks with the stakeholders in different contexts. They are not gained by validated requirements gathering methods.

For management, one important quality of the system is its cost. While there is no clear budget, cost is obviously a key requirement for the system. An important thing to notice, is that while paid systems have a clear cost to them, systems provided gratis do not come without costs. These costs may include for example the cost of the server uptime, as well as extra labour due to the lack of dedicated customer service. Whatever the licensing or other fees for the tools decided will be, they must be evaluated based on all costs linked to the product. Security of the system is also a big factor for both SWE department, as well as other authorities, such as the digital learning team. GDPR has had a major impact in university teaching, and the regulations imposed by it should be fit as well. The management also drives cooperation between different parts of LUT, LUT-Group, as well as other Finnish universities. Finally, systems that are already used in Finland are preferred. This preference is to make sure that the systems do not face any unexpected difficulties during implementations, such as following the local laws and regulations.

C4 – The system must have a security policy in place

C5 – The system must be in use in Finland

QR4 – The system should be considered for licensing fees

QR5 – The system should be considered for its costs outside of licensing fees

QR6 – Security of the system should be proven

QR7 – The system should promote cooperation between Finnish universities

Maintenance of the system brings its own set of requirements. First and foremost, the hosting and uptime of the system should be considered. Systems that are hosted outside of the university are preferred, since it alleviates the technical needs for information services. If the system fits the needs of other stakeholders, the more work that is outsourced, the better it is for maintenance. In other words, the technical support for the system in form of actual support personnel and documentation is of high importance. During tools gathering, the technical documentation must have enough information to try the software out, as well as show what it is capable of. Security is also an important feature for maintenance stakeholders. For them, the most important features are that it supports SSO (Single Sign-

On) login and is compatible with LTI (Learning Tools Interoperability) . With these checked, the system should be clear for common security issues. System sandboxing, especially if hosted inside LUT, is of high importance.

C6 – The system must have documentation that supports testing it

R12 – The system should have documentation to help with its usage

R13 – The system should have dedicated support personnel for technical difficulties

R14 – The system should provide an SSO login

R15 – The system should be compatible with LTI

QR8 – Hosting of the system should be considered

C7 – Sandboxing of the system should be secure

3.5 Requirements gathering conclusion

The requirements gathering phase yielded many results that can be taken further. **Stakeholders** and **stakeholder groups** were used during the gathering process to ensure that everyone's voice was taken to account. Stakeholders will be a crucial part of the process in later stages as well. **Constraints** will be used in the next chapter as minimal requirements for the tools. **Functional requirements** ensure that the tool has everything needed in the use of LUT University, while **quality requirements** ensure that the system remains usable outside of its functionality. In Table 3 are listed all the requirements and constraints of different stakeholder groups, while stakeholder groups were listed already in chapter 3.2.

Table 3: Initial requirements of the system

ID	Name	Related stakeholder group
C1	The system must be compatible with Moodle	Assessment environment at LUT
C2	The system must be able to assess complex programs	Assessment environment at LUT
C3	The system must be reliable for mass courses of the university	Assessment environment at LUT
R1	The system should include a feature for detecting plagiarism	End-user
R2	The system should allow unlimited submissions by students	End-user
R3	The system should be able to evaluate how student programs handle resources	End-user
R4	The system should give automated feedback for submissions	End-user
R5	The system should have a possibility for manual feedback of submission	End-user
R6	The system should have a possibility for manual assessment of submission	End-user
R7	The system must be usable for the assessment of terminal programs	End-user
R8	The system should be usable for the assessment of web applications	End-user
R9	The system should be usable for the assessment of mobile applications	End-user
R10	The system should allow submissions through a common version control	End-user
R11	The system should provide support for external tools	End-user
QR1	The system should be easy to use for the students	End-user
QR2	The system should be customizable for different tasks	End-user
QR3	The system should provide rich assessment properties	End-user
C4	The system must have a security policy in place	Management
C5	The system must be in use in Finland	Management
QR4	The system should be considered for licensing fees	Management
QR5	The system should be considered for its costs outside of licensing fees	Management
QR6	Security of the system should be proven	Management
QR7	The system should promote cooperation between finnish universities	Management
C6	The system must have documentation that supports testing it	Maintenance
C7	Sandboxing of the system must be secure	Maintenance
R12	The system should have documentation to help with its usage	Maintenance
R13	The system should have dedicated support personnel for technical difficulties	Maintenance
R14	The system should provide an SSO login	Maintenance
R15	The system should be compatible with LTI	Maintenance
QR8	Hosting of the system should be considered	Maintenance

4 TOOLS GATHERING PROCESS

In this chapter, the tools gathering step of the process is presented. This involves the methods used for tools gathering, as well as the listing of the tools themselves. Chapter 4.1 will state all the tools found. Rest of the chapters are dedicated to refining this initial set of tools based on constraints and requirements made in the previous chapter.

4.1 Generating a comprehensive list of tools available

Tools gathering process started with a list of tools suggested by SWE that should be investigated. To this list, tools found during literature review were added. These tools were gathered from research articles found on Google Scholar, as stated in chapter 2.1. Regular Google Search was used to broaden the array of tools gathered. This additional search was especially important to find corporate and private tools with no linked scholarly articles.

In addition to searching online for resources, the existing tools in other learning institutions were considered. The automated assessment environment in Finnish Universities is varied. The problem faced is the same as in LUT University. There are too many students with too many assignments, and the assessment process should be automated especially for mass courses such as the CS1 courses. Discussions were concluded with a number of Finnish universities. Out of these universities, every university had an automated assessment system in use for some parts of their teaching. These systems however varied based on university, and there was not a clear consensus on what tool would be suitable for everyone. Choosing a universal system is further limited by the fact, that many universities had made their own systems to fit their own needs. In Table 4 the systems used in several Finnish universities are covered. In the first column is the name of the system, and in the second the name of the university it is used in. On the last column, the nature of the system is described. Self-made systems are made and maintained by the universities themselves, while corporate systems are solutions maintained by external parties. Some free, community-based systems were also in use.

Table 4: National AAS Environment

System	*Used in*	*Description*
A+ / Plussa	Aalto University (A+), Tampere University (Plussa)	Self-Made system
Lovelace	Oulu University	Self-Made system
TIM	Aalto University, University of Jyväskylä	Self-Made system
TMC	Aalto University, Helsinki University	Self-Made system
ViLLE	UTU	Self-Made system
WETO	Tampere University	Self-Made system
Codegrade	University of Eastern Finland, Åbo Akademi University	Corporate System
Viope	LUT University	Corporate System
Jupyterhub	Aalto University	Corporate System
Coderunner	Aalto University, University of Eastern Finland	Free Moodle-plugin
VPL	LAB University of Applied Sciences	Free Moodle-plugin

The most fitting tools used in Finnish universities were added to the so-far gathered tools list. A comprehensive list of all tools available is included in Appendix 3. This includes both the tools given by LUT University, as those found during further tools gathering. These tools are further discussed in the following chapters. The list includes a total of 30 different tools.

4.2 Suitability of tools gathered

As a list of 30 tools is far too many to make selections on, the gathered list of tools needed to be reduced based on their suitability. Insights gained from requirements gathering phase would be used for making the needed cuts. In the following tables, tools cut by constraints are covered. Table 5 covers the first batch of tools cut from final decision, and Table 6 covers the second batch. Name of the tool is on the first column, while the source of where information of the tool was found is named on the second column. Finally, the reason for the tools rejection is on the last column, noted by an abbreviation to relevant constraint stated in chapter 3.5.

Based on constraint "C6 – The system must have documentation that supports testing it", 2TSW (Polito, Temperini and Sterbini, 2019), autoCOREctor (Barra *et al.*, 2020), Autogradr (Autogradr, 2021), Edgar (Mekterović *et al.*, 2020), FitchFork (Pieterse, 2013), MIT

OverCode (Glassman *et al.*, 2015), SIETTE (Conejo, Barros and Bertoa, 2019), and SAUCE (Schlarb, Hundt and Schmidt, 2015) were removed.

