A Computer-Based Examination System for XLogoOnline

Master Thesis

Author(s):
Eschbach, David

Publication date:
2019

Permanent link:
https://doi.org/10.3929/ethz-b-000372375

Rights / license:
In Copyright - Non-Commercial Use Permitted
A Computer-Based Examination System for XLogoOnline

David Eschbach

Master Thesis
01.09.2019

Supervising Professor:
Prof. Dr. Juraj Hromkovič

Supervisor:
Jacqueline Staub

ETH zürich
Abstract

Recently the efforts to provide sound computer science education as part of general education have increased. The focus is moved away from information and communications technology (ICT) skills and algorithmic thinking, the way to think in terms of algorithms, is emphasized instead. One way to teach algorithmic thinking is programming education, which is getting more common as part of general education in Switzerland. People start to realize that programming concepts can already be taught in primary schools, if appropriate tools, as for example the programming language LOGO, are used.

Teaching programming in primary schools is challenging for teachers. Among the reasons for this are that only few primary school teachers have programming experience and that pupils need extensive support while learning how to program. In order to master these challenges, teachers need sound, self-contained teaching materials and appropriate tools. One such tool is the web-based LOGO programming environment XLogoOnline which was introduced a few years ago [49].

In this project we extend XLogoOnline by presenting a computer-based examination system. Teachers can use our system’s teacher platform to create exams, which are solved by pupils in their used programming environment XLogoOnline. Afterwards, our system collects all pupils’ solutions and supports teachers in their correction work. For this purpose, our system’s teacher platform contains a semi-automated assessment system, that supports teachers without taking away their responsibility.

We tested our examination system by carrying out a programming contest in a primary school in Switzerland. The pupils were able to solve the contest’s tasks in XLogoOnline. Our system automatically collected the pupils’ results in the teacher platform and we were able to correct all tasks in the contest using our semi-automated assessment system. This way we demonstrated that our system is ready to be used in its target environment.
Our system simplifies the process of carrying out a programming exam in the classroom. Previously, pupils had to solve LOGO exams with pen and paper. Using our system, they can solve the exam in the familiar programming environment XLogoOnline. This reduces the load imposed on pupils. Further, our system speeds up the correction work. Previously, correcting a LOGO exam was challenging and time consuming because teachers had to interpret LOGO programs written by pupils. Our system enables reexecuting a pupil’s LOGO program to obtain the image it draws. Furthermore, teachers can rely on our automatic correction, which compares a pupil’s LOGO program with a corresponding master solution.
Acknowledgment

I first met Jacqueline Staub in September 2016, when I was her assistant in a LOGO workshop offered in the context of the Digital Festival 2016 in Zürich. On this occasion she told me that she was working on her Master thesis with the goal to develop a web-based LOGO programming environment: XLogoOnline. I did not see her again for more than two years but the idea to do something similar for my own Master thesis never left me. When I reached out to Professor Juraj Hromkovič’s research group to ask if a Master thesis was available, I was quite surprised when Jacqueline responded to me and offered a project to integrate an examination system into XLogoOnline. I did not need to think long before I knew that this was the right Master thesis for me. I am very grateful that I was able to work on this challenging, inspiring, and exciting project.

I would like to thank my direct supervisor Jacqueline Staub, who invested much time to collaboratively discuss my ideas, give me feedback, and help me when I did not know how to proceed. Her joy about XLogoOnline in general and my project in particular was an inspiration.

Further, I would like to thank Professor Juraj Hromkovič and his research group for welcoming me as part of the group and offering me a spot in one of the group’s offices. It is not self-evident to be able to write a Master thesis in such a comfortable and caring work environment.

Next, I would like to thank Ulla Aeschbacher and Adrian Meier for giving me feedback and helping me to proofread my thesis. Their feedback on my examples, figures, and explanations was an appreciated help to improve my thesis.

Additionally, I would like to thank all primary school teachers who filled out my questionnaire. Their feedback helped a lot in order to improve the examination system.

A very big "thank you" belongs to my wife, Naël Eschbach, who supported me throughout the last six months. She helped to proofread my thesis, encouraged me when I did not know how
to proceed, and was always there for me when I needed it.

I am very grateful that I could write my thesis in such good circumstances. I would like to thank everyone who supported me during the past six months.
Contents

1. Introduction 1
   1.1. Our Project’s Context 1
   1.2. Problem Statement 3
       1.2.1. Difficulty for Pupils 3
       1.2.2. Difficulties for Teachers 3
   1.3. Goals of the Thesis 4
   1.4. Organization of this Document 4
   1.5. Conventions 5

2. Background and Related Work 7
   2.1. Computer Science in Swiss Education 7
   2.2. Contribution of Computer Science to General Education 8
   2.3. XLogoOnline 10
   2.4. Assessing Computer Science Skills 11
       2.4.1. Bloom’s Taxonomy 11
       2.4.2. SOLO Taxonomy 14
       2.4.3. Taxonomies for Lower Education 15
   2.5. Assessment Systems 17
   2.6. XLogoOnline’s Judge 18

3. Requirements 21
   3.1. Use Case 21
   3.2. Requirements 22
       3.2.1. Questionnaire 22
       3.2.2. Functionality 24
       3.2.3. Non-Functional Requirements 25
## Contents

### 4. Design

4.1. Model of Distributed System ........................................... 27
   4.1.1. Server ......................................................... 29
   4.1.2. Teacher-Client ............................................... 30
   4.1.3. Pupil-Client .................................................. 31

4.2. Design Choices ......................................................... 31
   4.2.1. Teacher in Control ............................................ 31
   4.2.2. Authorisation of Teacher ..................................... 32
   4.2.3. Conflicts Resolving ........................................... 32

### 5. Tools and Technical Background

5.1. Communication .......................................................... 35
   5.1.1. HTTP ............................................................ 35

5.2. Server ................................................................. 36
   5.2.1. Bottle .......................................................... 36
   5.2.2. JSON Web Tokens ............................................. 37
   5.2.3. Pillow .......................................................... 38

5.3. Teacher-Client .......................................................... 38
   5.3.1. TypeScript ...................................................... 39
   5.3.2. Angular .......................................................... 40

5.4. Pupil-Client ............................................................. 42
   5.4.1. XLogoOnline .................................................... 43

### 6. Implementation

6.1. Architecture ............................................................ 45
   6.1.1. Server .......................................................... 45
   6.1.2. Teacher-Client .................................................. 47
   6.1.3. XLogoOnline ................................................... 48
   6.1.4. Communication Between Nodes and Error Handling ......... 49

6.2. Managing Courses ....................................................... 54
   6.2.1. Course Enrollment .............................................. 54

6.3. Running an Exam ........................................................ 55
   6.3.1. Exam Preparations .............................................. 56
   6.3.2. Exam Solving .................................................... 56
   6.3.3. Summary .......................................................... 60

6.4. Correction of LOGO Exams ............................................. 60
   6.4.1. Manual Correction .............................................. 60
   6.4.2. Automatic Correction ......................................... 62

### 7. Evaluation

7.1. Setup ................................................................. 71

7.2. Functional Evaluation ................................................ 73
   7.2.1. Exam Preparations .............................................. 73
   7.2.2. Carry Out Exam ................................................ 73
   7.2.3. Correction ...................................................... 74

7.3. Research Potential of Backup Mechanism .......................... 81
   7.3.1. CS-Score ...................................................... 82
7.3.2. Planning Phase Duration ........................................... 84
7.3.3. History Length ....................................................... 85

8. Conclusion ................................................................. 89
  8.1. Conclusion ............................................................ 89
  8.2. Limitations and Implications ....................................... 90
    8.2.1. Separation of History and Editor .......................... 91
    8.2.2. Offline Mode .................................................. 92
    8.2.3. Multiple Colors .............................................. 92
    8.2.4. White Strokes ................................................ 93
  8.3. Future Work ......................................................... 94
    8.3.1. Improvements of Current System .......................... 94
    8.3.2. Extensions to Our System ................................... 95

A. Questionnaire .......................................................... I

B. Contests ................................................................. XI
   B.1. Contest Used to Test Examination System .................... XI
   B.2. Contests Used in Previous LOGO Courses ..................... XVII

C. XLogoOnline Cheat Sheet ............................................ XXIII

Bibliography .............................................................. XXVII
Introduction

Computers are omnipresent in our world. Almost every business sector is affected by the trend towards more automation. As a result, software gets an ever more important role in our world. This makes our life more comfortable but also introduces new challenges. In order to be prepared for those challenges, it is important that school graduates have at least a basic understanding of how software works. A very suitable way to achieve this is by teaching programming. Basic programming skills are beneficial for everyone. Our work contributes to the process of developing good programming classes as part of general education and thereby preparing pupils for the jobs of the future. We present an examination system that is integrated into the web-based LOGO programming environment XLogoOnline\(^1\).

In this chapter we first derive our problem statement by contextualizing our project. We move on to define our problem statement and goals for this project. We conclude the chapter by providing an overview of this document and introducing conventions used throughout this work.

1.1. Our Project’s Context

In this section we derive our problem statement for this project. We first outline the project’s context of computer science education. Particularly, we discuss why computer science education is advantageous and important already on primary school level. We further shortly introduce XLogoOnline, a web-based programming environment, into which our system is integrated. We highlight challenges that primary school teachers face when teaching programming and show how our project helps to overcome these challenges.

\(^1\)We assume that readers are familiar with the programming language LOGO. All the same, in appendix C we provide a cheatsheet that introduces the most important LOGO commands.
1. Introduction

**Relevance of Computer Science**  The importance of computers has rapidly increased in recent years. Today most people are confronted with software in some way at their workplace. Automation is a trend that has started to reshape most business sectors. Also in our everyday life we are increasingly dependent on software, since a substantial part of our social life takes place on the Internet. We can not expect that the trend of increasing automation is going to stop in the near future. It is likely that our dependency on computerized systems will continue to increase. Therefore, we should ensure that we, and in particular our children, are prepared for this.

**Computer Science Education in Switzerland**  On college level Switzerland has a long tradition of both research and education in computer science. Especially ETH has been leading the way not only in computer technology but also in computer science didactics. For example the programming language Pascal was developed by ETH Professor Niklaus Wirth with a clear emphasis on being a didactically good choice to teach programming [58]. But on the level of primary and secondary school education, Switzerland has not been strong when it comes to computer science. If computer science was taught at all, too often there was a focus on ICT skills like typing or mastering Microsoft Office. ICT skills, as important they might be, are not computer science.

**Recent Improvements in Computer Science Education**  In recent years, teachers and politicians in Switzerland started to care more about providing sound computer science education on all school levels. A few years ago the Deutschschweizer Erziehungsdirektoren-Konferenz (D-EDK) has accepted a new curriculum draft for Swiss school education called Lehrplan 21. This curriculum is currently about to be implemented in practice. In Lehrplan 21 computer science finally receives a more important role. Lehrplan 21 contains appropriate learning objectives for computer science including programming skills already for primary school level [11].

**Benefits of Programming Education**  Programming is not only a useful skill in order to better understand how software works but also beneficial in an interdisciplinary context. A part of this interdisciplinary nature is due to the increasing automation in almost all business sectors. But even more importantly, teaching programming enables pupils to learn to structurally solve a problem. Programming can be defined as communicating with a machine in a language the machine understands. Communicating to a machine requires absolute precision. Therefore, programming in its core is about cleanly formalizing a problem and then finding a precise solution to it. Being able to approach problems in this fashion is helpful in all situations life has in store for us.

**Challenges of Programming in Primary Schools**  Computer scientists generally welcome the increased focus on programming skills in primary and secondary school education. But it is obvious that new challenges arise, especially on the level of primary schools. On primary school level teachers are all-rounders and not specialists. Thus, only few primary school
teachers are experienced programmers. They need help in order to teach programming skills to primary school pupils, particularly in order to develop a good teaching curriculum.

**Efforts of ABZ** The Ausbildungs- und Beratungszentrum für Informatikunterricht (ABZ) strives to help primary and secondary school teachers to provide good computer science education [2]. ABZ has developed computer science learning materials for all levels of school education. ABZ has further conducted pilot programming courses in LOGO in primary schools all over Switzerland since 2005. These pilot courses are carried out by computer science students and ABZ employees, but teachers take part to be prepared for teaching the content themselves in the future. Additionally, ABZ provides education and further training to teachers that are willing to teach programming in their classes.

**XLogoOnline** One of the more recent tools developed at ABZ is XLogoOnline ([49] [20]). XLogoOnline is a web-based integrated development environment (IDE) for LOGO programming and has been introduced specifically to be used in LOGO programming courses organized by ABZ. It has already been used in many programming courses and was shown to be very suitable for teaching programming [34].

### 1.2. Problem Statement

In this section we present our problem statement. Our problem statement captures three challenges occurring when working with XLogoOnline in class. In order to explain these challenges, we first focus on the pupil and then emphasize the problems that are faced by teachers. To conclude the section, we discuss how our project specifically tackles these challenges.

#### 1.2.1. Difficulty for Pupils

**Exam in Known Environment** XLogoOnline proved to be very suitable for teaching LOGO [34]. But so far teachers working with XLogoOnline lacked possibilities to assess the pupils’ learning process. Many teachers expressed the desire to carry out programming exams directly in XLogoOnline. This would enable pupils to take exams in the environment they are familiar with. Currently, teachers have to switch to a pen and paper setup in order to carry out an exam. This adds an overhead for pupils and might affect their performance.

#### 1.2.2. Difficulties for Teachers

**Designing Exams** Since many primary school teachers have few programming experience, designing suitable questions for a programming exam is challenging for them. Working with the self-contained learning materials of ABZ is much easier than coming up with good tasks to assess the pupils’ programming skills. Thus, teachers need a source for good examples of programming exams.
1. Introduction

Correcting Exams  Correcting programming exams is a time consuming task even for experienced programmers. For teachers, who are normally not exercised programmers, correcting a programming exam is even more difficult. When correcting a programming exam, a teacher has to interpret programs that potentially solve a task in a completely different way than the master solution. This can be challenging even for experienced programmers. Therefore, teachers would benefit from a tool that automatically carries out some parts of the correction and assists them during the rest of the correction work. There is a wide range of possible exam tasks on LOGO programming which could be checked automatically. If the whole examination environment is computer-based, there are no technical obstacles preventing us from using automation in the correction process.

Our Project As Solution

Our project integrates an examination system into XLogoOnline. Additionally, our system supports teachers in designing suitable exams and helps them to speed up an exam’s correction. Thus, our project contributes to solving the problems described in this section. We present a prototype that helps teachers to design exams. The teacher then can distribute a LOGO exam to pupils, who can solve the exam directly in XLogoOnline, which they are familiar with. After the exam, our system collects all pupils’ answers and assists the teacher throughout the correction work. Our system can correct an exam automatically, but the teacher has the possibility to overwrite all corrections manually.

1.3. Goals of the Thesis

The main objective of this thesis consists of designing, implementing, testing, and integrating an examination system that can be used with the browser-based programming environment XLogoOnline. It should allow teachers to design and distribute exams for their pupils, collect and automatically verify the submitted results whenever possible, link the results back to individual students, and summarize all information conveniently for grading. The programming platform is to be adapted and a teacher’s platform has to be implemented. The outcome of this thesis is a well-documented, stable, and reliable prototype, providing the functional elements to be used in schools. Furthermore, XLogoOnline and the new teacher’s platform shall be compliant to the original design goals of XLogoOnline [49]. In particular, XLogoOnline must still be universally usable among different browsers and its support for offline working must be preserved. Additionally, the teachers platform must provide a user-friendly graphical user interface.

1.4. Organization of this Document

Next, we provide a short overview of this document.
Chapter 2  In chapter 2 we provide important background knowledge for this project. Most importantly, this includes a more thorough discussion of our project’s context and a presentation of related work. It does not contain technical background information, which can be found in chapter 5.

Chapter 3  In chapter 3 we derive a set of requirements that our system has to satisfy. To replenish our own experiences, we designed a questionnaire and asked primary school teachers to fill it out. The results of this questionnaire are presented in chapter 3.

Chapter 4  In chapter 4 we present our system’s model, which is derived from the requirements worked out in chapter 3. Further, we highlight important design decisions taken in order to satisfy the requirements imposed on our system.

Chapter 5  In chapter 5 we provide background knowledge for the technologies used to implement our system. This includes an introduction of XLogoOnline from a technical perspective.

Chapter 6  In chapter 6 we describe how we implemented our system. We start by giving an overview of the system’s architecture. Then we study selected examples in detail in order to provide insights into how our system works.

Chapter 7  In chapter 7 we evaluate our system. For this purpose, we carried out a LOGO programming contest in a primary school. We describe our experiences related to this contest and highlight improvements we made as a result of this contest. Further, we show how our system could be used to examine interesting research topics like programming related differences between girls and boys.

Chapter 8  In chapter 8 we reflect on our project and present our conclusion. We further present limitations of our system and provide ideas for future work.

1.5. Conventions

In this section we introduce conventions used in this document. The purpose of these conventions is to provide better readability and avoid being overly redundant. The conventions adhered in this document are:

- Throughout this document we refer to a teacher or other persons in the male form. The motivation behind this is to simplify the language and ensure better readability. We do not mean to discriminate anybody.

- We discuss several examples where a teacher and a pupil have to interact. In those examples we refer to the teacher as Tom and to his pupil as Paola.
1. Introduction

- We use italic font and numbers when we refer to particular instances of a certain item. For example exam1 is an instance of an exam and task1 is an instance of a task.
- Elements written in this font, refer to source code elements like variables or parameters.
Background and Related Work

In this chapter we will provide background knowledge which influenced our project’s design decisions. We start with a discussion of didactical considerations on how to teach computer science as part of general education. Here we both refer to classic literature and to more recent research topics. We further provide background information about XLogoOnline’s design and give an overview of ABZ’s work. Additionally, we discuss research results concerning didactical aspects of computer science assessment and consider examination systems presented in literature.

2.1. Computer Science in Swiss Education

In this section we study how computer science is taught in swiss schools. For this purpose, we first highlight that for a long time computer science education focused too much on using common software. We then show how ABZ tries to improve computer science education.

Heavy Focus on End-User Software  Since many years Swiss high-schools offer a subject called computer science. This subject was not part of the mandatory curriculum for a long time and could only be chosen on a voluntary basis in many schools. Unfortunately, this subject apparently did not convey the core principles of computer science. This can be deduced from the results of a study that the Hasler Stiftung funded in 2008 [52]. The goal of the study was to examine the image of the school subject computer science in Switzerland. The study’s executive summary states that a vast majority of high school students and teachers regard computer science as "usage of common software". Especially alarming is that this result is extracted even clearer from the teacher survey than from the student survey.
2. Background and Related Work

Efforts to Improve Computer Science Education ABZ has realized the mismatch between computer science’s core principles and the reputation of its high-school subject early on. In order to improve computer science education, ABZ started to develop more appropriate teaching materials for all levels of general education. These teaching materials focus on core principles of computer science like algorithms, abstraction, and automation. ABZ’s work is driven by the belief that computer science can make a significant contribution to general education, as we will discuss in section 2.2. This implies that teaching computer science should start as early as possible. Consequently, ABZ soon started to carry out programming courses in primary schools based on their teaching materials. Furthermore, ABZ closely cooperates with pedagogical colleges to provide the best possible computer science instruction for future primary school teachers [9]. This is particularly important because in Switzerland primary school teachers are all-rounders that have to teach many distinct subjects.

2.2. Contribution of Computer Science to General Education

In this section we discuss how computer science can contribute to general education. For this purpose, we highlight computer science’s interdisciplinary nature and consider computational thinking. We explain why computational thinking is a useful ability for everyone. Further, we will discuss a spiral curriculum for computer science education, which builds upon computational thinking.

Computer Science’s Interdisciplinary Nature Already several years before the Hasler study, ETH Professor and ABZ founder Juraj Hromkovič has warned that too many people relate computer science to the “ability to drive a computer” [29]. He concludes that computer scientists have to rethink how computer science contributes to general education. Doing this leads him to the interdisciplinary nature of computer science that combines aspects of metascience, natural science, and engineering. He identifies this interdisciplinary nature as one of computer science’s biggest strengths and uses it to justify the importance of computer science to general education. He further manifests algorithms as a core concept of computer science. Consequently, he suggests that computer science in general education should focus on teaching pupils to think in terms of algorithms.

Computational Thinking Jeanette M. Wing refers to "thinking in terms of algorithms" as "computational thinking" and highlights its importance to general education [57]. She describes computational thinking as the ability to reformulate a seemingly difficult problem into another problem to which a solution approach is known. She emphasizes that computational thinking is a fundamental skill not only for computer scientists but for everyone. Consequently, she states that computational thinking should be added to every child’s analytical ability. Apart from describing the nature of computational thinking, she also makes clear what computational thinking is not. Among other aspects she warns from putting to much focus on end-user software by clarifying that "computational thinking is about ideas and not about artifacts".
2.2. Contribution of Computer Science to General Education

**LOGO Programming for Computational Thinking**  Giovanni Serafini reaffirms that computational thinking is the main contribution of computer science to general education [47]. He discusses how computational thinking should be conveyed and presents programming education as a good approach. He recommends to work with LOGO because it relies on a small but concise instruction set. Therefore, LOGO imposes the need for correct syntax rather than relying on a drag-and-drop approach as other mini-languages like Scratch. Working with LOGO enables a good balance between simplicity and learning the crucial ability to work precisely in terms of syntax.

**Experiences in Teaching LOGO Programming**  Serafini further shares experiences in teaching LOGO programming [47]. During the described teaching sequences he relies on materials derived from the German textbook *Einführung in die Programmierung mit LOGO* [30]. He emphasizes that teaching materials for primary school should be self-contained, a requirement that is satisfied by *Einführung in die Programmierung mit LOGO*. An important reason why teaching materials should be self-contained is that primary teachers are often not much more comfortable in programming than their pupils. Additionally, he suggests that computer science education should be constructed in a spiral curriculum.

**Spiral Curriculum**  Hromkovič et al. present their version of a spiral curriculum for computer science education ([32] [33]). It exploits computer science’s interdisciplinary nature and provides bridges to other subjects. This enables smooth embedding into general education. The curriculum is built upon three pillars of computational thinking:

1. A formal language as a means to express algorithms
2. Abstraction and automation as strategies for problem solving
3. The limits of practical computability implying the need to improve algorithms

**The Turtle**  One of the spiral curriculum’s central threads is the concept of a turtle as programmable machine. The turtle has a pen attached to its belly and can be moved over a canvas in order to draw images. The turtle can move forward and backward and can turn to the left and to the right. This idea was introduced by Seymour Papert, who was also involved in the design of LOGO [42]. Using a turtle as programmable machine on which programs are executed defines programming as "talking with the turtle" and the programming language as "the turtle’s language". Upon this abstraction further powerful didactic metaphors can be built. For example "teaching the turtle a new word" can be viewed as analogy for writing a new program. The turtle also acts as bridge to higher grades, where the spiral curriculum moves away from teaching LOGO. Instead of LOGO, in higher grades the spiral curriculum relies on the more powerful programming language python [31]. However, the spiral curriculum does not use the common version of Python but a special dialect. This dialect is sometimes referred to as TigerJython, but actually TigerJython is the name of the programming IDE that supports the dialect\(^1\) [38]. The dialect most importantly contains a loop construct that can be used without introducing the concept of variables and support turtle graphics.

\(^1\)The dialect itself has no name.
2. Background and Related Work

![Figure 2.1: This figure presents an example on how XLogoOnline supports pupils during the debugging process. In figure 2.1a we show a screenshot of a LOGO command sequence that contains an error. Already before executing the command sequence, the pupil’s attention could be drawn to the error because of the colors (**fd100** is blue because it is not recognized as LOGO command). Known LOGO commands are colored differently in XLogoOnline’s input line. If the pupil still executes the command sequence, he will receive the error message displayed in figure 2.1b. This enables the pupil to quickly spot the erroneous command and fix the error.](image)

2.3. XLogoOnline

Recent work of ABZ includes the single-page and browser-based programming environment XLogoOnline that was first presented in the Master thesis of Jacqueline Staub [49]. In this section we introduce XLogoOnline from a design and context perspective. We do not consider XLogoOnline’s implementation, as this will be discussed in section 5.4.1.

**XLogo** XLogoOnline works with the LOGO dialect XLogo, which was proposed by French mathematics professor Loïc le Coq [19]. XLogo was then adapted during a Bachelor thesis at ETH resulting in the version used in XLogoOnline [54].

**Reducing Cognitive Load** XLogoOnline is specifically designed for LOGO programming in primary schools. This implies that the target users are children that are approximately ten years old. Therefore, XLogoOnline aims at reducing the cognitive load on pupils while learning how to program. This enables pupils to fully focus on concepts as for example repetition, modular design, and parametrization [34]. In order to reduce the cognitive load on pupils, much effort was put into developing an intuitive and comprehensive user interface (UI). This UI ensures that all relevant information is visible at the pupil’s first glance.

**Debugging** A big part of programming has to be reserved to finding and correcting errors in the code. For beginners this process is particularly cumbersome. The reason for this is that beginners do not know yet where errors most likely occur. In order to help pupils, XLogoOnline supports them in the debugging process by providing smart hints and pinpointing errors. In figure 2.1a we present an example.
2.4. Assessing Computer Science Skills

**ABZ Study**  ABZ carried out a study during which pupils and teachers were interrogated on their experience with XLogoOnline [34]. The results are encouraging since most pupils understood the usage and meaning of XLogoOnline’s UI components correctly. In particular, the study showed that pupils could easily transfer from the old system XLogo4Schools to the new XLogoOnline. Furthermore, ABZ has received much positive feedback on XLogoOnline. Especially the syntax checker, error pinpointing, layout, clarity, and comprehensibility were approved.

2.4. Assessing Computer Science Skills

In this section we consider assessment in the context of computer science education. Assessment is an important topic in our project because we consider exam generation. We will discuss Bloom’s taxonomy and the "Structure of the Observed Learning Outcome" (SOLO) taxonomy and emphasize how they can be used in the context of computer science. To conclude the section, we particularly focus on the context of primary schools.

**Summative vs. Formative Assessment**  Assessment has two main goals: Firstly, it is about comparing the achievements of pupils to the set of teaching objectives defined before the learning unit. This is also called summative assessment. Secondly, assessment serves as feedback mechanism helping to improve the teaching itself. This is often referred to as formative assessment.

**Taxonomies to Design Assessment Instruments**  Assessing programming skills is not a trivial task. A good starting point for studying the topic is the literature review by Maria Kallia [37]. This literature review is particularly useful because it contains many pointers to interesting contributions. The literature review’s main focus is to review existing assessment approaches, tools, and instruments. Doing this Kallia emphasizes the process of developing assessment instruments. For this purpose, she names Bloom’s taxonomy [24] and the SOLO taxonomy [23] the two most important tools. Both serve as means to categorize questions according to their difficulty, but there is a subtle difference in the approach. Bloom’s taxonomy considers the analytical skills required to answer a question while the SOLO taxonomy focuses on the analytical skills applied in the process of deriving a response.

In the following we look at Bloom’s taxonomy and the SOLO taxonomy in order to discuss how they can be used for assessment. During this process we focus on the context of computer science.

2.4.1. Bloom’s Taxonomy

In this subsection we study contributions that consider Bloom’s taxonomy in the context of computer science. Our main focus is to show that using Bloom’s taxonomy can be used to improve computer science assessment.
2. Background and Related Work

### Original Bloom’s Taxonomy
In 1956 Bloom et al. presented a taxonomy to assign the cognitive aspects of learning to six distinct levels [24]. These levels are in increasing order: Knowledge, comprehension, application, analysis, synthesis, and evaluation. Every level has to be mastered in order to reach the next higher level. Therefore, for example synthesis is a higher cognitive aspect of learning than application. In figure 2.2 we present a visualization of Bloom’s taxonomy.

### Norm- vs. Criterion-Referenced Assessment
Norm-referencing assessment is common in computer science education because its main focus is to compare multiple students’ performances on the same task set. This approach is efficient for teachers because they do not have to take into account different students. However, Lister et al. state that using the norm-referencing approach in computer science tends to handicap the learning process of both strong and weak students, while only being appropriate for the average student [39]. As alternative they propose the criterion-referenced approach, which focuses on how well students perform on a particular task. The criterion-referenced approach requests teachers to communicate explicit and clear criteria for each grade to their students. For the design of these criteria, Lister et al. recommend to build upon Bloom’s taxonomy.

### Bloom to Smooth Performance Curves
Another problem is that many exam questions are either low level or high level, but only few questions have a difficulty in between. This often results in a bimodal distribution of frequency versus scores. According to Scott, assessing computer science skills based on Bloom’s taxonomy would help to get smoother performance curves.
2.4. Assessing Computer Science Skills

curves [45]. The overall message is that assessment should address all levels of Bloom’s taxonomy. This was already stated by Bloom et al., because in their context too much emphasis was put on the levels "knowledge" and "comprehension". In computer science education of today we face the opposite challenge, as assessment too often only targets the higher levels of Bloom’s taxonomy too often [39].