Table 5: System rejections table #1

Name	***Source***	***Reasons for rejection***
2TSW	Literature Review	C6
autoCOREctor	Literature Review	C6
Autogradr	Google Search	C6
Edgar	Literature Review	C6
FitchFork	Literature Review	C6
MIT OverCode	Google Search	C6
SAUCE	Literature Review	C6
SIETTE	Literature Review	C6
Zybooks	Initial tools from SWE	C6

CodeGrades (CodeGrades, 2021), CSES (CSES, 2021), UZH Sylva (SYLVA, 2021), and Hypergrade (Hyper Grade, 2021) had some documentation available, but due to not being suitable to assess complex programs, they were rejected as well. They all had the same problem of not having great automated assessment properties available. CodeGrades would only provide a curriculum to follow, while assessment would be done manually by experts of the field. CSES & Hypergrade provide limited assessment properties through I/O testing. Finally, UZH Sylva would provide many types of assessment, but it did not support assessment for programming. Further on, STACK (STACK, 2021), Assignment-Autograder (Zmiev, 2021), GatorGrader (GatorGrader, 2021), and Codehs (CodeHS, 2021), were rejected as well. STACK would not support programming assignments but was instead focused more on mathematics. Assignment-Autograder would support programming assignments, but it did so as a standalone script ran from the instructor's machine – falling under the constraint of not being reliable for mass courses. Codehs might have had decent assessment available, but it was targeted to high schools and had a set of courses to follow, leading to worries about its fit in the context of LUT University. GatorGrader was deemed as only an extension to GitHub Classroom, and thus would not cut on its own. CodePost and Sphere Engine were added to the list since there were no use cases for them in Finland, and they had a lacking documentation. These two tools are examples of corporate systems that require a demo for the usage of the tool, which is something that could not be done for every

tool. Finally, TMC was dropped from the list due to having a lacking documentation. Information available was scarce and contact with the system developers could not be gained.

Table 6: System rejections table #2

Name	***Source***	***Reasons for rejection***
Assignment-Autograder	Google Search	C3
CodeGrades	Google Search	C2
Codehs	Google Search	C3
CodePost	Literature Review	C5, C6
CSES	Google Search	C2
GatorGrader	Google Search	C3
Sphere Engine	Google Search	C5, C6
STACK	Digital Learning Team	C2
TMC	Initial tools from SWE	C6
UZH Sylva	Google Search	C2
Hypergrade	Google Search	C2

5 SELECTION OF THE NEW TOOL

The initial rejection of tools that were not available or by no means suitable was a simple task. In this chapter, the list is further reduced by their ranking amongst other tools. As the remainder of the initial tools list provided many worthy choices, all the tools are described in this chapter further. Most weight is given however for the tools that based on ranking were the most suitable for stakeholders' needs.

5.1 Matrix listing

To go forward with the tool selection process, a matrix listing for the remainder of tools was made. This list was made based on the most important characteristics that were introduced in chapter 2, and further recognized during survey of chapter 3. The tools were considered also for additional factors that were found during requirements gathering process, such as its usage in other universities, as well as its suitability for LUT University's IT services. Points were given for the following qualities and functionalities:

Suitability

- Terminal Programming (R7)
- Web Development (R8)
- Mobile Development (R9)
- External Tools (R10)

Maintainability

- SSO Login (R14)
- LTI (R15)
- Usage in Finnish Universities (QR7)
- External Hosting (QR8)

Features

- Plagiarism Detection (R1)
- Unlimited Submissions (R2)
- Resource Management (R3)

- Git Submissions (R10)

Assessment

- Automated Assignment Feedback (R4)
- Possibility for Manual Feedback (R5)
- Possibility for Manual Assessment (R6)

Each tool was given points based on these features. These are documented in Tables 7, 8, and 9. Each table has the tools listed on the left-most column. On the other columns' headers, the names of the requirements considered are stated. On each column, an 'X' states that the requirement is satisfied, whereas a '-' states that the requirement is not satisfied. Points are marked as a running total, increasing with each following table. It should be noted that points were given based on resources made available. Tools would be considered for the documentation, videos, and demos that they had, but no further examination was made at this point. Some tools had a lower score due to lack of accessibility of these resources, while others had a higher score by having the relevant documentation available. Many of the tools could be modified to satisfy certain requirements that they on their own do not yet satisfy. Tools like these have an advantage in that they could be made better, however this was not a premise that the university could count on as points were given.

The system's considered here are A+ (A+, 2021) (Karavirta, Ihantola and Koskinen, 2013), CodeGrade (CodeGrade, 2021), Coderunner (Coderunner, 2021), GitHub Classroom (GitHub Classroom, 2021), Gradescope (Gradescope, 2021), Repl.it (replit, 2021), Submitty (Submitty, 2021), TIM (TIM, 2021), and VPL (VPL, 2021)

Table 7: Matrix scoring on tool suitability

Tool	*Terminal Programming*	*Web Development*	*Mobile Development*	*External Tools*	*Running total*
A+	**X**	**X**	**X**	**X**	**4**
CodeGrade	**X**	**X**	**X**	**X**	**4**
Coderunner	**X**	**-**	**-**	**-**	**1**
GitHub Classroom	**X**	**X**	**X**	**X**	**4**
Gradescope	**X**	**X**	**X**	**X**	**4**
Repl.it	**X**	**X**	**X**	**X**	**4**
Submitty	**X**	**X**	**X**	**X**	**4**
TIM	**X**	**-**	**-**	**X**	**2**
VPL	**X**	**-**	**-**	**X**	**2**

In Table 7, the tools are considered for how they function for assessment in different environments. All the tools that would not satisfy these environments were rejected before based on the constraint of having to suit for complex programming tasks. However, a stricter base was taken here, and points would only be given if the tool would already be fit for environments found in university. Coderunner was the only tool that clearly would not function for web & mobile development. VPL, TIM, and TMC have a system that could support these environments, but no pre-existing use cases were found that would support the needs of the stakeholders of LUT University.

Table 8: Matrix scoring on tool maintainability

Tool	*SSO Login*	*LTI*	*Usage in Finnish Universities*	*External Hosting*	*Running total*
A+	**X**	**X**	**X**	**-**	**7**
CodeGrade	**X**	**X**	**X**	**X**	**8**
Coderunner	**X**	**X**	**X**	**-**	**4**
GitHub Classroom	**X**	**X**	**-**	**X**	**7**
Gradescope	**X**	**X**	**-**	**X**	**7**
Repl.it	**-**	**-**	**X**	**-**	**5**
Submitty	**-**	**-**	**-**	**-**	**4**
TIM	**X**	**-**	**X**	**-**	**4**
VPL	**X**	**X**	**X**	**-**	**5**

Table 8 compares the tools for their maintainability. Points on SSO login and LTI were given here if the systems had any documentation on conforming to these requirements. Some systems such as VPL and Coderunner were Moodle-plugins, easily gaining full marks on these categories. Most systems provided features for both requirements, while others were found lacking. On usage in Finnish universities, points would be given if any source of information could be found, whether it was through searching online or from talks with Finnish universities – as pointed in chapter 4.1. Finally, points for external hosting would be given here if the system could be hosted fully elsewhere. Even if some systems would offer demos or other ways of testing the system without having it hosted at LUT, if this was not a permanent solution, no points would be given. The benefits of corporate systems shine for this requirement, as they would be the only systems with a chance for external hosting.

Table 9: Matrix scoring on tool features

Tool	*Plagiarism Detection*	*Unlimited Submissions*	*Resource Management*	*Git Submissions*	*Running Total*
A+	**X**	**X**	**X**	**X**	**11**
CodeGrade	**X**	**X**	**X**	**X**	**12**
Coderunner	**-**	**X**	**-**	**-**	**5**
GitHub Classroom	**-**	**X**	**X**	**X**	**10**
Gradescope	**X**	**X**	**X**	**-**	**10**
Repl.it	**-**	**X**	**-**	**X**	**7**
Submitty	**-**	**X**	**X**	**X**	**7**
TIM	**-**	**X**	**-**	**-**	**5**
VPL	**X**	**X**	**X**	**-**	**8**

Table 9 shows how the tools support different functionalities required by the stakeholders. With these requirements, it is important to remind, that points were given for the tools based on resources available and based on what is currently possible. If the tool had lacking documentation and if use cases for these features could not be found, no points would be given. Additionally, as mentioned before, many of the tools can be extended to cover a variety of features. However, if the tool did not yet support a feature and no such use-case could be found, no points would be given. With this table, both the documentation, but also the actual use of the tools would matter, as only features that can be adopted by LUT University are features that truly matter.