Revised Bloom’s Taxonomy

More recent approaches rarely refer to the original Bloom’s taxonomy but build upon the revised Bloom’s taxonomy as proposed by Anderson et al. [22]. This revision suggests that analytical skills have to be measured in two dimensions: the knowledge dimension and the cognitive dimension. The knowledge dimension distinguishes the increasingly challenging levels factual, conceptual, procedural, and meta-cognitive. The cognitive dimension is a slightly adapted version of Bloom’s original taxonomy and consists of remember, understand, apply, analyze, evaluate, and create. The most notable adaption of the cognitive dimension is that, in contrast to Bloom’s original taxonomy, the two highest levels are exchanged. In figure 2.2 we oppose Bloom’s original taxonomy and the cognitive dimension of the revised Bloom’s taxonomy. For the rest of this work we only consider the cognitive dimension of the revised Bloom’s taxonomy, because the knowledge dimension is usually ignored in literature concerning computer science.

Level-Assignment of Tasks Is Hard

Johnson and Fuller carried out a study considering all assessments in the first year of computer science studies at the university of Kent [36]. A team of five members of their department assigned the assessments to the levels of the revised Bloom’s taxonomy. This revealed that assigning computer science tasks to the appropriate level is hard. This is mostly due to the need of in-depth knowledge about the corresponding course for an appropriate decision. For example if a task requests students to write a function that sorts an array, the cognitive level of this task depends on whether sorting algorithms were covered in the course. If sorting algorithms were covered, this task could be assigned to "Remember" otherwise the level could be as high as "Create". The need for in-depth knowledge about the course was already stated by Bloom et al. in the original contribution [24].

Higher Application

In the context of computer science, many experts hold the opinion that most emphasis should be put on the application level. Johnson et al. accommodate this by suggesting an additional level called "higher application" as highest level for the context of computer science. They define this level as: "the application informed by a critical approach to the subject, but where the criticism is not, as such, the focus of the work" [36]. However, the disagreement on choosing an appropriate highest level could also arise from a missing common understanding of the Bloom levels in computer science.

Common Understanding of Levels

Thompson et al. point out that computer scientists lack a common understanding of how the levels of Bloom’s taxonomy should be mapped to programming tasks [50]. They attempt to bring more clarity to the matter by discussing a set of example tasks. For each of these tasks they suggest a categorization and elaborate on this
choice. They admit that during the categorization process there had been disagreements, but these were overcome by discussing assignments with all people involved.

**Consensus Without Collaboration** Thompson et al. silently accept that agreement is only reached after collaboratively discussing classification [50]. This aspect is criticized by Gluga et al. [27], but they fail too to provide a solution eliminating this problem from scratch. However, they conclude that teachers aiming to work with the revised Bloom’s taxonomy have to be trained in order to reach consensus. To achieve this, they present a computer science contextualized tutorial on Bloom’s taxonomy with interactive examples. As others before, they further point out that the cognitive level required to solve a particular task depends on the student that solves it due to different backgrounds.

While Bloom’s taxonomy is especially useful to design questions for an exam, we also have to take a closer look at the answers to those questions. For this purpose, we consider the SOLO taxonomy in the next section.

### 2.4.2. SOLO Taxonomy

In this subsection we consider the SOLO taxonomy, which categorizes answers to a particular task according to the cognitive skills applied while deriving the answer. We first explain how the SOLO taxonomy works in general and continue by focusing on its usage in the context of computer science.

**SOLO Taxonomy Definition** The SOLO taxonomy presents five levels of solution approaches. Considering these levels it is important to notice that the correctness of an answer does not determine the assignment to a level. The only thing that matters is the way a solution is achieved. Thus, the SOLO taxonomy can be used to estimate the skills of the students independent of their performance in tests. Lister et al. describe the SOLO taxonomy’s levels as follows [40]:

- **prestructural**: "The least sophisticated type of response a student may give."
- **unistructural**: "A response where the student manifests a correct grasp of some but not all aspects of the problem."
- **multistructural**: "A response where the student manifests an understanding of all parts of the problem, but does not manifest an awareness of the relationship between them."
- **relational**: "A response where the student integrates the parts of the problem into a coherent structure, and uses that structure to solve the task."
- **extended abstract**: "The student response goes beyond the immediate problem to be solved, and links the problem to a broader context."

**SOLO Is Applicable to Computer Science** Whalley et al. have run an experiment on this topic by analyzing students answers in selected low level programming courses [56]. They
propose to rely on the revised Bloom’s taxonomy for designing questions and on the SOLO taxonomy for analyzing students’ answers. They found that the students’ performances reflect both the questions’ Bloom levels and the answers’ SOLO levels. For example tasks with low success rates tended to have high levels of Bloom’s taxonomy. Also students with a weak overall performance were less likely to have used higher level solution approaches of the SOLO taxonomy [56]. This is interpreted as indication that Bloom and SOLO are suitable tools to improve the assessment of programming skills.

SOLO Level Assignment  Lister et al. discuss the application of the SOLO taxonomy to code reading exercises [40]. Code reading exercises require a student to read a piece of code and for example explain what it does or find a bug in it. Lister et al. try to determine the levels of the SOLO taxonomy more accurately by comparing a group of computer science experts to a set of novices. All participants had to "think loud" while attempting to solve the tasks, such that their solution approaches could be analyzed. Not surprisingly Lister et al. found that experts tend to apply higher level strategies than novices. But they also show that higher level solution approaches are no guarantee for correctness in program reading questions. Experts tend to oversee details, because they think too much in terms of what the code should do. Therefore, graduates should be able to apply all levels of the SOLO taxonomy. To ensure this, Lister et al. recommend teachers to use a variety of assessment strategies.

2.4.3. Taxonomies for Lower Education

All contributions presented so far in this section have the drawback that they do not match our setting, since they address the context of higher education. The main reason to consider them all the same is that there are only few usable contributions that consider taxonomies in the context of lower education. Furthermore, we believe that many of the findings above are also valid in our context, although we must be careful about the transfer. While there is not much literature on the topic outside the realms of higher education, there are a few noteworthy contributions, which we want to study in this subsection.

Programming Adaptive Testing (PAT)  Chatzopoulou et al. have presented a web-based, adaptive testing system called PAT for high-school level [25]. PAT is intended to be used for assessing students’ programming knowledge and offers a set of multiple-choice questions. These questions are categorized along two dimensions: The first dimension describes a task’s difficulty and the second dimension represents cognitive levels. For these cognitive levels, a subset of the Bloom’s taxonomy’s cognitive levels is used: recall, comprehension, and application. The most interesting feature of PAT is its adaptability. The questions in the assessment are chosen iteratively and upon every wrong answer the system ensures that the next question belongs to a lower cognitive level.

Meerbaum-Salant Taxonomy  Meerbaum-Salant et al. combined the SOLO taxonomy and the revised Bloom’s taxonomy into a new taxonomy, because they consider neither of them expressive enough in the context of teaching programming [41]. In their perception, revised
2. Background and Related Work

Figure 2.3: In this figure we present a visualization of the new taxonomy proposed by Meerbaum-Salant et al. [41]. The SOLO levels unistructural, multistructural, and relational are used as super-categories to mirror the aspect of an increasingly holistic solution approach. Each of these super-categories has the three subcategories understanding, applying, and creating which are taken from the revised Bloom’s taxonomy.

Bloom’s taxonomy lacks means to consider the difficulty of the concepts being learned. For example “understanding” the complex concept of concurrency is harder than “analyzing” a simple "Hello World"-program. The reason for this is that to understand concurrency a holistic view is required, while for analyzing "Hello World" a very small scope is sufficient. The SOLO taxonomy distinguishes between holistic and local views. But the SOLO taxonomy can for example not be used to compare program creation and analysis, which is an important distinction. Therefore, Meerbaum-Salant et al. created a new taxonomy that includes elements from both Bloom and SOLO. It views the SOLO categories unistructural, multistructural, and relational as super-categories, which all have the same three sub-categories: the revised Bloom taxonomy’s levels understanding, applying, and creating. In figure 2.3 we present a visualization of this taxonomy.

SOLO for Primary School  Learning taxonomies have been rarely considered when it comes to computer science education on the level of primary schools. As an exception, Linda Seiter presents an assessment designed to measure computational thinking skills of fourth grade students [46]. For this purpose, she uses a set of Scratch challenges [44]. She ran experiments and scored responses to the programming challenges based on correctness and the applied SOLO levels. She found that the performance of the pupils is heavily correlated with their reading skills. She claims that learning to synchronize multiple concerns within a single script is possible within a few weeks for pupils who read at fourth grade level.
2.5. Assessment Systems

So far we discussed assessment only from a didactical perspective and left away implementation aspects of computer-based assessment. There are many proposals of computer-based examination systems in literature, which is not surprising given the huge potential to reduce work particularly for educators. In this section we discuss notable contributions to the topic of computer-based assessment systems.

Semi-Automatic vs. Automatic  There are two common categories of computer-based assessment systems: semi-automatic and automatic systems. According to Chatzopoulou et al. semi-automatic systems usually differ from automatic systems in the following points [25]:

1. The teacher completes the grading.
2. The target group consists of novices that need support by a human.
3. The marking is flexible enough to allow partial points even when a task is not solved correctly.
4. The quality and efficiency of the code is feasible to be considered.

Therefore, the main drawback of automatic systems is that they can not take all aspects of programming into consideration. This could imply to only address the automatically checkable aspects in an exam. However, Ala-Mutka calls this idea unsuitable for bigger assignments and concludes that a combination of manual and automatic assessment is favourable [21].

Continuous Interaction  An early semi-automatic approach has been presented by Jackson, who relies on a model of continuous interaction between system and assessor [35]. The system performs as many checks as possible in fully automatic manner and additionally prompts questions about the task to the assessor. This way the system guides the assessor through the remaining correction work. Jackson considers a setting where students hand in their final answer by e-mail. This means that the assessment system presented by Jackson is not a priori an examination system because there is no information about the environment in which the student produces the solution.

Tool Access  English on the other hand presents a real semi-automatic examination system and closely describes its environment [26]. While external network connection should be prevented, English nonetheless advocates a setup including common development tools like compilers. While these tools theoretically allow the student to care less about syntax in the preparation, using them extensively during the exam imposes a prohibitive time penalty. The students know this and try to be well prepared from the start.

CourseMarker  Higgins et al. use a high degree of automation in their system *CourseMarker* [28]. Humans only have to react upon potential failures of the system. In describing their system, the authors unveil a huge didactical challenge in the setup of automatic systems: tasks
must be formulated and designed with extraordinary care because the automatic system has no possibility to discover misunderstandings. Furthermore, Higgins et al. present the idea of applying distinct sets of metrics to each task. This way they enable better feedback about the quality of the solution.

MOOCs More recent work in the area of assessment systems for programming often relates to the learning model of massive open online courses (MOOC). Pieterse presents an automated assessment system and lists both success factors and concerns that have to be considered in the development of such a system [43]. Particularly, she warns that automatic assessment involves the risk of suppressing the creativity of students. Pieterse further notices that in the context of MOOCs summative assessment may become unnecessary because the students learn for the sake of learning and not for credits. Consequently, formative assessment has to be emphasized, which implies an even increased need to provide high quality feedback to the students.

Testing, Verification, and CFG Similarity Vujošević-Jančić et al. present a grading framework for small size programming tasks, which is built upon a combination of traditional testing, software verification, and control flow graph (CFG) similarity [53]. They claim that these three components are all needed for sophisticated grading. The authors justify this by iteratively leaving away one component and presenting a task which cannot be graded satisfyingly only using the remaining two components. They built a prototype of their approach and perform an empirical evaluation to show that it provides grades that heavily correlate with manual grading.

Machine Learning One consequence of the emergence of MOOCs as learning model is the availability of a large number of digitized, graded programming snippets. Large amounts of data call for Machine Learning solutions and indeed Srikant and Aggarwal have developed a grading system building on a Machine Learning algorithm [48]. The aim of this system is to determine the closeness of a given program’s logic to a correct program.

2.6. XLogoOnline’s Judge

In this section we discuss XLogoOnline’s judge, which is the result of Dominic Weibel’s Bachelor Thesis [55]. We first explain how the judge works and then focus on how the judge can be used to examine pupils’ programming behaviour. To conclude the section, we compare our setup to Weibel’s setup and use this to justify that we can not directly reuse implementation aspects of XLogoOnline’s judge.

Daily Challenges Weibel integrated a judge into XLogoOnline allowing pupils to work more autonomously. The idea of this judge is that pupils can attend a daily challenge in the form of a small programming task. Such a task requests a pupil to draw a particular shape and the judge will check the pupil’s solution. To evaluate the correctness of the program, the judge
performs a pixelwise comparison of the produced image and a reference image. Further, the judge provides praising or comforting feedback depending on the outcome.

**Examine Pupils' Programming Behaviour** Arguably, the judge’s most interesting part is that it collects data about the pupils’ programming behaviour. This information can be used to gain valuable insights for teachers. For this purpose, Weibel discusses a set of metrics that describe a pupil’s programming behaviour. These metrics include the following:

- **Time to develop**: This metric measures the time or work spent while looking for a solution. This gives an indication of the difficulty level of a task.

- **Length of solution**: This metric measures how many commands were used to arrive at a solution. Remembering our discussion of taxonomies in section 2.4, this metric could be used to estimate the SOLO level of the response.

- **Usage of the editor**: This metric measures the level of modularity applied.

- **CS-Score**: This metric records how many times a pupil started over new when solving a task.

**Setup Differences** While we build upon findings of Weibel’s contribution, we can not directly reuse the judge. This is due to a fundamental difference in the setup: Weibel uses an automatic system and provides live feedback to pupils. We, on the other hand, decided for a semi-automatic system and we only deliver feedback to pupils via the teacher. Also, the judge is an integrated part of XLogoOnline, while we chose to evaluate pupils’ solutions in a designated teacher platform which is separated from XLogoOnline\(^2\).

---

\(^2\)We will motivate the decisions for a semi-automatic system, delivering feedback via the teacher and the introduction of a separate teacher platform in the following chapters.
2. Background and Related Work
Requirements

In this chapter we work out the high-level requirements for our examination system. We approach this by first discussing our system’s target environment and explaining how our system is intended to be used. This serves as a clean problem statement and facilitates neat modelling afterwards. We then proceed by deriving requirements. In this process we also present a questionnaire which we created and used to gain closer insights in the desires of teachers.