Table 10: Matrix scoring on tool assessment properties

Tool	*Automated Assignment Feedback*	*Possibility for Manual Feedback*	*Possibility for Manual Assessment*	*Running Total*
A+	**X**	**-**	**X**	**13**
CodeGrade	**X**	**X**	**X**	**15**
Coderunner	**X**	**-**	**-**	**6**
GitHub Classroom	**X**	**X**	**-**	**12**
Gradescope	**X**	**X**	**X**	**13**
Repl.it	**X**	**X**	**-**	**9**
Submitty	**X**	**X**	**X**	**10**
TIM	**X**	**X**	**X**	**8**
VPL	**X**	**X**	**X**	**11**

Finally, in Table 10, the richness of feedback provided was considered. Automated assignment feedback was given by all the tools, as one would expect from an AAS. This does not mean that all feedback is equal, but here points were given with no regard to how well the automated feedback could be formed. Some discrepancies could be found with the manual options of feedback. These tables allowed to present a list of the best tools for the job based on the requirements of LUT University. Scores yielded provided much needed quantitative measure on the suitability of the tools.

The tools listed can be divided by four distinct properties. Open versus closed and focused versus broad. Tools in the ‘open’ category are typically community created & maintained tools. They do not have a licensing fee, but also lack dedicated customer support among other paid services. Open tools also require for the university to figure out hosting of the services. In the closed category are services that are maintained by a corporation & therefore kept closed from their customers. These tools typically provide a better solution for the client, but the toolset comes at a cost and with an agreed upon level of service. Tools in the ‘focused’ category are tools that fit for a certain problem well, while those tools in ‘broad’ are tools that could be used for a variety of problems in the university’s domain. To put it in other words, tools in focused category could be used for some of the university’s needs but not all, while those in broad category were found to be useful for many different needs. In table 11 the highest ranked tools for these categories are considered. In the headers on both

top and left are the names of the categories, while tools have been laid out based on the category that they fit the best.

Table 11: Categorization of considered tools

	Open	***Closed***
Broad	A+ Submitty	CodeGrade GradeScope GitHub Classroom Repl.it
Focused	VPL Autograder TIM Coderunner	Viope

For the stakeholders' purposes, the tools in broad & closed categories were the most fitting. Tools in focused categories did not seem to fit the needs of the university well enough. This was covered by the key requirements R7-R10, which considered the usage of the tool for different tasks. The way that the tools manage this, whether the tool is closed or open is not the most important factor. For maintenance of the system, closed tools are preferred, as they conform better to requirements QR8 & R13. Closed systems managed by a corporation typically also felt more robust and better managed which ties into the usability by maintenance stakeholders, and security policies imposed by management stakeholders.

This categorization, as well as the matrix listings shown before were discussed with the university. Four tools from the list were selected for testing. These were A+, CodeGrade, GitHub Classroom, and VPL.

5.1.1 Tools selected for testing

From the AAS's used in Finnish universities, **A+** stands out. It has been developed at the largest technical university of Finland, Aalto University (Karavirta, Ihantola and Koskinen,

2013). It has been in use there for many years now and has been since used in Tampere University as well under the name Plussa. This already makes it one of the most used automated assessment systems in Finland. As for features, A+ has been created to be broad in its provided feature set. Some tools created in universities have been focused on their current problem scope. A+ seems to have done a good job at providing a system that can be used for many different problems out of the box – i.e., without having to take a deep dive to the system and develop features that are not yet supported.

CodeGrade is a corporate solution for the problem. Its feature set was deemed the best out of the systems before further testing. It also provided a great on-demand support, an extensive documentation, and well thought out tutorials. Hosting would be supported by them, and as the business provides automated assessment for several universities, the servers should have a good capacity. This would all come with a price, but during testing it would remain to be seen whether the features cover the price-tag.

GitHub Classroom is an extension to GitHub, adding together many different technologies to form a virtual classroom for programming. Namely, it uses GitHub (GitHub, 2021a) as a version control to manage student submissions, GitHub Actions (GitHub, 2021b) to provide automation on each student version committed to the version control, and an LMS integration to export the results to Moodle for instance. Currently, GitHub is the most common version control in use at Finland, and as such the integration with GitHub was seen as a big plus for the system.

VPL (Virtual Programming Lab) is a free Moodle plugin that allows code to be run on the server that Moodle is hosted in. The main interest for testing is in its usage at LAB university of applied sciences, which cooperates with LUT under LUT Group. Cooperation with LAB is an important point for many of the stakeholders. For the research team involved, support made available meant a great deal.

5.1.2 Additional Considerations

GradeScope seemed like a good option, with feature set at least as good as other options. GradeScope could also have catered other fields of study as well, such as automating parts

of natural science teaching. However, there was no indication that GradeScope would be used in Finland at all, and as such, it was not taken to test.

TIM is a platform for publishing interactive lecture materials. For this job, it functions well, and could be used at LUT University. However, for automated assessment, its functionalities remain limited, and it was deemed not fit for the stakeholders.

Coderunner is a Moodle plug-in, that provides Moodle quizzes a new question type. What this provides, is a way to test students small code snippets. This is not the primary need for the software engineering department of LUT University, as it is not suitable for the assessment of complex programs. Tools like this, however, could be a large aid for teaching.

Submitty and **Repl.it** were the contenders for other systems. Submitty was perceived to provide less than A+, and Repl.it was compared to CodeGrade. Both systems were also not widely used in Finland, where as CodeGrade and A+ had clear use cases to refer.

5.2 Suitability of tools

To further compare the tools and determine their suitability, demo-courses were created. The main goal for the demo courses was to show how well they could handle different assignments. Assignments were modified from those that have been used at the corresponding courses at the university. The languages used were Python, C, Java, and JavaScript. These are used at the university at corresponding courses; Introduction to Programming, Principles of C–programming, Object-Oriented Programming, and Web Applications. The Java assignment was aimed at mobile programming through Android Studio, while assignment with JavaScript was aimed at web development through Node.js.

After demo-courses were created, the tools could be further ranked based on their perceived suitability. In this section, the tools tested are compared from different viewpoints. The first point of interest is the tool’s suitability in contrast to demo-course goals. These goals are stated in the second column of Table 12. The first column shows which assignment was used to determine the stated functional goal. The next four columns state the points assigned to tools given based on the demo-course. In the last column, the existing tool at the university

is compared to, based on the opinions of the system's users. Two points were given if the tool was deemed suitable for the functionality. One point would be given if there was an excess amount of work to be put in, or if the functionality could not be made within the timeframe of the testing. 0 points were given for the functionalities that tools could not perform at.

Table 12: Tool suitability on demo-course goals

Assignment	*Functionality*	*A+*	*Codegrade*	*Github Classroom*	*VPL*	*Viope*
Python	Simple I/O Testing	2	2	0	2	2
	Unit Testing	2	2	1	1	0
C	Memory Control through Valgrind	2	2	2	1	0
	Command-line parameters	2	2	2	2	2
Web	Git Submission	2	2	2	1	0
	NPM Install	2	2	2	1	0
Android	Per-Assignment Installation scripts	2	2	0	1	0
	Device Emulation	2	0	2	0	0
	Total	***16***	***14***	***11***	***9***	***4***

The survey results of chapter 3.3 gave a baseline for the requirements that were perceived the most useful by LUT University's end users. In Table 13, these requirements are considered again to make a comparison between the tools. Each tool would be given points on a scale from 0 to 2 based on how well they would fill the requirement at hand. The final points were multiplied by weights determined on survey results. Full descriptions and all points given in this table are available in appendix 4.

Table 13: Tool suitability on survey results

Section	*A+*	*Codegrade*	*GitHub Classroom*	*VPL*
Functionality	70.87	65.87	51.86	55.33
Quality	41.5	48.25	38.375	21.875
Assignment support	2	1	1	1
Programming Language Support	1.75	1.25	1.75	0.75

As interest peaked during the final selection of the tool, many relevant parties became active in the selection process. One of these parties was a usability expert at SWE, a stakeholder group determined in chapter 3.2. Many questions and points of interest for the tool were gathered by them, while answers were given by our research team. Once again, in Table 14 the tools are compared based on relevant attributes. Points were given by the research team from 0-2 based on how well they conformed to the requirements at hand. Full list of relevant features is available in appendix 5. In this table, the average scores for each category are shown. A comparison can be made within each category, but an overall comparison is an issue due to categories not sharing the same importance.

Table 14: Tool suitability on usability expert features

Area	***A+***	***CodeGrade***	***Github Classroom***	***VPL***	***Viope***
Testing Capabilities	1.92	1.69	1.77	1.31	0.62
Submission Features	1.14	1.43	0.57	1	0.71
Student Feedback	1.38	1.78	1.78	1.11	0.44
Additional Features	1.6	1.2	0.8	1.2	1.6
Plagiarism Detection	1.2	2	0	2	0.8
Implementation and Usage	1.11	1.22	1.11	1	1.22
Moodle Integration	0.67	2	0.67	2	2
Course Administration	1	1.6	0.2	1.4	1.4

5.3 Tool Decision

From the tables in the last section, it is apparent that two of the tools were more suitable to the university's stakeholders than the other two. Namely, the choice would end up being between A+ and CodeGrade. GitHub Classroom is compared to Codegrade due to their similar pricing. Codegrade handles Moodle-integration better, has a better user experience for a student, and has shown the quality of their support. It is preferred over GitHub Classroom for these reasons. VPL had problems for fitting to assignments with GUI. It is

also not clearly better than the existing solution for CS1 courses, which is Viope. As such, it can also be discarded as a fitting tool for the university.

Tools were presented to the university in a seminar, as well as written pdf. The audience included both the LUT University software engineering department, as well as people from other related departments, and people from LAB. The demos created were made open to everyone, which helped with decision. The seminar was well received, and this finally sparked a serious discussion for all related courses. The need for a new automated assessment system on advanced programming courses was recognized. Discussion was still left open whether any of the tools were better than the existing one in the context of the introductory programming courses. These opinions are in-line with the research findings. This leads to believe, that the process and results of the research were communicated with the department well enough to be on the same page with the outcome.

After seminar, the teachers were given a few more weeks for reaching a decision. During this time, talks were had in different parts of the department to clarify anything left unclear. After this short thinking period, the tool chosen was CodeGrade. Main selling points of the system were its usability for both students and teachers, as well as being externally hosted and maintained. It also conformed to the requirements well, providing a wide set of functionalities, support, documentation, and overall fine user experience. Android assessment is one of the few things that CodeGrade lacked. It was still left under consideration, whether another system such as A+ would be used to fill this need for the required courses.

6 DISCUSSION

In this chapter, the work done is reflected upon. In chapter 6.1, answers are given to the research questions laid out earlier in chapter 1.1.2. They will be considered for whether the research questions were filled, and whether work could have been done differently to better answer the questions. In the next chapter, thesis work is reflected upon for its implications to both the researchers and the practitioners of related fields. After these, a short look on the future of the area of automated assessment is considered.

6.1 Research Questions

1. What are the requirements of an automated assessment system implemented for the stakeholders of the university?

This question demanded a two-part answer. First, the stakeholders of the system were considered to make sure that everyone's opinions could be heard. This was done in chapter 3.2. Stakeholders were gathered slowly during the first stages of work. There could have been more work done in making sure that all stakeholders are identified before carrying on with rest of the work. In the end however, in my opinion all of the relevant stakeholders were considered. Some work was alleviated due to SWE Department holding accountability on the tool decision, and they would work on their own set of stakeholders. For this thesis, those stakeholders were simply ruled as other regulatory authorities. Were the system developed from scratch, and if there was more work done for the system, the stakeholder gathering phase would have been much more thorough. For the purposes of this thesis, the 12 stakeholder types and 4 stakeholder groups were sufficient.

The set of requirements of the system is large. These requirements included for example the functional features of the tool, as well as the non-functional quality aspects. In addition to the features of the tool, cooperation between different stakeholders, such as information services and other universities, were crucial. These requirements were considered further in chapters 3.1, 3.3, and 3.4. Before any stakeholders were considered, the current automated assessment environment of the system was covered. This was a good idea, as learning from

the existing system gave some key constraints for work. During requirements gathering, the most important stakeholder was deemed to be teachers. This has a risk of the tool chosen not being suitable for students. On the other hand, the tool is to be taken as a demo for the next year, during which its suitability with all relevant stakeholders is considered. Gathering methods used were in my opinion sufficient, even if they could have been better. The role of the questionnaire was discussed further in chapter 3.3.2.

2. What tools are available for automated assessment?

Chapter 4 reviewed the tools available for automated assessment. 30 different tools were considered for the task, of which 9 tools fit the requirements of the stakeholders well. Tool gathering was done through an open search from online resources. In addition, the automated assessment environment in Finland was considered. While there were many tools found for the task, an equal number of tools was most likely missed as well. In the end though, only a few tools were sufficient for the requirements gathered. This leads to believe that the best automated assessment systems for LUT University's needs were found. Due to time constraints, most of the tools were considered based on SWE's needs. This begs the question of how the results would have differed, if other stakeholders of LUT University, LUT-Group, or other universities were further considered? Systems like GradeScope could have been more useful for LUT University as a whole and adopting to systems like A+ or TIM could have improved the cooperation of universities.

3. How do the tools available conform to the requirements presented by the stakeholders of the university?

Chapter 5 was dedicated for work related to the acceptance of the requirements & tools gathering processes. Demo course was made for the chosen tools. The demo courses served well as a way of gaining more understanding on the tools. For the most invested stakeholders, the demos also granted an opportunity to further their knowledge. After testing of the demo course was finished, the tools were ready to be presented for a wider audience. Some of the tools proved to be unfit for the task. This could have perhaps been found out earlier, if the review based on documentation found was more thorough for all tools. In the end, two tools proved to be capable conforming to the requirements of the stakeholders well. In my opinion

the tools found were the best tools available on the market for the job. The results should give the same impression to anyone reading through this thesis.

6.2 Implications to literature and practise

This thesis is by no means a comprehensive look on the tools available for automated assessment, the history of it, or the methods related. It is not a systematic literature review, nor should it be taken as such. What this thesis offers is a point of reference, for both researchers and practitioners alike.

For the researcher, this thesis includes the key articles to refer in regard to automated assessment and the development of an automated assessment system. A large portion of the literature review was based on categorization and methodology introduced by Ala-Mutka (2005). Potential pitfalls as branded and described by Pieterse (2013) are a great resource for anyone dealing with the design or implementation process of these systems. These articles denote how systems of high quality create a potential for enhanced student learning. Whether this potential is tapped into in practise is reviewed through examples found in relevant literature. The need for both dynamic and static assessment is underlined, as they play a part in factors such as code correctness, security, plagiarism, and submission feedback. Finally, while there is a large amount of literature considering the design process of AAS's, literature for comparison and selection process of such systems remain limited. What this thesis offers is discussion on our requirements engineering process, from motivation to tools gathering and the final selection.

As for the practitioner, the implications of the work put into the thesis are much greater. This is especially true for other practitioners in Finland. It includes a shortlist of notable tools available, including a section on the tools in use at other local universities. Many hours of work and effort show in the list of requirements gathered, and in the comparison tables between tools on how these requirements are filled. Work with stakeholders revealed the multitude of relevant stakeholders that are related to system design, development, and implementation. These aspects were well received by the main stakeholder of the thesis, the software engineering department of LUT University. For them, and their stakeholders, the

work done here will bear fruit through a revised and upgraded automated assessment environment.

6.3 Future

The area of automated assessment is now more important than ever. Talks with different universities and other interested parties have shown, that the problems at LUT University are by no means unique. There is still a lot to learn on the automated assessment environment, not only at LUT University, but in all environments that deal with mass-courses related to software engineering.

At LUT university, the chosen tool will be taken to use during for the semester following this research. There is another master's thesis (Järventausta, N.D.) being made on the implementation process of selected tool. There is still a lot of work to be done to implement the tool for all related courses. LAB and other groups dealing with programming were also interested in implementing new tools for automated assessment. It remains a question, whether the selected tool will need to be implemented for parties other than SWE as well.

Since other universities and communities share similar problems with our university, there is a great need for unifying the efforts of creating and implementing new systems. Thus far, it has been a trend that all communities either make their own research for a fitting tool or make a completely new tool for their needs. This is not the best way of going forward. Systems like A+ could be made nation-wide with a possibility for everyone to collaborate through open development.

7 SUMMARY

In this thesis, the feature evaluation and selection process of an automated assessment system in university context was carried out, as the title suggested. In the introduction, the motivation behind the thesis work was described and delimitations were set. Goals for the research were set, and fitting research methods for these goals were described.

In the literature review, a look was taken on two things. First, the automated assessment environment was investigated. From this, a set of features for automated assessment, as well as potential pitfalls were determined. Secondly, requirements engineering process was researched to learn how best to conclude a successful selection process that would take the many stakeholders of the system into account. Literature review was concluded, and the research was left with a good base for how to continue with the rest of the research, which was more real-world oriented.

The next steps for work were to gather the requirements of the system, as well as the tools filling the requirements. After interacting with the stakeholders at the university, as well as stakeholders in other similar communities, the gathered tools could be ranked based on the most valued features of the tool. Upon reviewing these tools with the university, demonstrative courses could be made, that would aid in the final selection of a system. The choice was made by the university based on our, and the teaching staff's opinions. Implementation of the tool is the next step, which will be covered in another thesis.