3.1. Use Case

In this section we present a use case to demonstrate how and in which context our system is intended to be used. The target context of our system are LOGO programming courses in primary schools where pupils work with XLogoOnline. At the end of such courses, teachers shall be able to review the learning progress of their pupils by carrying out an exam. The pupils solve this exam directly in XLogoOnline. In the following we provide a sequence of actions that explains how our system is intended to be used.

Prepare Exam  After working with XLogoOnline in class, a teacher wants to assess the abilities learned by pupils during the course. For this purpose, a teacher can use our system to create an exam to be carried out directly in XLogoOnline. An exam contains a set of tasks that can have different types. Most importantly, there are tasks which can be solved by providing a LOGO program. Teachers can write exam tasks using our system. Further, a teacher can define a set of pupils which will take the respective exam.
3. Requirements

**Carry Out Exam**  As soon as all pupils are present, the teacher can start the exam. The exam start happens at the same time for all pupils taking the exam. As soon as an exam is started, pupils see it in XLogoOnline and can start to solve it. When an exam is over, pupils are prevented from continuing to work on their exam.

**Correct Exam**  After the exam has been stopped, the teacher receives all pupils’ solutions in order to inspect and correct them. Our system supports the teacher in this process by taking on all aspects that can be corrected automatically. Further, our system guides the teacher through the correction work and enables him to complete the work efficiently. At the end of the correction work the teacher can assign pupils points depending on their performance. Further, a teacher can access statistics to compare pupils’ performances across different tasks or different exams.

3.2. Requirements

In this section we derive requirements for our system. For this purpose, we first present the questionnaire we used to figure out what primary school teachers want from our system. From this questionnaire’s results and our own experiences we then derive functional and non-functional requirements imposed on our system.

3.2.1. Questionnaire

In this subsection we discuss the results of our questionnaire, which we used to gather information about what primary school teachers require from our system. We start by motivating why relying on a questionnaire is useful. Afterwards, we discuss our most important findings resulting from the questionnaire.

**Customer Is Always Right**  Every software system’s success heavily depends on client approval. Therefore, software development has to start with formalizing the clients’ wishes and intentions. To reach this goal, it is inevitable to communicate with the clients. For this purpose, we wrote a questionnaire and asked teachers who are experienced in working with XLogoOnline to fill it out.

**Goal of the Questionnaire**  With our questionnaire we wanted to figure out how teachers would like to use our system. In particular, we were looking for aspects related to the school setting that we could miss because we lack teaching experience. For example that pupils are likely to forget user credentials is evident for every teacher. But this aspect might be overlooked by someone without teaching experience. The challenge of using a questionnaire to find such aspects is that we do not exactly know what we are looking for content-wise. We tried to overcome this challenge by formulating the question in our questionnaire very openly and giving teachers much freedom how to answer.
3.2. Requirements

Our Questionnaire’s Structure  In our questionnaire we first sketched our vision of the examination system for XLogoOnline. This gives teachers a clearer picture of our intentions and enables them to give a much more concise feedback. Our questionnaire then contains twenty questions on our design. Most of them require a free text answer to ensure that we do not influence the teachers’ answers by providing answer possibilities. The full questionnaire is given in appendix A.

Qualitative Feedback  We contacted about forty teachers\(^1\) and asked them to complete our questionnaire. Furthermore, a group of students of a pedagogical university filled out our survey. Five teachers and eight students responded with a completed questionnaire. Since most of the questions in our questionnaire focus on qualitative rather than quantitative feedback, this was enough to receive useful insights. We were able to draw some conclusions, which are summarized in the following paragraphs.

Tasktypes  One of the most important questions is how to present a task in XLogoOnline. This depends heavily on the nature of a task. Therefore, we wanted to know what questions teachers wish to ask in the context of a LOGO exam. In the following list we present the teachers’ answers:

1. Write a program that draws the given figure.
2. What kind of shape will be drawn after the execution of the given program.
3. Find and correct the error in the given program.
4. Complete the following program, such that it draws the given figure.
5. Adapt the following program, such that it uses a parameter.
6. Adapt the following program, such that it uses \texttt{repeat}.
7. Adapt the following program, such that it uses \texttt{subprogram}.
8. Optimize the given program, such that it yields the same result but uses less lines of code or draws less strokes.
9. Concerning the following figure and the current position of the turtle, determine the position where the turtle started the execution of the given program.
10. Move the turtle from A to B without drawing.
11. Why is it important to use \texttt{repeat}?

Considering this selection of exam tasks we notice that almost all expect a LOGO program as answers. The only exception are tasks 2, 9 and 11. These tasks either expect a text solution or could request pupils to choose one option of a set of possible answers. This means that by supporting only three types of answers (LOGO programs, multiple choice selection and free text) we would cover all tasks in this list.

\(^1\)We sent the questionnaire to 34 addresses, but some of these addresses are shared by multiple people. Also we do not know if some of the teachers have further spread the questionnaire. Therefore, the exact number of teachers receiving a questionnaire is not known.
3. Requirements

**Teacher Control**  Multiple questions dealt with the degree of automation our system would offer. It turned out that teachers do not want to lose control. They want to be in charge of correction and grading. They are reluctant to use a system that takes away their decision responsibility. Further, they want to have the control over the system in order to adapt to unforeseen situations. For example multiple teachers did not think that it would not be a good idea to stop exams automatically after a predefined amount of time. They want to be in charge of stopping the exam. However, teachers appreciate the possibility to tune parameters in order to personalize the system. We conclude that our system should be as adaptive as possible and that it should not take away control from the teacher. Our system should support the work of teachers and not replace it.

**Impediments to Using Our System**  We asked teachers about concerns that could prevent them from using our system. It turned out that most of their worries are related to the stability of the infrastructure. In particular, they fear undefined behaviour in case of lost connection to the Internet or in the case of a computer crash during an exam. Further, some of the teachers worry about automatic computer updates that can not be suspended. Our main consequence from this issues is the need for a backup functionality. In particular, a pupil must never lose his work progress due to infrastructure problems.

**Login System**  We asked teachers if they preferred a system requesting only the name of a pupil at the begin of an exam over a more complex login system with username and password. There was less criticism about a more complex login system than we expected. Apparently most of the teachers agree with a login system, despite the possibility of lost passwords.

**Sharing Accomplishments**  We wanted to know from teachers whether they would appreciate a shared task pool to which all teachers can contribute. Generally this idea was more popular than we expected. Some teachers even suggested that all tasks should be included in that task pool automatically. But there were also some doubts about practical issues. Most importantly, some teachers asked who would be responsible for the quality of these tasks and particularly the corresponding solutions. All the same, this feedback encouraged us to pursue the idea of a shared task pool further\(^2\).

### 3.2.2. Functionality

In this subsection we discuss our system’s functional requirements. Functional requirements specify a system’s behaviour, functions, and features. The described requirements are influenced by the feedback resulting from our questionnaire discussed in section 3.2.1.

**User Management**  Our system must be able to uniquely identify all actors involved with an exam. For this purpose, every actor must have the possibility to introduce himself to the

\(^2\)We leave implementing this feature to future work, as described in section 8.3, but we prepared our system for the introduction of such a task pool.
3.2. Requirements

system using unique credentials. Later the actor can use those credentials to access resources that belong to him. Apart from authenticating particular actors, the system must also be able to distinguish between pupils and teachers.

Exam Design A teacher must be able to create an exam containing multiple tasks of various tasktypes. The set of available tasktypes must be expressive enough to handle multiple choice questions and tasks that request a LOGO program as answer. Further, a teacher must have the possibility to determine which pupils have to solve an exam. The teacher must be able to save an exam and retrieve it again later. Pupils and teachers must be known to the system for this process.

Carrying Out Exams A teacher must be able to distribute an exam to all pupils associated to that exam. After this, access to the exam still has to be locked for all pupils until the teacher explicitly starts the exam. When the exam is started, a pupil must be able to switch between different tasks of the exam in any order he wishes. A pupil must be able to solve the tasks and hand in solutions to them. As soon as an exam is finished, a teacher must have the possibility to stop it manually. Stopping an exam must prevent the pupils associated to the exam from further modifying their solutions.

Correction After an exam, teachers must have a possibility to inspect their pupils’ solutions and correct them. Our system shall guide the teacher through the correction work and examine automatically checkable aspects. The system has to inform the teacher about the automatic checks’ results. During this process the teacher must be able to overwrite the suggestions of the system. At the end of the correction work, the system has to present an overview of performance statistics to the teacher. This way a teacher is enabled to compare the pupils’ achievements.

3.2.3. Non-Functional Requirements

A substantial part of a software system’s requirements is non-functional. Non-functional requirements describe must-have properties of a system that do not directly relate to the system’s behaviour, features, and functionality. We have developed a set of non-functional requirements and provide it in the paragraphs below. For the development of non-functional requirements we again considered our questionnaire discussed in section 3.2.1.

Similarity Our project’s intention is to enable pupils to solve exams directly in the environment XLogoOnline introduced in class. Therefore, the exam environment must look as similar to the standard XLogoOnline as possible. This suggests that we should not change XLogoOnline’s user interface except for the parts that are absolutely necessary. A further implication is to stick to the design goals of XLogoOnline’s UI. Most importantly, this includes that all relevant information must be visible at one glance for the pupil.
3. Requirements

**User Friendliness** We can not assume that the actors in our system are adept computer users. This part of our setup clearly differs from many examination systems found in literature, as most of them rely on the experience of their users. This fact implies that we must emphasize user-friendliness in the implementation of our system. Our system can only be successful if the involved actors are happy with it. This goal can only be reached if the system heavily emphasizes user-friendliness.

**Robustness** Our system must be robust and reliable even in unexpected situations. This includes both prevention of unwanted behaviour and recovery from unexpected system states. For example we must support pupils to stay connected with the system during an exam and provide recovery measures for the case our prevention measures did not succeed. Furthermore, we have to ensure that exams can be carried out without interruptions. Finally, we must prevent data loss for pupils and teachers even in scenarios that we can not foresee.

**Generality** Our project description requests us to design and implement an examination system for XLogoOnline. Thus, theoretically our examination system only has to support XLogoOnline. However, we decided that we want to follow a more modular approach. This means that we want to design and implement our system in such a way, that it is easily extensible to support programming IDEs other than XLogoOnline. In this statement we mainly think of WebTigerJython, a web-based IDE for Python programming which was also developed at our chair at ETH Zürich [51]. However, the generality requirement must not lead to a restriction of the support our system offers to XLogoOnline.
Design

In this chapter we show how we designed our system in order to meet the requirements presented in chapter 3. We first motivate why we model our system as a distributed system consisting of three nodes: Pupil-Client, Teacher-Client, and Server. After that, we proceed by assigning responsibilities to the distinct nodes of our distributed system. Finally, we discuss important design questions that affected our design.

4.1. Model of Distributed System

In this section we present our system’s underlying model. We first motivate our choice for a distributed system consisting of the three nodes Teacher-Client, Pupil-Client, and Server. We then emphasize the communication between these nodes and discuss why we decided that the Teacher-Client and the Pupil-Client communicate only via the Server. Afterwards, we consider each of our system’s three nodes in turn and focus on its responsibilities as part of the whole system.

Built Around XLogoOnline Since our system is conceived as extension of existing software, thinking about the design has to start with the already present part of the software. Therefore, our design has to be built around the existing implementation of XLogoOnline.

Separate Teacher- and Pupil-Platform One possibility to model our system would be to completely integrate the examination system into XLogoOnline. But this approach has significant disadvantages. Most importantly, the required features of teachers and pupils differ substantially. Therefore, it would be challenging to integrate the functionality required by teachers
4. Design

![Diagram of a tree-shaped distributed model with the Server as root and the Pupil-Client and the Teacher-Client as leaves.](image)

**Figure 4.1.** This figure shows our system’s tree-shaped distributed model with the Server as root and the Pupil-Client and the Teacher-Client as leaves. The edges between nodes describe a communication channel. Messages can be sent in both directions along an edge, however, the Server never sends a message on its own initiative. The Server only sends responses to received messages.

Integrating our system into XLogoOnline without colliding with the genuine design goals of XLogoOnline. Furthermore, integrating our examination system in XLogoOnline would conflict with the generality requirement motivated in section 3.2.3. For these reasons we discarded the idea of an integrated solution.

**Our System’s Nodes** Instead of integrating our system into XLogoOnline, we decided to do it the other way around: We integrated XLogoOnline into our system model. We model our system as a distributed system with multiple communicating nodes. One of these nodes corresponds to the existing design of XLogoOnline. We call this node Pupil-Client. Additionally, we introduced a node that provides the functionality required by teachers. This node is further referred to as Teacher-Client. We further chose to introduce a third node called Server which should provide server functionality. The main motivation for this choice was to provide the possibility for online storage to both teachers and pupils. Also, a small server offering a few basic features already existed for XLogoOnline. Thus, it was natural to adapt this server to be accessible by both teachers and pupils. The logic of the already present XLogoOnline server was easily extensible to the final server logic required for our system.

**Communication Between Nodes** The Teacher-Client, the Pupil-Client, and the Server are enough to cover all functionality, we do not need further nodes. The remaining question is how these nodes interact and particularly between which nodes pairwise communication is enabled. We decided to connect the nodes in a tree-shaped manner with the Server as the root and the Pupil-Client and the Teacher-Client as leaves. This design is shown in figure 4.1.

**Server Only Reacts** We chose our Server to be only reactive. This means that the Server only reacts upon requests and never contacts another node proactively. The motivation for this decision was to improve modularity. If a new Pupil-Client is to be introduced, it should not be necessary to change the Server’s implementation for this. Our server functionality is open for any actor that implements the communication interface and sticks to the rules defined by the Server.
4.1. Model of Distributed System

*Communication Via Server*

We decided that the Pupil-Client and the Teacher-Client do not have to communicate directly with each other. In the following we motivate this choice and discuss its implications.

**Easier Communication Scheme** One advantage of avoiding direct communication between the Teacher-Client and the Pupil-Client is that we do not have to establish a connection scheme between the Pupil-Client and the Teacher-Client. This would have been complicated, since pupils and teachers change the computers they are working on. Because of this, we can not assume that the Pupil-Client and the Teacher-Client have fixed IP addresses, as we can assume for the Server.

**Avoiding Dependencies** Having no direct communication between the Pupil-Client and the Teacher-Client helps us to avoid dependencies between them during the implementation. This is important in the case that we want our system to support a new client IDE different from XLogoOnline. With our model choice, this would be done by introducing a different type of Pupil-Client. The more the Teacher-Client depends on particular properties of the Pupil-Client, the harder it is to introduce a new Pupil-Client\(^1\).