LIST OF REFERENCES

A+ (2021) *A+, A+ LMS*. Available at: https://apluslms.github.io/ (Accessed: 29 June 2021).

Abelló, A. *et al.* (2016) 'A Software Tool for E-Assessment of Relational Database Skills', *International Journal of Engineering Education*.

Ala-Mutka, K. M. (2005) 'A Survey of Automated Assessment Approaches for Programming Assignments', *Computer Science Education*, 15(2), pp. 83–102. doi: 10.1080/08993400500150747.

Asnar, Y., Giorgini, P. and Mylopoulos, J. (2011) 'Goal-driven risk assessment in requirements engineering', *Requirements Engineering*, 16(2), pp. 101–116. doi: 10.1007/s00766-010-0112-x.

Autogradr (2021) *Autogradr*. Available at: autogradr.com (Accessed: 1 February 2021).

Barker-Plummer, D., Dale, R. and Cox, R. (2021) 'Using Edit Distance to Analyse Errors in a Natural Language to Logic Translation Corpus'.

Barra, E. *et al.* (2020) 'Automated Assessment in Programming Courses: A Case Study during the COVID-19 Era', *Sustainability*, 12(18), p. 7451. doi: 10.3390/su12187451.

Baskerville, R. L. (1999) 'Investigating Information Systems with Action Research', *Communications of the Association for Information Systems*, 2. doi: 10.17705/1CAIS.00219.

Chai (2021) *Chai*. Available at: https://www.chaijs.com/ (Accessed: 29 June 2021).

Chung, L. and Do Prado Leite, J. C. S. (2009) 'On non-functional requirements in software engineering', *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 5600 LNCS, pp. 363–379. doi: 10.1007/978-3-642-02463-4_19.

CodeGrade (2021) *CodeGrade - Deliver Engaging Feedback on Code.* Available at: https://www.codegrade.com/ (Accessed: 29 June 2021).

CodeGrades (2021) *CodeGrades*. Available at: https://codegrades.com/ (Accessed: 28 June 2021).

CodeHS (2021) *CodeHS - Teach Coding and Computer Science at Your School | CodeHS*. Available at: https://codehs.com/ (Accessed: 28 June 2021).

Coderunner (2021) *CodeRunner*. Available at: https://coderunner.org.nz/ (Accessed: 29 June 2021).

Conejo, R., Barros, B. and Bertoa, M. F. (2019) ‘Automated Assessment of Complex Programming Tasks Using SIETTE’, *IEEE Transactions on Learning Technologies*, 12(4), pp. 470–484. doi: 10.1109/TLT.2018.2876249.

Conejo, R., Guzmán, E. and Trella, M. (2016) ‘The SIETTE Automatic Assessment Environment’, *International Journal of Artificial Intelligence in Education*, 26(1), pp. 270–292. doi: 10.1007/s40593-015-0078-4.

CSES (2021) *CSES*. Available at: https://cses.fi/ (Accessed: 28 June 2021).

Cypress (2021) *JavaScript End to End Testing Framework, JavaScript End to End Testing Framework | cypress.io*. Available at: https://www.cypress.io/ (Accessed: 27 June 2021).

Daradoumis, T. *et al.* (2019) ‘Analyzing students’ perceptions to improve the design of an automated assessment tool in online distributed programming’, *Computers & Education*, 128, pp. 159–170. doi: 10.1016/j.compedu.2018.09.021.

Fernández, D. M. *et al.* (2017) ‘Naming the pain in requirements engineering: Contemporary problems, causes, and effects in practice’, *Empirical Software Engineering*, 22(5), pp. 2298–2338. doi: 10.1007/s10664-016-9451-7.

GatorGrader (2021) *GatorGrader, GatorGrader*. Available at: https://www.gatorgrader.org/ (Accessed: 28 June 2021).

GitHub (2021a) *Build software better, together, GitHub*. Available at: https://github.com (Accessed: 29 June 2021).

GitHub (2021b) *GitHub Actions Documentation - GitHub Docs*. Available at: https://docs.github.com/en/actions (Accessed: 29 June 2021).

GitHub Classroom (2021) *GitHub Classroom*. Available at: https://classroom.github.com/ (Accessed: 29 June 2021).

Glassman, E. L. *et al.* (2015) ‘OverCode: Visualizing Variation in Student Solutions to Programming Problems at Scale’, *ACM Transactions on Computer-Human Interaction*, 22(2), p. 7:1-7:35. doi: 10.1145/2699751.

Gradescope (2021) *Gradescope | Save time grading*. Available at: https://www.gradescope.com/ (Accessed: 29 June 2021).

Hollingsworth, J. (1960) ‘Automatic graders for programming classes’, *Communications of the ACM*, 3(10), pp. 528–529. doi: 10.1145/367415.367422.

Hull, E., Jackson, K. and Dick, J. (2011) *Requirements engineering (third edition)*. (Requirements Engineering (Third Edition)), p. 207. doi: 10.1007/978-1-84996-405-0.

Hyper Grade (2021) *Hyper Grade - Automated Code Testing and Grading*. Available at: https://www.hypergrade.com/ (Accessed: 28 June 2021).

‘ISO/IEC/IEEE International Standard - Systems and software engineering–Vocabulary’ (2017) *ISO/IEC/IEEE 24765:2017(E)*, pp. 1–541. doi: 10.1109/IEEESTD.2017.8016712.

Järventausta, A. (N.D.).

Karavirta, V., Ihantola, P. and Koskinen, T. (2013) ‘Service-Oriented Approach to Improve Interoperability of E-Learning Systems’, in *2013 IEEE 13th International Conference on Advanced Learning Technologies*. *2013 IEEE 13th International Conference on Advanced Learning Technologies*, pp. 341–345. doi: 10.1109/ICALT.2013.105.

Knutas, A. *et al.* (2019) ‘Constructive Alignment of Web Programming Assignments and Automated Assessment with Unit Testing’, in *Proceedings of the 19th Koli Calling International Conference on Computing Education Research*. *Koli Calling ’19: 19th Koli Calling International Conference on Computing Education Research*, Koli Finland: ACM, pp. 1–2. doi: 10.1145/3364510.3366150.

Kratzke, N. (2019) ‘Smart Like a Fox: How Clever Students Trick Dumb Automated Programming Assignment Assessment Systems’:, in *Proceedings of the 11th International Conference on Computer Supported Education*. *11th International Conference on Computer Supported Education*, Heraklion, Crete, Greece: SCITEPRESS - Science and Technology Publications, pp. 15–26. doi: 10.5220/0007424800150026.

Kyrilov, A. and Noelle, D. C. (2016) ‘Do students need detailed feedback on programming exercises and can automated assessment systems provide it?’, *Journal of Computing Sciences in Colleges*, 31(4), pp. 115–121.

Liang, Y. *et al.* (2009) ‘The Recent Development of Automated Programming Assessment’, in *2009 International Conference on Computational Intelligence and Software Engineering*. *2009 International Conference on Computational Intelligence and Software Engineering*, pp. 1–5. doi: 10.1109/CISE.2009.5365307.

Luukkainen, R. (2020) ‘ASPA : a static analyser to support learning and continuous feedback on the first programming course’. Available at: https://lutpub.lut.fi/handle/10024/161320 (Accessed: 14 June 2021).

Mekterović, I. *et al.* (2020) ‘Building a Comprehensive Automated Programming Assessment System’, *IEEE Access*, 8, pp. 81154–81172. doi: 10.1109/ACCESS.2020.2990980.

Mocha (2021) *Mocha - the fun, simple, flexible JavaScript test framework*. Available at: https://mochajs.org/ (Accessed: 29 June 2021).

Moodle (2021) *Moodle - Open-source learning platform | Moodle.org*. Available at: https://moodle.org/?lang=fi (Accessed: 27 June 2021).

MOSS (2021) *MOSS*. Available at: https://theory.stanford.edu/~aiken/moss/ (Accessed: 29 June 2021).

Noonan, R. (2006) *The back end of a grading system*, p. 60. doi: 10.1145/1121341.1121360.

Pfleeger, S. L. and Kitchenham, B. A. (2001) ‘Principles of survey research: part 1: turning lemons into lemonade’, *ACM SIGSOFT Software Engineering Notes*, 26(6), pp. 16–18. doi: 10.1145/505532.505535.

Pieterse, V. (2013) ‘Automated Assessment of Programming Assignments’, in, pp. 45–56.

Polito, G., Temperini, M. and Sterbini, A. (2019) ‘2TSW: Automated Assessment of Computer Programming Assignments, in a Gamified Web Based System’, in *2019 18th International Conference on Information Technology Based Higher Education and Training (ITHET)*. *2019 18th International Conference on Information Technology Based Higher Education and Training (ITHET)*, pp. 1–9. doi: 10.1109/ITHET46829.2019.8937377.

replit (2021) *The collaborative browser based IDE, replit*. Available at: https://replit.com/ (Accessed: 29 June 2021).

Runeson, P. and Höst, M. (2008) ‘Guidelines for conducting and reporting case study research in software engineering’, *Empirical Software Engineering*, 14(2), p. 131. doi: 10.1007/s10664-008-9102-8.

Schlarb, M., Hundt, C. and Schmidt, B. (2015) ‘SAUCE: A Web-Based Automated Assessment Tool for Teaching Parallel Programming’, in Hunold, S. et al. (eds) *Euro-Par 2015: Parallel Processing Workshops*. Cham: Springer International Publishing (Lecture Notes in Computer Science), pp. 54–65. doi: 10.1007/978-3-319-27308-2_5.

Schleimer, S., Wilkerson, D. S. and Aiken, A. (2003) ‘Winnowing: Local Algorithms for Document Fingerprinting’, p. 10.

Siochi, A. C. and Hardy, W. R. (2015) ‘WebWolf: Towards a Simple Framework for Automated Assessment of Webpage Assignments in an Introductory Web Programming Class’, in *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*. New York, NY, USA: Association for Computing Machinery (SIGCSE ’15), pp. 84–89. doi: 10.1145/2676723.2677217.

Skalka, J. and Drlik, M. (2020) ‘Automated Assessment and Microlearning Units as Predictors of At-Risk Students and Students’ Outcomes in the Introductory Programming Courses’, *Applied Sciences*, 10(13), p. 4566. doi: 10.3390/app10134566.

STACK (2021) *Moodle plugins directory: STACK*. Available at: https://moodle.org/plugins/qtype_stack (Accessed: 28 June 2021).

Štajduhar, I. and Mauša, G. (2015) ‘Using string similarity metrics for automated grading of SQL statements’, in *2015 38th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*. *2015 38th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, pp. 1250–1255. doi: 10.1109/MIPRO.2015.7160467.

Staubitz, T. *et al.* (2015) ‘Towards practical programming exercises and automated assessment in Massive Open Online Courses’, in *2015 IEEE International Conference on Teaching, Assessment, and Learning for Engineering (TALE)*. *2015 IEEE International*

Conference on Teaching, Assessment, and Learning for Engineering (TALE), pp. 23–30. doi: 10.1109/TALE.2015.7386010.

Submitty (2021) *Welcome, Submitty*. Available at: https://submitty.org/index/overview (Accessed: 29 June 2021).

SYLVA (2021) *SYLVA | Better Teaching Within Your Reach*. Available at: https://www.sylva.ac/ (Accessed: 28 June 2021).

TIM (2021) *Welcome - TIM*. Available at: https://tim.jyu.fi/ (Accessed: 29 June 2021).

Van Lamsweerde, A. (2000) 'Handling obstacles in goal-oriented requirements engineering', *IEEE Transactions on Software Engineering*, 26(10), pp. 978–1005. doi: 10.1109/32.879820.

Viope (2021) *Viope*. Available at: https://www.viope.com (Accessed: 1 February 2021).

Vipunen - Education Statistics Finland (2021) *Opiskelijat ja tutkinnot, University Students*. Available at: https://vipunen.fi/fi-fi/yliopisto/Sivut/Opiskelijat-ja-tutkinnot.aspx (Accessed: 20 May 2021).

VPL (2021) *VPL - Virtual Programming Lab - Home*. Available at: https://vpl.dis.ulpgc.es/ (Accessed: 29 June 2021).

Vujosevic Janicic, M., Tošić, D. and Kuncak, V. (2013) 'Software Verification and Graph Similarity for Automated Evaluation of Students' Assignments', *Information and Software Technology*, 55, pp. 1004–1016. doi: 10.1016/j.infsof.2012.12.005.

Wang, T. H. *et al.* (2004) 'Web-based Assessment and Test Analyses (WATA) system: development and evaluation', *Journal of Computer Assisted Learning*, 20(1), pp. 59–71. doi: 10.1111/j.1365-2729.2004.00066.x.

Zmiev, S. (2021) *assignment-autograder: Automatic assignment grading for instructor use in programming courses*.

Zombie (2021) *Zombie.js, Zombie*. Available at: http://zombie.js.org/ (Accessed: 29 June 2021).

Appendix 1: Questionnaire Specification

Questionnaire Specification

Automated Assessment Systems questionnaire
28/01/20
Petteri Mäkelä

Questionnaire Objectives

Literature Review on automated assessment systems gives a good outlook on typical features of assessment systems. With this questionnaire, this information is presented into the University to reach an understanding of the important features for them. In addition, the questionnaire will try and answer what makes the teacher most comfortable in using an assessment system.

Design Rationale of Topics and Questions

Objectives of the questionnaire were condensed into few topics that were laid out by the objectives. The questionnaire was designed with these topics in mind:

1. Which of the usual AA features are the most important in our university's context?
    - Ranking functional features
    - Ranking non-functional features
2. In what different environments could the system be used?
    - Support for different (programming) languages
    - Support for different assignment types
3. What makes the course staff most comfortable with the system?
    - Tools used

In regard to the topics, 10 questions were formed.

1. Familiarity with Automated Assessment
2. The assessment system should consider the statements listed below

3. What other types of assessment should the system consider?
4. What other functional features should the system have?
5. Properties of the new system
6. I would use the automatic assessment system for activities stated below.
7. In which of the following environments would you consider using an automated assessment system?
8. For what other reasons, or in what other environments would you use the system in?
9. The system should take the following environments in to consideration.
10. State below if there are any other considerations to be made with the system environment.

In addition to questions, each section of the questionnaire had a small leading text of few sentences, that gave background information of the topic in question. This was made, so that respondent would be more comfortable answering the questions stated, especially if some parts of AA systems are unknown.

1. Familiarity with Automated Assessment

Type: Closed, Choices: No/Yes

Background info asking for example, whether the respondent has used tools like this before. This will allow us to learn more about the system's potential userbase, as well as factor in the knowledge of the respondent for their answers.

2. The assessment system should consider the statements listed below

Type: Closed, Choices: Not at all important / Important / Very Important / No Opinion

Statements were given based on literature, as well as colleague's opinions on what features AA systems should or could have. The responses gained will allow us to rank different functional features based on how important they are for our university's staff.

3. What other types of assessment should the system consider?

Type: Open

An open question to consider whether we have missed some ways of assessment that they system could conduct.

4. What other functional features should the system have?

Type: Open

An open question to consider whether we have missed some other functional features that the system should have.

5. Properties of the new system

Type: Closed, Choices: “Quick to set-up”, “Customizability for different tasks”, “Ease of use for students”, “The system has comprehensive learning resources available”

This question aims to answer what non-functional requirements are the most important for the teaching staff. Formulating a question for this was quite difficult, as most of the requirements are something that you would wish for a system. Using a priority system is common in requirements engineering, and that is what we have gone with on this question. The number of choices was kept low (5) to not make it overly difficult to rank the options.

6. I would use the automatic assessment system for activities stated below.

Type: Close, Choices: “Programming Exercises”, “Creating & Publishing Quizzes or Other Assignments”, “Creating & Publishing course material”, “Creating independent course components or whole courses”

The types of assignments created e.g., the scope of the system, is considered here. Answer options range from only programming exercises to whole courses. This lets us know, if there is a need for a system that does more than just assess code.

7. In which of the following environments would you consider using an automated assessment system?

Type: Closed,

Choices: “Mobile Programming, such as Android”, “Web languages and technologies, such as HTML & JavaScript”, “Text-Based/Terminal programming, such as Python and C”, “Database Management, such as SQL”, “Mathematics, such as MatLab”.

The choices here are typical uses for the system in our university. Most importantly, this question will serve as a starting point for everything that could be done with it, the open question after this might give more ideas on what could be done.

8. For what other reasons, or in what other environments would you use the system in?

Type: Open

Here, it’s hoped that many different answers will follow. It is hard to consider what different uses the University could have for an assessment system, other than what it has been used for so far, or what was considered for the initial requirements.