**Privacy** If the Pupil-Client and the Teacher-Client can not directly communicate, this means that all resources they exchange go through the Server. For example every answer given by a pupil during an exam will pass the Server before arriving at the teacher. Because of this our system’s users might be worried about privacy issues\(^2\). However, the results from our questionnaire presented in section 3.2.1 indicate that teachers do not consider privacy as a critical issue in our system.

In the following subsections we take a closer look at the distinct nodes of our distributed system. In particular, we will assign tasks and responsibilities to the respective nodes.

### 4.1.1. Server

In this subsection we consider the Server and focus on its responsibilities as part of the whole system.

**Data Storage** One of the Server’s most important tasks is data storage. The Server must offer clients suitable instruments to store and retrieve resources from the Server. Most notably this includes exams, tasks, solutions, or just ordinary program texts. One important prerequisite

---

\(^1\)We want to emphasize that it is *not* required to introduce a new Teacher-Client in order to support an additional Pupil-Client.

\(^2\)Of course, from a technical perspective, providing this direct communication would not guarantee privacy any more than avoiding it implies absence of privacy. But for our system’s users the feeling of privacy could potentially be increased if we enabled direct communication between the Pupil-Client and the Teacher-Client.
4. Design

for this is a reasonable data management system. This data management system is responsible that no data loss can occur and that privacy is ensured for users.

**Communication Establishment**  Another important duty of the Server is to establish all necessary communication between the Teacher-Client and the Pupil-Client, as they can not directly communicate with each other. Therefore, the Server must enable data exchange between all its clients.

**User Management**  Both the aspect of data management and communication establishment build upon an important assumption: The Server must be able to distinguish pupils from teachers and identify individual users. Therefore, the Server is first and foremost responsible for the user management. This includes authentication of unique users, registration of new users, and the distinction between pupils and teachers. The latter is due to the fact that the Server has to offer functionality that is only accessible to teachers. This fact implies that the Server does not only have to perform authentication but also authorisation.

**API**  The Server provides an Abstract Program Interface (API) that defines and documents all requests it accepts. This API also determines the order of requests for all actions that involve cooperation between pupils and teachers.

4.1.2. Teacher-Client

In this subsection we focus on the Teacher-Client. For this purpose, we discuss its responsibilities as part of our overall system. Basically the Teacher-Client offers all functionality that must be available for a teacher. In particular, this means that designing, managing, and correcting an exam happens in the Teacher-Client.

**Create Exams**  To support exam creation, the Teacher-Client offers the possibility to create new tasks and put together an exam of existing tasks. Furthermore, the Teacher-Client provides suitable methods to manage pupils. Most notably for each exam a teacher can define a set of pupils taking the exam.

**Carry Out Exams**  The Teacher-Client offers the possibility to carry out an exam. A teacher can hand out an exam to the intended set of pupils. Further, the teacher can use the Teacher-Client to start and stop an exam.

**Correct Exams**  For the correction phase, the Teacher-Client presents all pupils’ answers to the teacher along with the original exam. The Teacher-Client supports a teacher while correcting exams. For this purpose, it gives the teacher feedback on the correctness of answers. For example the Teacher-Client informs the teacher on whether a program handed in by a pupil has the same result as the corresponding master solution. The Teacher-Client allows the teacher to
4.2. Design Choices

record the performance of pupils in an exam. Additionally, the Teacher-Client allows a teacher to compare the performance of pupils in different exams. To perform all mentioned tasks, the Teacher-Client cooperates with the Server and implements its API.

4.1.3. Pupil-Client

In this subsection we discuss the Pupil-Client’s responsibilities as part of our overall system. The Pupil-Client is a pupil’s entry point to the system. Its main purpose as part of our examination system is to enable pupils to solve exams.

**Carry Out Exams** The Pupil-Client can load an exam from the Server and display it to the pupil. Further, it provides a pupil possibilities to write answers and return these answers to the Server. The Pupil-Client also saves backups to the Server that prevent a pupil from losing his work progress in case of unexpected events. To perform all these tasks, the Pupil-Client implements the Server’s API.

**Pupil-Client vs. XLogoOnline** We want to emphasize that the Pupil-Client is not equivalent to XLogoOnline in our model. XLogoOnline is one particular implementation of the Pupil-Client. XLogoOnline also offers more functionality than required to be a valid Pupil-Client. However, this additional functionality is not part of XLogoOnline’s nature as Pupil-Client.

4.2. Design Choices

In this section we discuss important design choices that had a direct impact on how we chose to implement certain aspects of our system.

4.2.1. Teacher in Control

In this subsection we motivate and discuss the decision that the teacher is in charge of his pupils’ exam environment. During this process we directly refer to XLogoOnline instead of a generic Pupil-Client. While the design choice is also reasonable for potential future Pupil-Clients, the issues we face in enforcing it are specific to XLogoOnline’s implementation³.

**Prevent Cheating** From a pedagogical perspective a teacher has to be in control of his pupils’ exam environment in order to ensure that cheating is prevented. For this reason, in most examination systems described in section 2.5 a pupil can only use a subset of his computer’s features. For example browser usage or Internet access are often prevented. Obviously, we can prevent neither of them, since XLogoOnline is a browser-based IDE. We could only try to ensure

³The issues arise from XLogoOnline’s nature as website. However, a future Pupil-Client does not have to be a website.
4. Design

that a pupil is "locked up" in XLogoOnline for the course of an exam. But entirely locking a system from a website is not possible, because browsers open websites in a sandbox. This means that a website can not access or manipulate data on its host computer without browser support. For security reasons no browser allows a website to lock the entire computer. Therefore, we can not completely satisfy the constraint of teacher control over the exam environment in our setting.

Best Effort  We state in our model that the teacher has the control over the pupil’s exam environment, although we know we can not guarantee this in practice. Instead of promising a guarantee, we follow a "best-effort" strategy here. We offer a teacher the control over a pupil’s XLogoOnline during an exam. The only way a pupil can escape from the teacher’s control is by handing in the exam or closing the website.

Controlling XLogoOnline  Controlling XLogoOnline mainly concerns managing the functionality available for pupils during an exam. For example loading pre-written programs from the local storage into XLogoOnline should not be possible during an exam.

4.2.2. Authorisation of Teacher

In this subsection we consider the design choice that a teacher can only carry out an exam with pupils that allowed him to do this.

Power of Teacher  Having a teacher control a pupil’s exam environment gives him a substantial amount of power over the pupil’s side of our system. Theoretically, anybody, most notably a pupil, could claim to be a teacher and exercise that power. This could cause trouble and therefore we decided to introduce a safety mechanism by letting pupils confirm their teacher. This means that a pupil must give his teacher permission before the teacher can do anything that affects the pupil.

Advantages of Teacher Confirmation  One could argue that there is only a small probability that real problems would arise without this confirmation. All the same, we decided to introduce this confirmation mechanism. This decision was particularly motivated by the two following aspects. Firstly, from a pedagogical perspective we hope to sensitize pupils for privacy issues. It is an important lesson for pupils that on the Internet one has to be careful about data and permissions. Secondly, from a modelling perspective we prefer a system that prevents unwanted behaviour over one that relies on optimistically estimating the probability of unwanted behaviour.

4.2.3. Conflicts Resolving

In this subsection we elaborate on our design decision that teachers are in charge of resolving potential conflicts. Conflicts could for example arise if a pupil has to take multiple exams at the
4.2. Design Choices

**Number of Teachers**  So far we did not consider the number of teachers supported per pupil. It could prove valuable if we allow more than one teacher per pupil. For example there is the possibility of a temporary replacement for a teacher. Therefore, we do not want to restrict our system to support only one teacher per pupil in general.

**Potential Conflicts**  The issue with supporting multiple teachers is that there is a potential for conflicts between teachers. What if a teacher wants to start an exam for a pupil, that is already taking an exam right now\(^4\)? Sticking to the choice of multiple teachers, there is no way of completely ruling out conflicts.

**Who Resolves Conflicts?**  Since we can not completely rule out conflicts, we have to provide mechanisms to resolve them. From a design perspective the relevant question about this issue is: "who gets the responsibility to resolve conflicts?". It could be either the pupil or the teacher. Choosing the pupil would be straightforward to implement, since the pupil could just select in favor of one of the exams ready for him. The disadvantage of having pupils resolve conflicts is that we do not want to create too much overhead for pupils. Therefore, beginning an exam should be as easy as possible for a pupil. On the other hand, if teachers are responsible for resolving conflicts, we need a hierarchy among teachers. Otherwise we can not resolve conflicts arising through the involvement of two teachers, as they could disagree on how the conflict is resolved. Having a hierarchy among teachers effectively limits us to one teacher per pupil, as it hardly makes sense to have such power differences between teachers.

**Number of Teachers Per Pupil-Client**  Because drawbacks arise both from choosing teachers and pupils to resolve conflicts, we came up with a compromise: We decided that the responsibility over conflict resolving depends on the Pupil-Client’s particular implementation. For XLogoOnline as Pupil-Client we decided that teachers resolve conflicts. The consequence is that for XLogoOnline as Pupil-Client we only support one teacher per pupil. The motivation for this is that we favour user-friendliness and simplicity for pupils over an extended functionality.

---

\(^4\) A possible scenario for this problem to occur is that a teacher forgets to stop an exam. Stopping an exam has to be done explicitly by a teacher, as we discussed in section 3.2.2
4. Design
Tools and Technical Background

In this chapter we present the most important technologies used during this project and motivate why we chose to use them. First, we focus on the technology used to implement the communication between the nodes of our distributed system presented in chapter 4. Then we iterate over our system’s nodes and for each of them discuss which technologies we used in its implementation. Additionally, we introduce XLogoOnline from an implementation perspective.

5.1. Communication

In this section we consider the tools used for communication between our distributed system’s nodes. Choosing a certain technology for the communication is an important decision because it imposes additional requirements on potential technologies for the distinct nodes. We decided to rely on the Hyper Text Transfer Protocol (HTTP) [8].

5.1.1. HTTP

In this subsection we introduce HTTP. As HTTP was designed specifically to meet the requirements of the client-server model, it is a reasonable choice in our context. Furthermore, XLogoOnline was not standalone before our project but already contacted a server in order to provide online storage of LOGO programs. The communication in this predecessor system was implemented on top of HTTP. Thus, backwards compatibility was a further argument in favour of choosing HTTP.
5. Tools and Technical Background

**Communication Scheme**  HTTP is a protocol used to send documents over the web. It involves two actors: a client and a server. The client initiates the communication by sending a HTTP request to the server. The server processes this request upon receiving it and eventually answers with a HTTP response.

**Media Independent**  A very important feature of HTTP is that it is media independent [6]. Media independent describes the fact that there are no restrictions on the type of data sent in a HTTP message. For example the body of a HTTP request can contain a JSON object, an image or regular text.

**Stateless**  Another important property of HTTP is that it is stateless. Stateless means that successive requests between a client and a server are completely independent of each other. On the communication level, every request is treated as if it was the first ever. This allows the server to forget past communication, which makes HTTP scalable to large contexts.

**GET and POST Requests**  The two most used request types in HTTP are GET requests and POST requests. The purpose of GET requests is to fetch a resource from the server while POST requests are used to send data to a server.

**URI vs. URL**  In both GET and POST requests the server is identified using a given request Uniform Resource Identifier (URI). URIs are often used interchangeably with the term Uniform Resource Locator (URL). However, the two concepts are not exactly the same. As the names suggest, the purpose of an URI is only to identify a resource while URLs are additionally meant to locate a resource. Therefore, all URLs are URIs but not the other way around [17].

### 5.2. Server

In this section we focus on the tools and technologies used to implement our server. As already mentioned earlier, XLogoOnline was equipped with a basic server when we started our project. This server offered online storage of LOGO programs. Its code was written in python and made use of the library bottle [4].

In the next three subsections we introduce the tools and technologies used to implement our server. These are bottle, JSON Web Tokens [10], and the python library Pillow [13].

#### 5.2.1. Bottle

In this subsection we introduce the python library bottle. The small server XLogoOnline used before our project was implemented using Python and bottle. We decided to continue working with this technology mainly in order to provide backwards compatibility.
5.2. Server

Listing 5.1: In this listing we provide an example to demonstrate how bottle and JRTs work. The code is a simplified version of a method in our server implementation that saves an exam on the server. The @post decorator is used by bottle to map the path /saveExam/<exam_id> to this method. The method is only accessible by teachers therefore in line 3 we call validate to check the user corresponding to the JWT auth corresponds to a teacher. The method returns whether the exam was successfully saved. In section 6.1.4 we will explain in more detail how validation works. This example is about the usage of bottle and JWT.

Overview Bottle is a small, lightweight web framework that implements the Web Server Gateway Interface (WSGI) [18]. Bottle is provided as single file module and enables mechanisms like routing and file support. Bottle is distributed with a built-in HTTP development server and is compliant with other WSGI capable HTTP servers.

Routing Bottle offers convenient ways of mapping functionality to routes. For an example we consider the code snippet in listing 5.1. By using the @post decorator, bottle maps HTTP POST requests sent to the path /saveExam/<exam_id> to the corresponding method do_save_exam_by_id. The path is relative to the server base URL and <exam_id> is a route parameter. Bottle takes route parameters and uses them as method parameters. This means if a client sends a request to /saveExam/exam1, then exam_id has the value exam1 in the body of do_save_exam_by_id.

Content-Type In the example in listing 5.1 it is assumed that the incoming HTTP request’s header field Content-Type is set to application/json. Therefore, in line 5, we can access body of the POST request through the bottle object request.

File Support and Templates Bottle offers support for download requests and provides templates, that for example support the server in generating HTML for the front-end.

5.2.2. JSON Web Tokens

In our server implementation we made use of JSON Web Tokens to encode some information before transmitting it to a client. We use JSON Web Tokens for the purpose of authorisation.

Overview JSON Web Tokens (JWT) are an open standard for transferring encoded information between nodes of a network. The JWT implementation we used is a plugin recommended
5. Tools and Technical Background

by bottle [5].

**Authorisation** We use JWTs because they allow us to implement authorisation. Upon login we create a personalized JWT for the respective user. This JWT uniquely identifies the user among all users known to the server. It has to be included in the request header whenever a user requests a functionality with restricted access from the server. For instance, saving an exam is an operation with restricted access as only teachers are allowed to save an exam on the server. In such a method with restricted access, the server checks if the user associated to the token has the permission to call this method. Again we consider listing 5.1 for an example on this. The parameter `auth` contains the JWT and the method `validate` on line 3 performs the authorisation. During this process it checks if the JWT corresponds to an existing user and if yes, whether the respective user is a teacher.