9. The system should take the following environments in to consideration.

Type: Closed, Choices: Not at all important / Important / Very Important / No Opinion

A few different points of interest have been stated here. The options were decided upon initial requirements given by the University. This question is targeted to a few specific worries that we had about the system. Also, this will trigger more ideas for the next open question.

10. State below if there are any other considerations to be made with the system environment.

Type: Open

An open question to consider whether there are some considerations about the system environment that we have not thought about.

Data analysis

Data can be exported in a csv file from the survey environment. Firstly, the validity of the questionnaire is considered based on the number of respondents, and the quality of their responses. The open questions will be evaluated to determine whether there is something of use to learn, or to use for the requirements of the system. Closed questions will be used to give a features list of what is important for the system for the teachers in the university. The most valued features of the system will be used in a feature-ranking matrix of different systems.

Appendix 2: Questionnaire Results

Question 1: Familiarity with automated asssessment

Question	*No*	*Yes*	*Average*	*Median*
I am part of courses that include programming tasks and assignments	0%	100%	2.00	2
I have used Automated Assessment tools before	25%	75%	1.75	2
I have created test cases for Automated Assessment tools before	25%	75%	1.75	2
I have designed tasks and/or courses that use an Automated Assessment system	25%	75%	1.75	2

Question 2: The assessment system should consider the statements listed below

Question	*No Opinion*	*Not at all important*	*Important*	*Very Important*	*Average*	*Median*
The program must function as specified	0%	0%	0%	100%	4.00	4
The program must be checked for plagiarism	12.50%	0%	12.50%	75%	3.50	4
The student must be given automated feedback	0%	12.50%	25%	62.50%	3.50	4
There must be an option for manual assessment (the scoring of a submission)	0%	0%	62.50%	37.50%	3.38	3
The program must handle resources, such as computer memory right (e.g pointing to memory areas and file interactions)	0%	14.28%	42.86%	42.86%	3.29	3
The program must allow resubmitting the solution for as long as it's necessary	0%	0%	87.50%	12.50%	3.13	3
There must be an option for manual feedback	0%	12.50%	75%	12.50%	3.00	3
Automated feedback must given for submission's specific issues	12.50%	12.50%	50%	25%	2.88	3
The program code must follow stated coding style	0%	25%	75%	0%	2.75	3
The program must include/exclude given keywords	25%	12.50%	50%	12.50%	2.50	3
The run time of the program must be calculated	0%	62.50%	37.50%	0%	2.38	2
The program must not contain dead code (e.g unused parts of the code) or other common errors	0%	75%	25%	0%	2.25	2
The program should follow the design of the model program	12.50%	50%	37.50%	0%	2.25	2

Question 5: Properties of the new system

Option	*1*	*2*	*3*	*4*	*5*	*Average*
Quick to set-up	25%	25%	25%	0%	25%	2.75
Customizability for different tasks	0%	0%	12.50%	50%	37.50%	4.25
Ease of use for students	0%	0%	25%	37.50%	37.50%	4.13
Usage of tools that I am familiar with	50%	37.50%	12.50%	0%	0%	1.63
The system has comprehensive learning resources available	25%	37.50%	25%	12.50%	0%	2.25

Question 6: I would use the automatic assessment system for activities stated below

Question	*n*	*Percent*
Programming exercises	8	100%
Creating & publishing quizzes or other assignments	4	50%
Creating & publishing course material	3	37.50%
Creating independent course components or whole courses	1	12.50%

Question 7: In which of the following environments would you consider using an automated assessment system?

Question	*n*	*Percent*
Mobile Programming, such as Android	4	50%
Web languages and technologies, such as HTML & Javascript	4	50%
Text-based/Terminal programming, such as Python and C	6	75%
Database Management, such as SQL	3	37.50%
Mathematics, such as MatLab	3	37.50%

Question 9: The system should take the following environments in to consideration

Question	*No Opinion*	*Not at all important*	*Important*	*Very Important*	*Average*	*Median*
The submissions should be handled through a common version control (e.g Github or Bitbucket).	12.50%	25%	37.50%	25%	2.75	3
The tools used for creating the tests for the system should use tools that I already know.	0%	100%	0%	0%	2.00	2
The system must use tools that have previously been used for similar assignments (e.g C & cUnit, Java & jUnit).	25%	62.50%	12.50%	0%	1.88	2
The tools used with the system must not cause extra work in learning the system.	0%	62.50%	37.50%	0%	2.38	2
Assignments used for previous assessment types should be compatible with the new system.	37.50%	50%	12.50%	0%	1.75	2

Appendix 3: Gathered systems list

Website included if found, otherwise source paper is referred to

Name	***Source***
2TSW	(Polito, Temperini and Sterbini, 2019)
A+	https://apluslms.github.io/
Assignment-Autograder	https://pypi.org/project/assignment-autograder/
autoCOREctor	(Barra *et al.*, 2020)
Autogradr	https://autogradr.com/
CodeGrade	https://www.codegrade.com/
CodeGrades	https://codegrades.com/
Codehs	https://codehs.com/
CodePost	https://codepost.io/
Coderunner	https://coderunner.org.nz/t
CSES	https://cses.fi/
Edgar	(Mekterović *et al.*, 2020)
FitchFork	(Pieterse, 2013)
GatorGrader	https://www.gatorgrader.org/
GitHub Classroom	https://classroom.github.com/
Gradescope	https://www.gradescope.com/
Hypergrade	https://www.hypergrade.com/
MIT OverCode	(Glassman *et al.*, 2015)
Repl.it	https://replit.com/
SAUCE	(Schlarb, Hundt and Schmidt, 2015)
SIETTE	(Conejo, Barros and Bertoa, 2019)
Sphere Engine	https://sphere-engine.com/
STACK	https://moodle.org/plugins/qtype_stack
Submitty	https://submitty.org/index/overview
TIM	https://tim.jyu.fi/
TMC	http://testmycode.github.io/
UZH Sylva	https://www.sylva.ac/
VPL	https://vpl.dis.ulpgc.es/
Zybooks	https://www.zybooks.com/

Appendix 4: Final tools ranking

Weight column has the average score gathered through the survey for the question. This is then multiplied with our opinion on how well the system performs, which are listed in under the columns for the tools. Based on this, total score for the tools is formed on the bottom.

Requirement	***Weight***	***A+***	***CodeGrade***	***Github Classroom***	***VPL***
The program must function as specified	4.00	2	2	2	2
The program must be checked for plagiarism	3.50	2	2	0	1
The student must be given automated feedback	3.50	2	2	2	1
There must be an option for manual assessment (the scoring of a submission)	3.38	2	2	0	2
The program must handle resources, such as computer memory right (e.g pointing to memory areas and file interactions)	3.29	2	2	2	1
The program must allow resubmitting the solution for as long as it's necessary	3.13	2	2	2	2
There must be an option for manual feedback	3.00	2	2	2	2
Automated feedback must given for submission's specific issues	2.88	2	2	2	2
The program code must follow stated coding style	2.75	2	2	1	1
The program must include/exclude given keywords	2.50	2		1	1
The run time of the program must be calculated	2.38	2	2	2	2
The program must not contain dead code (e.g unused parts of the code) or other common errors	2.25	1	1	1	1
The program should follow the design of the model program	2.25	0	0	0	0
	77,62	**70.87**	**65.87**	**51.86**	**55.33**

Here, the items would be ranked with 5 being the most important and 1 the least. To calculate scores, we ranked the tools to correspond to the topic – 4 conforming the best and 1 the least.

Non-Functional Requirement	***Weight***	***A+***	***Codegrade***	***Github Classroom***	***VPL***
Quick to set-up	2.75	1	4	3	2
Customizability for different tasks	4.25	4	2	3	1
Ease of use for students	4.125	3	4	1	2
Usage of tools that I am familiar with	1.625	3	2	4	1
The system has comprehensive learning resources available	2.25	2	4	3	1
		41.5	**48.25**	**38.375**	**21.875**

Usage	***Weight***	***A+***	***Codegrade***	***Github Classroom***	***VPL***
Programming excercises	100%	1	1	1	1
Creating & publishing quizzes or other assignments	50%	1	0	0	0
Creating & publishing course material	37.50%	1	0	0	0
Creating independent course components or whole courses	12.50%	1	0	0	0
		2	**1**	**1**	**1**

Usage	***Weight***	***A+***	***Codegrade***	***Github Classroom***	***VPL***
Mobile Programming, such as Android	50%	1	0	1	0
Web languages and technologies, such as HTML & Javascript	50%	1	1	1	0
Text-based/Terminal programming, such as Python and C	75%	1	1	1	1
		1.75	**1.25**	**1.75**	**0.75**

Appendix 5: Usability expert features comparison

Set of questions here were gathered by a usability expert at the university after demo-courses for the systems were made. These questions were given answers on a scale of 0-2. 0 means that given functionality could not be satisfied, 1 would mean that the functionality could be satisfied somewhat, while 2 points were given for functionalities satisfied fully. Some of the tables consider functionalities while others focus on different requirements, this is noted in the first cell on the top-left of each table.