5.2.3. Pillow

In our server’s implementation we also used a python library called Pillow [13]. This library is used to enable image support in the server. In this subsection we introduce Pillow.

**Why We Require Pillow** One common exam task is to let pupils write a LOGO program that results in a particular image drawn by the turtle. It is helpful for pupils if they can actually see the requested image, and not just work with a textual description of it. Therefore, we allow teachers to include images in exam tasks. Since exams can be saved on the server, we therefore also have to provide image storage.

**Downscaling Images** We want to control (and if necessary adjust) the size of uploaded images on the server side. For this we make use of the library Pillow. Pillow is a fork of the Python Imaging Library (PIL) and can be used for various tasks related to manipulating images. For example Pillow can be used to downscale images, which is our only requirement related to images on the server side.

5.3. Teacher-Client

In this section we consider the tools and technologies used to implement the Teacher-Client. In the development of the Teacher-Client we could start completely from scratch. Such a process begins with the decision for an appropriate technology. The chosen technology must enable us to satisfy all requirements imposed on the Teacher-Client. In the following we discuss how we arrived at the decision to rely on Angular [1] and TypeScript [16] for the Teacher-Client’s implementation. Afterwards, we introduce TypeScript and Angular from a technical perspective.

---

1 The JWT has to be called `auth` when used in bottle methods. The reason for this is the JWT plugin’s implementation.
5.3. Teacher-Client

**Server Compliance** The technologies used for the Teacher-Client must be compliant with the server. Since the server expects HTTP requests this only means that the technology for the Teacher-Client technology must be HTTP-capable.

**Website vs. Desktop Solution** Two reasonable choice for the Teacher-Client would have been a website such as XLogoOnline or a desktop solution. After careful consideration we decided in favour of a website. The main motivations for this choice were accessibility and reuse of XLogoOnline code.

**Accessibility** Accessibility is important in our context since we want to enable teachers to start using our system quickly and without the overhead of long installations. A website is a good choice in order to provide good accessibility as the user just needs a browser and does not have to perform additional installations.

**Reuse** We want to be able to use existing parts of XLogoOnline again in our Teacher-Client. In particular, for the correction part of our project this can be helpful. The argument of reuse implied that the Teacher-Client should be a website created with the same technologies as used for XLogoOnline: Angular and TypeScript.

**Missing Offline Support** The biggest downside of choosing a website to implement the Teacher-Client is the missing offline-support. We assume that teachers might enjoy the possibility to prepare or correct exams on their way to work in a train. Supporting this without Internet connection is not trivial in the chosen setting. But given the advantages of a website we consider this issue as an acceptable limitation for our system.

### 5.3.1. TypeScript

In this subsection we introduce the programming language TypeScript developed by Microsoft.

**Superset of JavaScript** TypeScript is a superset of JavaScript. This means that every JavaScript program is also a TypeScript program. The most important addition of TypeScript to JavaScript is the support for static typing. This means that in TypeScript a variable’s type is known at compile type, whereas JavaScript is an untyped language. TypeScript programs are compiled into plain JavaScript.

---

3One could argue that the reuse argument conflicts with our modularity requirement. To explain this, we consider the case that we want to introduce a new Pupil-Client to the System. Then indeed it is necessary to extend the implementation of the Teacher-Client, since the correction part is adapted to XLogoOnline. However, only the aspect of correction is affected by this. To carry out an exam and even display pupils’ solutions in the Teacher-Client, no changes are required. Furthermore, the correction part is a closed module in the implementation of the Teacher-Client as we will show in chapter 6. Therefore, also the extensions required in the Teacher-Client can be introduced in a modular way. Furthermore, in section 3.2.3 we stated that generality is secondary to an optimal support for XLogoOnline.
5. Tools and Technical Background

**Static Type Checking** During the compilation of TypeScript programs into JavaScript the rules of static typing are checked. Static type checking enables clean interfacing between different software components and therefore good support for modularity. Static type checking is one of TypeScript’s biggest advantages over plain JavaScript. For this reason TypeScript is also called "JavaScript that scales"[16].

**Classes** TypeScript does not only provide built-in types like `number` or `string`, but also offers the possibility to define classes and interfaces. Since the usage of type annotations is optional in TypeScript, type safety is still in the responsibility of the programmer. But by providing classes and interfaces TypeScript supports the programmer in order to write typesafe software.

5.3.2. Angular

In this subsection we introduce Angular. Angular is a platform and a powerful, programmer-friendly, and modern framework for web development. It provides everything required for building client applications in HTML and TypeScript. We will explain some of Angular’s most important features. Finally, in figure 5.1 we provide an Angular code example that uses the concepts introduced in this subsection.

**Directives** Angular provides extended HTML attributes called directives. There are three different kinds of directives in Angular:

- **Components**: The most important building blocks of Angular applications are components. A component is a TypeScript class, that has a view associated to it. This view is defined using a template consisting of HTML and CSS and controlled by the TypeScript class. Components can also contain other components, which are referred to as subcomponents.

- **Structural directives**: Structural directives are used to change the layout of the Document Object Model (DOM), which is the object model that defines properties, methods, and events for all HTML elements. Structural directives can add and remove elements from the DOM. The two most common structural directives are `*ngIf` and `*ngFor`. `*ngIf` is used to hide an element depending on a boolean property, that can be manipulated in a TypeScript class. `*ngFor` is used to create a DOM element for every element of a TypeScript array. We show the use of both `*ngIf` and `*ngFor` in our example in figure 5.1.

- **Attribute directives**: Attribute directives are used to change the DOM’s appearance. For example the attribute directive `NgStyle` allows to exchange the CSS class of a HTML element.

**Lifecycle of Directives** Directives have a lifecycle that is managed by Angular. That is Angular creates a directive, renders it, checks it when its data-bound properties change and destroys it before removing it from the DOM. Angular also provides lifecycle hooks to make
5.3. Teacher-Client

### Example

(a) This listing shows the service `ExampleService`.

```typescript
@Injectable()
export class ExampleService {
  public title: string = "Example";
}
```

(b) `AppComponent`’s resulting view.

```html
<h2 [ngStyle]="{'color':condition1 ? 'green' : 'red' }">
  {{exampleService.title}}
</h2>
<div class="blue" *ngIf="condition2"> </div>
<ul>
  <li *ngFor="let el of arr"> {{el}} </li>
</ul>
```

(c) This listing shows `AppComponent`’s HTML file ("app.component.html"). It demonstrates the usage of the attribute directive `ngStyle` and the structural directives `*ngIf` and `*ngFor`.

```typescript
//An Angular component is a typescript class with the decorator @Component.
//The templateUrl tells AppComponent where to find the corresponding HTML.
@Component({ templateUrl: './app.component.html' })
export class AppComponent {
  title: string = 'Example';
  condition1: boolean = true;
  condition2: boolean = true;
  arr: string[] = ["Hello", "World"];
  constructor(private exampleService: ExampleService) { }
  ngOnInit() {
    //Code to be executed when rendering is complete
  }
}
```

(d) In this Listing we present `AppComponent`’s Typescript class.

**Figure 5.1.** In this figure we present a small Angular code example. We consider an Angular component `AppComponent`, whose corresponding HTML code and TypeScript code are shown in the listing in figure 5.1c and in the listing in figure 5.1d respectively. In the listing in figure 5.1a we show an Angular service `ExampleService`, which is made available to `AppComponent` through dependency injection. In figure 5.1b we show the view resulting from our example. For simplicity we omit the corresponding CSS code in this example.
these moments visible and enable the programmer to react to them. An example for a lifecycle
hook is the method `ngOnInit()` which is called by Angular when the directives’s rendering
is complete.

**Services**  Another important building block of Angular applications are services. Services
offer functionality that is not directly related to specific views. In particular, they are used to
communicate between different components or to contact the server. Services are TypeScript
classes with the decorator `Injectable` making them available to Angular’s built-in mecha-
nism dependency injection. Dependency injection’s main intention is to define the initialization
order and to make services and other objects available for components. A component that wants
to use a service via dependency injection must include the service in its constructor as shown in
the listing in figure 5.1d.

**Router-Outlet**  Another notable construct provided by Angular is the Router-Outlet. A Router-
Outlet is a placeholder that can be filled with a component at runtime. The component in
a Router-Outlet can also change at runtime. The component displayed in a Router-Outlet is
determined by a route, which is the Angular name for a relative path. A Router-Outlet is a
convenient tool to support navigation on the website. The construct of Router-Outlets is also a
way to improve performance as it helps to control the rendering scope.

**Angular’s Advantages**  Among Angular’s advantages we want to emphasize the aspect
of modularity, which is important in our setting due to our generality requirement. The way
Angular applications are built of components and services provides support for modularity and
scalability in a very natural way.

### 5.4. Pupil-Client

In this section we focus on the tools and technologies used in a Pupil-Client. For this pur-
pose, we first discuss our requirements’ technological implications for a Pupil-Client. Since
during this project we focus on XLogoOnline as Pupil-Client, we then provide an introduction
to XLogoOnline from at technical perspective.

**Technological Implications for Pupil- Clients**  The Pupil-Client could be implemented
using any technology that allows the Pupil-Client to satisfy the requirements imposed on it in
section 4.1.3. Most importantly, the Pupil-Client must be able to carry out an exam. The only
additional restriction is that the technology has to be compliant to our choice for the server
technology. In particular, this means that the technology has to be HTTP-capable.
5.4. Pupil-Client

5.4.1. XLogoOnline

In the context of our project we mainly consider the particular Pupil-Client implementation XLogoOnline. Therefore, we want to introduce XLogoOnline from a technology perspective in this section. XLogoOnline is written using Angular and was already widely used when we started the project.

XLogoOnline’s Component-Tree  XLogoOnline is a single-page application. Due to this fact, XLogoOnline’s architecture can be drawn as a tree of Angular components as shown in figure 5.2a. The corresponding UI of XLogoOnline is shown in figure 5.2b.

AppComponent  XLogoOnline’s root component is called AppComponent. Its view comprises the whole visible page, although some parts are delegated to subcomponents.

 HeaderComponent  At the top of the page there is a navigation bar. It is the view associated to the HeaderComponent. Its main use is to personalize the environment for example by choosing the language or controlling the turtle’s drawing speed.

EditorComponent  On the left hand side of the AppComponent there is a component called EditorComponent. It contains the editor that can be used to write LOGO programs.

InputComponent  The right hand side of the AppComponent contains the InputComponent. It provides an input field, which can be used to execute LOGO commands and programs.

CanvasComponent  Below its input field the InputComponent contains the CanvasComponent, where the turtle draws the executed LOGO programs.

HistoryComponent  At the bottom of the InputComponent the HistoryComponent is located. It keeps track of commands and programs executed and displays that information. If XLogoOnline fails to execute a command because of an error, the HistoryComponent displays the corresponding error message.

Services  XLogoOnline also includes several Angular services. One important example of such a service is the LanguageService. It is responsible for translating all visible text into the language chosen by the user, respectively, to the host computer’s default language on startup. All components that display text use the LanguageService to request the translation of a text element into the language chosen by the user. When the user changes the language, this change is recorded in the LanguageService and Angular automatically performs all changes triggered by this event.
5. Tools and Technical Background

(a) This figure shows XLogoOnline’s component tree. The interpretation of such a component tree is that the view associated to a particular node is part of the view associated to its parent node. For example in our case the HistoryComponent is a subcomponent of the InputComponent.

(b) This figure shows XLogoOnline’s standard UI annotated with the names of the Angular components attached to the corresponding parts of the view. The rectangles’ colors correspond to the colors used in figure 5.2a.

Figure 5.2.
Implementation

In this chapter we describe how we implemented our system. We first present our system’s architecture and discuss important aspects of the communication between our system’s nodes. Afterwards, we dedicate the remaining sections to studying several example features of our system in detail. In particular, we will study how our system manages the pupils taking part in an exam, how exams are carried out, and how exams can be corrected in our system.

6.1. Architecture

In this section we provide an overview of our system’s architecture. We consider the nodes of our distributed system in turn and explain each node’s architecture. We then focus on how the different nodes communicate with each other. This includes a thorough discussion of our system’s error handling.

6.1.1. Server

In this subsection we present an overview of our server’s implementation, which is a collection of collaborating modules. For this purpose, we outline the server’s modules in the following list. This list is complemented by figure 6.1, where we visualize how the server’s modules interact:

- **RUN**: RUN is the module responsible for setting up the environment. It contains the server execution’s starting point and interconnects all other server modules. RUN initializes all libraries required. For example, RUN starts bottle’s HTTP development server and initializes the JSON Web Token plugin.
6. Implementation

![Diagram of server code structure]

**Figure 6.1.:** In this figure we visualize the structure of the server code. Each node is a server module and the directed edges represent a "has-access-to" relation. For example the RUN module can access the API module.

- **API:** API contains all methods that are attached to a bottle route. Therefore, every request sent to the server is parsed and validated in API. API forwards a successfully validated request to the module that offers the service demanded by the request. For example API forwards requests to retrieve an exam from the server to ExamServer.

- **UserManagement:** The UserManagement module is responsible for managing our system’s users. Most importantly, this involves login of existing users and registration of new users. For this purpose, we introduce usertypes that distinguish users of the different clients known to our server. For example XLogoOnline users have the usertype "xlogo". Additionally, a user has a username that uniquely identifies him among all users of the same usertype and a password to authenticate him.

- **CourseServer:** The task of the CourseServer is to connect teachers and pupils. For this purpose, the CourseServer defines the concept of a course. A course belongs to a teacher and contains a set of pupils that all have the same usertype. Each course has an id that is used to distinguish courses owned by the same teacher. Every course further has a recognition, which is used to globally identify a course.

- **ExamServer:** The ExamServer is responsible for managing exams. It handles all requests that modify an exam. An exam has an id that uniquely identifies it among all exams owned by the same teacher. An exam also contains a reference to the course that takes it. Additionally, an exam includes a set of references to the tasks that are part of it.

- **TaskServer:** The TaskServer module offers teachers the possibility to store, retrieve, and manipulate tasks. Tasks are stored separately from exams to support reusing tasks in multiple exams. Keeping exams and tasks separate also enables sharing of tasks between teachers. A task has an id that uniquely identifies it among all tasks owned by the same teacher. Optionally an image can be associated to a task.
6.1. Architecture

Figure 6.2: This figure displays the teachertool’s component tree. Nodes are Angular components except for the Router-Outlet. Directed edges represent a child-parent relation. Dashed edges indicate that a component can be routed into the Router-outlet. To simplify this overview, we only show the most important components.

6.1.2. Teacher-Client

In this subsection we consider the Teacher-Client’s implementation, which we call teachertool. We provide an overview of the teachertool’s architecture by introducing its most important Angular components. To describe how they interact, we further present the teachertool’s most important Angular services.

Components As Angular application the teachertool consist of components and services. In the following list we provide the teachertool’s key components. The teachertool’s component tree is visualized in figure 6.2 and complements this list:

- **AppComponent**: The AppComponent is the teachertool’s root component. Its main part is occupied by a Router-Outlet. The component displayed in the Router-Outlet depends on how a teacher navigates in the teachertool. For example if a teacher wants to modify a course, then the CourseComponent is displayed in the Router-Outlet. The AppComponent further contains the HeaderComponent.

- **HeaderComponent**: The HeaderComponent is visible in every state of the teachertool and offers functionality that should always be accessible. For example a teacher can configure the teachertool’s language in the HeaderComponent.

- **DashboardComponent**: The DashboardComponent is the first view a teacher sees after login. Its main purpose is to provide an overview over all functionality provided by the teachertool. Additionally, the DashboardComponent lists a teacher’s saved resources as for example courses. Selecting such a resource navigates the teacher to the component used for manipulating the chosen resource. For example selecting a course routes the CourseComponent into Router-Outlet.

- **CourseComponent**: The CourseComponent allows a teacher to modify a course. This includes adding or removing pupils from a course.
6. Implementation

- **ExamsDetailsComponent**: The main purpose of the ExamsDetailsComponent is enabling teachers to create an exam and run it. In order to create an exam, a teacher has to assign a course to the exam and set its tasks. When an exam is ready to be taken, a teacher can start and later stop it in the ExamsDetailsComponent.

- **FinishedExamsComponent**: The main purpose of the FinishedExamsComponent is to enable teachers to correct an exam. This includes our semi-automatic correction, which is discussed in detail in section 6.4. The FinishedExamsComponent further provides an overview of the pupils’ performance in an exam.

**Services** The teachertool also includes several services. The main purpose of these services is communication between different components and contacting the server. The most important services are summarized in the following list:

- **CourseService**: The main purpose of the CourseService is saving courses to the server and retrieving them from the server.

- **ExamService**: The ExamService is responsible for storing exams to the server and getting them from there. Further, it is used to start and stop exams on the server.

- **LanguageService**: The LanguageService is responsible for translating text elements visible in the UI into the language chosen by the teacher.

- **LoginService**: The purpose of the LoginService is to manage a teacher’s login state. This includes login, registration, and providing the teacher’s JWT access token to other services.

6.1.3. XLogoOnline

In this subsection we discuss how we extended XLogoOnline to support carrying out exams. For this purpose, we present the concept of the ExamMode. Additionally, we introduce the ExamService and provide an overview of our changes in the EditorComponent.

**ExamMode** To support running exams in XLogoOnline, we introduced the concept of the ExamMode. The purpose of the ExamMode is to distinguish whether an exam is currently running or not. The ExamMode is on if and only if an exam is either running or scheduled to start. XLogoOnline users can enter the ExamMode manually. When the ExamMode is on, some components adapt their functionality. For example the HeaderComponent allows an XLogoOnline user to load a LOGO program from local storage in normal mode. However, in the ExamMode this functionality is not available.

**ExamService** In order to manage all tasks arising from running an exam, we introduced the ExamService\(^1\). It is responsible for loading an exam from the server and saving a pupil’s

\(^1\)Both the teachertool and XLogoOnline have an ExamService. The functionalities of the two services are not related.
answer to the server. Further, the ExamService handles the logic involved in navigating between an exam’s tasks.

EditorComponent The main purpose of the EditorComponent is managing the editor pupils use to write their programs. When the ExamMode is on, the EditorComponent is further responsible for displaying an exam’s tasks.

6.1.4. Communication Between Nodes and Error Handling

In this subsection we discuss how we implemented the communication between our distributed system’s nodes. For this purpose, we first focus on our system’s error handling. Then we explain how a user can login to receive his JWT access token. To conclude this subsection, we discuss the example of saving an exam to the server. This example makes use of all concepts introduced in this subsection.

HTTP All communication between our distributed system’s nodes relies on the HTTP protocol. The server is the only node that can handle incoming HTTP requests. This means that direct communication between different clients is not enabled and that the server can not initiate communication with a client. Clients can issue a HTTP request to the server in order to demand a service from the server. The server reacts to such a request with a HTTP response.

Error Handling

In the following we focus on our system’s error handling. We first motivate why we need a clean error handling. Then we explain why HTTP’s error codes alone do not satisfy the requirements of our error handling. To overcome this problem, we present our own set of error-codes that complements HTTP’s error codes.

Why Error Handling During the processing of a request many errors can occur. Depending on the nature of an error, clients might want to react differently. As an example we consider the case that a pupil wants to retrieve an exam in order to solve it. Among the potential errors this request can trigger are the following:

- No exam is ready for the pupil.
- There is an exam ready for the pupil but it has not started yet. Thus, the pupil has not yet the permission to access the exam.
- The exam which the pupil wants to retrieve is already finished and the pupil is not allowed to proceed working on it.

All these reasons require a different reaction by the corresponding Pupil-Client. But the Pupil-Client can not react in different ways without being able to distinguish these situations in the first place. To enable this, we introduced a clean error handling.
HTTP Error Codes  HTTP has predefined response codes to distinguish certain error classes. For example response code 404 informs the user that the requested resource could not be found on the server. The error classes supported by HTTP are not fine-grained enough for our purpose. To illustrate this, we come back to the case that a pupil wants to retrieve an exam in order to solve it. HTTP’s API assigns response code 406 to cases when the server "does not find any content following the criteria given by the user agent"[7]. This description matches both the case that the requested exam has not started yet and the case that it has already finished. In order to distinguish these situation, we decided to add an additional layer of self-defined error-codes.

Our Error-Codes  We defined a customized set of error-codes. In case of an error during processing a request, such an error-code is included in the HTTP response-header to complement the HTTP response code. Our set is arbitrarily extensible, which enables good support when introducing new features to our system. We provide the current list of available error-codes including descriptions in figure 6.3. To illustrate the use of error-codes, we reconsider the example that a pupil wants to retrieve an exam in order to solve it. For the potential errors corresponding to this example our error handling proceeds as follows:

- No exam is ready for the pupil. In this case the HTTP response code is set to 404 and error-code to 5 (RESOURCE_NOT_FOUND)
- The exam has not started yet. In this case the HTTP response code is set to 406 and error-code to 12 (EXAM_NOT_YET_RELEASED)
- The exam is already finished. In this case the HTTP response code is set to 406 and error-code to 13 (EXAM_FINISHED)

Login

In the following we elaborate on important aspects of our system’s user management and explain how the server handles registration and login.

Registration  To register in our system, a user provides a username, a password, and a usertype. The usertype denotes the client a user belongs to and must be known to the server. The username must be unique among all users of the same usertype. If the registration is successful, the server stores all the user’s credentials. We encrypt the password before storing using the library bcrypt³ [3].

---

²Alternatively we could have used the fact that HTTP offers a range of error codes that can be customized. However, since this range is limited, we chose a more generic solution.
³We already encrypt a user’s password in the teachertool and XLogoOnline respectively. This avoids sending the password in plain text across the Internet. Still we encrypt the password on the server again. The motivation behind this is that stealing a user’s password from the server storage should not enable a villain to log in.
<table>
<thead>
<tr>
<th>Name</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNKNOWN_USERNAME</td>
<td>1</td>
<td>The provided username is not known to the server.</td>
</tr>
<tr>
<td>WRONG_PASSWORD</td>
<td>2</td>
<td>The provided password is wrong.</td>
</tr>
<tr>
<td>USERNAME_ALREADY_TAKEN</td>
<td>3</td>
<td>The provided username is already associated to another user.</td>
</tr>
<tr>
<td>BAD_TOKEN</td>
<td>4</td>
<td>The JWT token has not the correct format.</td>
</tr>
<tr>
<td>RESOURCE_NOT_FOUND</td>
<td>5</td>
<td>The requested resource was not found.</td>
</tr>
<tr>
<td>NO_PERMISSION</td>
<td>6</td>
<td>The user has no permission to access the requested functionality.</td>
</tr>
<tr>
<td>ERROR_FILE_READ_WRITE</td>
<td>7</td>
<td>Internal server error with file read/write.</td>
</tr>
<tr>
<td>RECOGNITION_ALREADY_TAKEN</td>
<td>8</td>
<td>This recognition is already associated to another course.</td>
</tr>
<tr>
<td>MISSING_REQUEST_PARAMETER</td>
<td>9</td>
<td>Your request is not in the correct format. Check the documentation of the API.</td>
</tr>
<tr>
<td>EXAM_NOT_READY_FOR_PREPARATION</td>
<td>10</td>
<td>The exam you want to distribute is not ready for distribution. Did you attach a course?</td>
</tr>
<tr>
<td>COURSE_NOT_READY_FOR_PREPARATION</td>
<td>11</td>
<td>The course associated to the exam is not ready for distribution. Are there pupils in the course?</td>
</tr>
<tr>
<td>EXAM_NOT_YETRELEASE</td>
<td>12</td>
<td>Exam is ready, but was not yet started and is therefore not accessible yet.</td>
</tr>
<tr>
<td>EXAM_FINISHED</td>
<td>13</td>
<td>Exam is already finished.</td>
</tr>
<tr>
<td>ALREADY_AN_EXAM_PREPARED_FOR_SOME_PUPILS</td>
<td>14</td>
<td>Pupils cannot take multiple exams in parallel.</td>
</tr>
<tr>
<td>UNKNOWN_USERTYPE</td>
<td>15</td>
<td>The provided usertype is not known to the server.</td>
</tr>
<tr>
<td>AMBIGUOUS_CALL</td>
<td>16</td>
<td>The requested functionality cannot be called unambiguously.</td>
</tr>
</tbody>
</table>

**Figure 6.3.:** In this figure we provide the list of available error-codes. An error-code is included in the header of the HTTP response in case of an error. This allows the client to distinguish responses with the same HTTP response code.
6. Implementation

**Login** From a user’s perspective, login is equivalent to obtaining the personalized JWT access token. For this purpose, the user must issue a login request containing the user’s username, usertype, and password. The server receives this request and checks that the provided password is correct. If it is correct, the server generates the JWT access token for the user by encoding the user’s username and usertype. Afterwards, the JWT token is sent to the user and login is complete.

**Example: Saving an Exam**

In the following we discuss the example of saving an exam to the server. We first introduce the server’s helper class `ServerResponse` and explain the most important aspects of request validation. Finally, we look at the server code to save an exam in order to demonstrate how request validation is handled.

**ServerResponse** A `ServerResponse` is an abstraction our server uses while processing a HTTP request. A `ServerResponse` is always associated to exactly one HTTP request issued by a client. It records results, potential errors related to the associated request, and the corresponding HTTP response code. For example if a teacher wants to retrieve an exam, the `ServerResponse` eventually contains the requested exam (in the success case). If an error occurs during the request’s processing, the `ServerResponse` contains the error-code after being processed. Finally, a `ServerResponse` is transformed into a HTTP response, that is sent to the client who issued the request.

**ServerResponses for Inter-Module Communication** `ServerResponses` are also used for communication between the server’s modules. Particularly, when API delegates processing a request to another module, it receives a `ServerResponse` in return.

**Validation Stages** Validating a request includes two stages. Firstly, we have to check that a request contains all parameters required for being successfully processed. Secondly, we have to check that the user issuing the request is allowed to access the demanded service. The most common cause to prevent a user from accessing a service is if the user has the wrong usertype. For example a pupil is not allowed to prepare an exam for another pupil.

**Code Example** The method `save_exam` provided in listing 6.1 is taken from the server module API and shows how the server handles a request to save an exam on the server. We first check that the request contains all parameters required to be successfully processed. Then we call `validate` to verify that the request was issued by a user of our system. For this purpose, we check the provided JWT access token. If the JWT access token corresponds to a known user, `validate` checks that this user has usertype "teacher". The reason for this is that only teachers are allowed to save an exam on the server. Then we delegate processing the request to the `ExamServer`. If the exam is saved for the first time, the `ExamServer` generates a new `id` for the exam before saving it. Finally, we transform the `ServerResponse` returned by the `ExamServer` to a HTTP response, which is sent to the teacher.
6.1. Architecture

```python
#API
@post("/saveExam", auth="access_token", method='POST')
def save_exam(auth):
    sr = ServerResponse()
    if request.json is None or EXAM not in request.json:
        # if the request’s content-type is not application/json (request.json is
        # None) or the request contains no exam to save (EXAM not in
        # request.json), then the request is invalid.
        sr.set_error_field(400, MISSING_REQUEST_PARAMETER)
    else:
        sr, t, _ = validate(auth, exp_usertype=TEACHER_TYPE)
        # if validation is successful, we delegate processing to the ExamServer
        exam = request.json[EXAM]
        sr = exam_server.save_ex(t, exam)
        # Finally sr is transformed into a HTTP response
        return sr.as_http_response()

def validate(auth, exp_usertype=None):
    sr = ServerResponse()
    username = usertype = None
    if "username" in auth and "usertype" in auth:
        username = auth["username"]
        usertype = auth["usertype"]
    if not user_management.check_exists_user(username, usertype):
        # The JWT is not associated to a known user
        sr.set_error_field(401, BAD_TOKEN)
    elif exp_usertype != None and exp_usertype != usertype:
        # The JWT is not associate to a user of the requested usertype
        sr.set_error_field(403, NO_PERMISSION)
    else:
        # the JWT is incorrectly formed
        sr.set_error_field(401, BAD_TOKEN)
        return sr, username, usertype
...

#ExamServer
def save_ex(teacher, exam):
    sr = ServerResponse()
    if not "id" in exam:
        # if the exam is saved the first time, we have to generate an id for it
        exam["id"] = generate_exam_id()
    try:
        utils.save_resource(teacher, EXAMS_DIRECTORY, exam)
    except Exception as e:
        sr.set_error_field(500, ERROR_FILE_READ_WRITE)
    return sr
```

**Listing 6.1:** In this listing we consider the function `save_exam` that saves an exam on the server for teachers. An incoming request is first validated by checking that it has the correct format and ensuring that it was sent by user of type "teacher". If the request can be successfully validated, `save_exam` delegates processing the request to the `ExamServer`. Finally, the resulting `ServerResponse` is transformed to a HTTP response.
6. Implementation

6.2. Managing Courses

In this section we discuss how our system manages the pupils taking part in an exam. We start by motivating the concept of a course and highlighting its advantages. We then mainly focus on our mechanism enabling pupils to enroll into a course.