Testing Capabilities

Most of the systems provide all the basic functionalities listed here. This is because all the tools support bash scripting and the usage of external tools. The most important question in each is how easily it can be done. VPL did not function well with complex programs which can be seen in the overall score. GitHub Classroom works well for testing of complex programs thanks to its underlaying infrastructure but could still not provide checkpoints due to the GitHub Action in use within Classroom.

Functionality	***A+***	***CodeGrade***	***GitHub Classroom***	***VPL***	***Viope***
Testing submissions with Valgrind	2	2	2	1	0
Testing submissions through I/O testing	2	2	2	2	2
Testing submissions through unit testing	2	2	1	1	0
Automated testing of CLI programs	2	2	2	2	2
Automated testing of GUI programs	2	1	2	0	0
Automated testing of browser-based programs	2	2	2	1	0
Automated testing of Android programs	2	0	2	0	0
Automated testing of submissions consisting of multiple files	2	2	2	2	2
Keyword detection in automated testing	2	2	2	2	2
Measurement of execution time in automated testing	2	2	2	2	0
Detection of subroutines in automated testing	1	1	1	1	0
Partial scoring of a submission in automated testing	2	2	2	2	0
Support for checkpoints in automated testing	2	2	1	1	0
Total	*25*	*22*	*23*	*17*	*8*
Average	*1.92*	*1.69*	*1.77*	*1.31*	*0.62*

Submission features

In the following table, many of the basic functionalities related to submissions are considered. Some of the functionalities have been voided from the system, and they should instead be provided by Moodle, or another tool.

Functionality	*A+*	*CodeGrade*	*GitHub Classroom*	*VPL*	*Viope*
Setting deadlines of assignments for day, hour, and minute	2	2	2	2	2
A possibility for students to see the model solution after a deadline	0	0	0	0	2
Manual override for points given	2	2	0	2	1
Limit for maximum number of submissions by a student	2	2	0	1	0
Support for timer between a student's submissions	0	2	0	0	0
Support for group submissions	2	2	2	2	0
Support for randomization through a pool of assignments	0	0	0	0	0
Total	*8*	*10*	*4*	*7*	*5*
Average	*1.14*	*1.43*	*0.57*	*1*	*0.71*

Student feedback

This table considers the richness of feedback options given by the systems. The feedback provided by the current system mostly consists of compiler errors, which rarely satisfies the needs of students. Linters are tools that provide feedback on the quality of the system, providing notes on aspects such as variable and file usage. The tools can use linters – some tools such as CodeGrade provide them out-of-the-box, while others will need further setup. ASPA can be integrated with the systems if a command-line version can be provided. The submission return options have a variety of stances by system providers. Especially in the CS1 course, it can be seen as beneficial, to let students create submissions through a text field provided. File insert & Git submission give students more power to work in an environment they choose, while making the submission process harder.

Question	***A+***	***CodeGrade***	***GitHub Classroom***	***VPL***	***Viope***
Support of linters	2	2	1	1	0
Support for integration of ASPA	2	2	2	2	0
Submission return through an IDE	0	0	1	0	2
Submission through a file insert	2	2	0	2	0
Submission through a version control	2	2	2	0	0
Support for rich feedback	2	2	1	0	0
Teacher feedback for the submission	0	2	2	2	0
Teacher feedback for single rows of a submission	0	2	2	0	0
Full automation of assessment process	2	2	2	2	2
Total	*12*	*16*	*16*	*10*	*4*
Average	*1.33*	*1.78*	*1.78*	*1.11*	*0.44*

Additional features

In the next table, some additional features have been considered. Namely, the exporting options and measurement of student activity within the system. Export functionality of student programs. For A+, GitHub Classroom, and VPL, export needs to be done one task at a time. CodeGrade supports mass export of entire courses' submissions.

Functionality	***A+***	***CodeGrade***	***GitHub Classroom***	***VPL***	***Viope***
Track time used by students in the system	0	0	0	0	2
See the number of attempted submissions of students	2	2	0	2	0
Statistics of student submissions & assessment	2	2	2	2	2
Export all student submissions	2	2	2	2	2
Usage of the system for functions other than code assessment	2	0	0	0	2
Total	*8*	*6*	*4*	*6*	*8*
Average	*1.6*	*1.2*	*0.8*	*1.2*	*1.6*

Plagiarism detection

In this table, the tools are considered for their aptitude in plagiarism detection. All the systems had some plagiarism detection systems built-in, which is covered here. Third-party systems would not cover this since the systems would have to deal with a corpus of student submissions. Tools used with the systems would be mostly related to the assessment process of a single submission.

Functionality	***A+***	***CodeGrade***	***GitHub Classroom***	***VPL***	***Viope***
Support for plagiarism detection	2	2	0	2	2
Percentage of similarity between assignments	2	2	0	2	0
Similarity comparison of submissions in a single course	2	2	0	2	2
Similarity comparison of submissions in a course to past instances of the course	0	2	0	2	0
Similarity comparison of submissions in a course to other sources of answers? (i.e Google, lecture materials)	0	2	0	2	0
Total	*6*	*10*	*0*	*10*	*4*
Average	*1.2*	*2*	*0*	*2*	*0.8*

Implementation and usage

The requirements for implementation and usage of the system are covered here. Licensing and usage fees for cannot be covered here for other than noting that CodeGrade and GitHub Classroom, and Viope have licensing fees, while other systems only have usage fees. 24/7 availability was derived from what the licenses would offer, while the systems' performance for large traffic was derived from other use cases of the system. The five questions starting with the word "Need" had a flipped scoring, 0 meaning a larger a need, and 2 meaning a no particular need.

Requirement	***A+***	***CodeGrade***	***GitHub Classroom***	***VPL***	***Viope***
Licensing fees for running the system	0	2	2	0	2
Need to train staff at SWE for the system	0	0	0	1	0
Need to train staff at IS for the system	2	0	0	0	0
Need for maintenance of the system	0	1	1	2	1
Need for maintenance of the courses	2	2	1	2	0
Need for hosting the system at LUT University	2	0	0	0	2
24/7 availability of the service	0	2	2	0	2
Data security policy	2	2	2	2	2
Functionality under high amounts of traffic	2	2	2	2	2
Total	*10*	*11*	*10*	*9*	*11*
Average	*1.11*	*1.22*	*1.11*	*1*	*1.22*

Moodle integration

The Moodle integration of the systems is considered here. A+ has a Moodle plugin that is used at Tampere University, but the plugin is not officially supported. GitHub classroom has an LMS integration, but its usage is very limited. Other tools work well with Moodle – VPL is integrated within Moodle, so it shines here. Information flow within the tools was provided by the Moodle, GitHub Classroom had some troubles here as well. Finally, the export process of assessments to Moodle is an important consideration. A+ still has some work to be done on that matter, while GitHub Classroom supported it only manually.

Functionality	***A+***	***CodeGrade***	***GitHub Classroom***	***VPL***	***Viope***
Support for Moodle integration	1	2	1	2	2
Access to student information purely through Moodle	1	2	1	2	2
Automated export of grades to Moodle	0	2	0	2	2
Total	*2*	*6*	*2*	*6*	*6*
Average	*0.67*	*2*	*0.67*	*2*	*2*

Course administration

Here, the systems are considered for their functionalities in course administration. The answers to these questions were taken from documentation and use-cases of the tools. Demonstrations of the tools gave the research team an insight whether they would fit for larger courses, but no actual evidence could be given for many of the topics

Funtionality	***A+***	***CodeGrade***	***GitHub Classroom***	***VPL***	***Viope***
Management of large numbers of assignments	2	2	0	2	2
Importing of I/O tests easily into the system	1	1	0	2	2
Tools for managing the assignments of the system	1	1	0	2	2
Assignment of assessments to TAs	0	2	0	0	0
Different user roles within system	1	2	1	1	1
Total	*5*	*8*	*1*	*7*	*7*
Average	*1*	*1.6*	*0.2*	*1.4*	*1.4*