Why Courses? A course is the abstraction we use to represent a set of pupils that take a particular exam. Alternatively, it would have been possible to directly attach pupils to an exam. Our concept’s main advantage is the support for exam series. This is relevant because the evaluation of our questionnaire’s results presented in section 3.2.1 suggested that teachers would like to compare pupils’ results across multiple exams. Exam series are not yet supported by the teachertool, but relying on the concept of courses it will be easy to add this functionality to the teachertool.

6.2.1. Course Enrollment

In this subsection we consider how pupils can enroll into a course. We start by explaining why this feature is required. We then explain how course enrollment is implemented and discuss how we handle synchronization issues arising from our implementation. To conclude this subsection, we elaborate on the aspect of teacher confirmation.

Why Course Enrollment? In order to carry out an exam, a teacher has to create a course that includes all pupils taking the exam. Pupils can be added to a course manually, provided the teacher knows the pupils’ usernames. However, manually adding pupils to a course is cumbersome for the teacher. For this reason, we introduced a feature that allows pupils to enroll into a course without the need for further teacher actions.

Course Recognition Each course has a recognition that globally identifies the course among all courses by all teachers. This recognition is created by the server, when the course is saved for the first time. The recognition can be changed in the teachertool. During this process the teacher is prevented from choosing a recognition that is already associated to another course.

Using the Recognition to Enroll To enable pupils to enroll into a course, the teacher provides them the course’s recognition. For example the teacher can write the course’s recognition onto the blackboard before the start of an exam. In order to enter the ExamMode in XLogoOnline, a pupil first has to provide a course recognition. This triggers a course enrollment request to the server.

Enrollment on the Server When the server receives a course enrollment request from a pupil, it uses the provided recognition to figure out which course the pupil wants to enroll into. Then the server adds the pupil to the respective course.
6.3. Running an Exam

Synchronization Issue  One issue arising from our enrollment mechanism is that the teacher is not the only actor that modifies a course. Therefore, we have to be careful about synchronization. To explain this, we consider the following scenario: Tom uses the teachertool to manipulate course1, which was previously loaded from the server. At the same time, Paola enrolls into course1. This results in two different versions of course1: The first is known to the server and contains Paola and the second is known to the teachertool and does not contain Paola. If Tom saves course1, the server’s version is replaced by the teachertool’s version. Therefore, the enrollment of Paola into course1 is lost. To avoid this problem, we need a mechanism to merge a course’s server version and teachertool version without losing updates on either side. For this purpose, we introduced the feature of reloading courses in the teachertool.

Reloading Courses in the Teachertool  Reloading a course enables a teacher to merge the course’s server version into the teachertool’s version without losing local updates. Course reloading adds all pupils of the course’s server version to the teachertool’s local version. This feature enables a teacher to recognize when all his pupils have enrolled into a course, such that he can start an exam.

Implicit Teacher Confirmation  So far in this section, we ignored the concept of teacher confirmation introduced in section 4.2. We interpret course enrollment as implicit confirmation of a teacher. This means that if Paola enrolls into course1 owned by Tom, the server records that Paola has confirmed Tom.

6.3. Running an Exam

In this section we consider all actions involved with carrying out an exam in XLogoOnline. We first define what kind of tasks we support for XLogoOnline exams. Then we focus on the actions required to prepare an exam. Afterwards, we elaborate on important aspects related to solving an exam in XLogoOnline. This includes introducing our backup mechanism used to restore a pupil’s working progress after leaving the ExamMode during an exam. To conclude the section, we present a diagram that summarizes all these actions.

Supported Tasks  The first step towards running an exam is creating an exam. One of the most important components of an exam is a collection of tasks. Therefore, we have to determine what kind of tasks we support. In section 3.2.1 we already outlined a list of tasks which teachers want to use in exams. Based on this list, we defined the following tasktypes, that are expressive enough to handle all tasks mentioned in section 3.2.1:

- DrawX: This tasktype describes tasks that request pupils to write a program that draws a certain shape X. An example task of type DrawX is: "Write a program that draws a square with side length 100"
- Extend Template: This tasktype allows a teacher to provide pupils a code template that they have to extend or modify. An example task of type Extend Template is "The following
6. Implementation

program should draw a square of side length 100. Unfortunately, there is an error in the program. Find the error and correct it."

- **Multiple Choice**: This tasktype describes tasks that request pupils to select a correct answer from a set of multiple answer possibilities. An example task of type Multiple Choice is "The image you see was drawn by the following LOGO program SQUARE. Given the turtle’s current position, at which point did the turtle start to draw the square? a) At the bottom left corner, b) at the top right corner, or c) in the square’s center?"

6.3.1. Exam Preparations

In this subsection we focus on the preparations necessary to set up an exam for pupils working with XLogoOnline. These preparations include setting up a course, distributing an exam to the pupils in the course, and starting the exam.

**Course Enrollment** Before exam1 can be run, its attached course course1 must be set up. For this purpose, all pupils of Tom’s class have to enroll into course1 as explained in section 6.2.1. When Paola enrolls into course1, XLogoOnline assumes that an exam is soon to be started. Therefore, XLogoOnline starts to periodically check if an exam is ready on the server for Paola. The server responds to this check with a HTTP response 404 as long as no exam is prepared for Paola. At the same time, Tom can reload the course in the teachertool as explained in section 6.2.1. This way he can check that all pupils have enrolled.

**Exam Distribution** As soon as course1 contains all pupils, Tom can distribute exam1. Distributing exam1 means that the server prepares exam1 for all pupils in course1, but does not allow them to access exam1 yet. Paola’s XLogoOnline still periodically checks if an exam is ready for her. As soon as Tom distributes the exam, Paola’s screen gets locked and she is informed that exam1 is soon to start.

**Exam Start** After distributing exam1, Tom can walk around the classroom to check that all pupils’ screens are locked. This should be the case for all pupils in course1. However, it is possible that a pupil has closed or reloaded XLogoOnline. This pupil has to enter the ExamMode again to see the same behaviour as all other pupils. Once all pupils’ screens are locked, Tom starts exam1 by pressing the corresponding button in the teachertool.

6.3.2. Exam Solving

In this subsection we focus on important aspects related to solving an exam in XLogoOnline. We start by explaining how pupils can navigate between tasks and what they have to do in order to solve tasks. We then focus on the concept of handing in tasks. Afterwards, we elaborate on our backup mechanism, which enables pupils to restore their working state after accidentally leaving the ExamMode.
6.3. Running an Exam

Receiving an Exam  Paola receives exam1 as soon as Tom has started it. To enable this, XLogoOnline started to poll for exam1, once it realized that exam1 is ready for Paola (when exam1 was distributed). When Paola receives exam1, her screen is released and she can start to solve the exam. Apart from the task navigation area and the description of the currently selected task, XLogoOnline looks exactly as Paola is used to.

Solving Tasks  Paola can select one of exam1’s tasks to be displayed in XLogoOnline’s EditorComponent such that she can solve it. Both the content of her editor and her canvas are directly associated to the selected task. To explain this, we assume that Paola has written and executed a LOGO program square for task1. Therefore, a square is visible on her canvas and her editor shows that code used to draw the square. If Paola switches to task2, both the square on the canvas and the corresponding code disappear. This is due to the fact that now the editor and canvas are associated to task2. If Paola switches back to task1, her square appears again both on the canvas and in its representation as a LOGO program in the editor. In addition to the editor and the canvas, also XLogoOnline’s history is associated to a task.

Handing In Tasks: Why?  Paola has to hand in her solutions manually and independently for every task in exam1. The motivation behind this is to enable Paola to improve her answers until exam1 ends. For example Paola can first hand in a solution to task1, then solve the rest of the exam and finally reconsider task1 in order to improve her answer. If Paola can improve her answer, she hands in task1 again. If she can not improve her answer, she does not hand in task1 again. In this case, her most recently handed in version of task1 will be considered during correction. The downside of this choice is that Paola could forget to hand in a task, which is equivalent to not solving a task. To help Paola not to forget handing in a task, we visualize for every task if it has been handed in yet. In figure 6.4 we show how this visualization and the ExamMode in general appear to Paola.

Handing In Tasks: How?  When Paola has finished a task, she can choose to hand in the corresponding task by pressing the corresponding button in XLogoOnline. When Paola hands in a task, XLogoOnline sends the corresponding history and editor to the server. The server saves these resources.

Restoring from Backup

In the following we focus on our backup mechanism that enables pupils to restore their working state after leaving the ExamMode. We explain why backups are important, what they are, and when they are saved to the server. Additionally, we discuss implications of our mechanism on the server’s memory consumption. Finally, we demonstrate how a pupil’s working state can be restored from a backup.

---

4In future work our system could be improved to automatically hand in all tasks that were never handed in during an exam. This means that when the exam is stopped, all tasks that appear gray in the navigation area are handed in. This feature will be discussed in section 8.3.
6. Implementation

**Figure 6.4:** In this figure we present a screenshot of XLogoOnline’s UI during an exam. On the left side, there is the navigation area. It contains the button to hand in the current task and visualizes for every task whether it has been handed in. If a task appears green in the navigation area it has been handed in. If it appears orange then it was modified after it was last handed in. If it appears gray, it has not been handed in at all yet.
6.3. Running an Exam

Listing 6.2: In this listing we present an example for a backup. The "id" denotes the exam’s id. "tasks" is an array containing all tasks of an exam. In this example there is just one task. For every task of an exam, the backup contains the corresponding id, the editor content and the history. In this example the history of the first task contains one element (“square”), which shows that the program square defined in the editor content has been executed once. Further, the backup contains the information whether a task has been handed in and if it has been modified after it was handed in the last time.

Why Backups Are Required  As we discussed in section 5.4.1, we can not guarantee that pupils do not leave the exam environment. Already from the modelling perspective this situation is uncomfortable, but in practice it could be a disaster. What if Paola accidentally exits the ExamMode and loses all her progress that was not yet handed in? If this happened, it would most certainly prevent Tom from further using our system. To solve this issue, we created a backup mechanism that enables restoring the last working state after accidentally leaving the ExamMode.

What Backups Are  A backup contains all information required to restore Paola’s most recent working progress, when she has accidentally left the ExamMode. For this purpose, a backup includes the editor content and the history associated to every task. Additionally, a backup contains the information which tasks have already been handed in. An example backup is displayed in listing 6.2.

When Backups Are Saved  While Paola is solving exam1, XLogoOnline periodically saves backup states to the server. Every action by Paola triggers a notification to XLogoOnline’s ExamService. When receiving such a notification, the ExamService measures the time passed since last saving a backup to the server. If this time exceeds a predefined threshold, the ExamService saves the next backup to the server.

Server Memory  We want to emphasize that neither the task description nor the corresponding image is included in the backup. Further, XLogoOnline saves a task’s canvas state as image
6. Implementation

during an exam to enable faster switching. These images are also not included in the backup. This is important to save memory space on the server. The reason for this is that the server saves all backups and not just the most recent one. The motivation for this is, that the information in the backups could be used to answer interesting research questions as we will discuss in section 7.3.

**Restoring Working Progress** Now we consider the case that Paola accidentally exits the ExamMode during exam1, for example by closing the browser. Then she has to log in again and enter the ExamMode. When she enters the ExamMode, XLogoOnline starts checking on the server whether an exam is available. Since exam1 is ready and even started, the server immediately responds by sending exam1 to Paola’s XLogoOnline. Along with exam1, the server sends Paola’s most recent backup. XLogoOnline can now restore the state from before exiting the ExamMode. This means that for every task the corresponding editor and history are set according to the backup. Every task’s canvas content has to be redrawn from the history, which is done lazily. This means that a task’s canvas is only redrawn if the task is selected again by Paola.

6.3.3. Summary

In this subsection we present a summary of all concepts introduced in the enclosing section. For this purpose, we show a workflow diagram in figure 6.5 that gives a high-level overview over all process involved with carrying out an exam in XLogoOnline.

6.4. Correction of LOGO Exams

In this section we discuss how semi-automatic correction is implemented in the teachertool. We first describe how we support teachers that want to correct a LOGO exam manually. Afterwards, we derive and present the algorithm implemented in the teachertool’s automatic correction. To conclude the discussion, we provide the source code of automatic correction in a simplified version.

6.4.1. Manual Correction

In this subsection we focus on how the teachertool enables teachers to correct exams without relying on our automatic correction. We first discuss how pupils’ answers to LOGO tasks are represented in the teachertool. Then we describe how we assist the teacher to speed up the manual correction.

**TaskSolutions** The teachertool represents a pupil’s answer to a particular task as TaskSolution. A TaskSolution contains the corresponding task’s description and the pupil’s
6.4. Correction of LOGO Exams

Figure 6.5: In this figure we present an overview of all actions involved with carrying out an exam. For this purpose, we look at the most important messages exchanged between our distributed system’s nodes. Each node is associated to a vertical edge representing the node’s execution over time. Edges connecting nodes represent messages between nodes, where the message is sent along the edge’s direction. Running exam1 first requires all pupils to enroll into course1. Tom then reloads course1 to check that all pupils have enrolled. If this is the case, Tom distributes exam1. Before the distribution, Paola’s XLogoOnline received an "exam not found" message when trying to access it. When she tries again after the distribution, she receives an "exam not released" message. When Tom starts exam1, Paola gets it on her next try to access it. Paola now has time to solve exam1. During this time, she repeatedly sends backups and solutions to the server. Eventually, Tom stops exam1. If Paola tries to save another solution to the server after exam1 is finished, the server rejects this with an "exam finished" message. Tom can now request all pupils’ solutions from the server and start to correct exam1.
corresponding editor content and history. A teacher can set multiple parameters on a TaskSolution to record the conclusions of correcting a task. The most important such parameters are:

- **Correctness**: Correctness describes if the figure drawn by the pupil has the correct shape. Correctness can be assigned one of the enum values correct, incorrect or not yet corrected.

- **ScaleCorrectness**: ScaleCorrectness describes whether the pupil’s resulting image has the correct scale. For example if a pupil draws a square of side length 100 instead of 200, the shape is correct but the scale is not. ScaleCorrectness can be assigned one of the enum values correct, incorrect or not yet corrected⁵.

- **AwardedPoints**: Another parameter of a TaskSolution is AwardedPoints. It records the points awarded for a solution and can be set to any number between zero and a maximum, which is defined per task.

### Reexecuting LOGO Programs

To facilitate manual correction, the teachertool enables reexecuting a pupil’s LOGO program program1. After executing program1 we display the resulting image. Along with program1 we execute the corresponding master solution solution1. We also display the image resulting from solution1, which enables the teacher to compare the two resulting images.

### Modularity

To enable execution of LOGO programs, we reused a big part of XLogoOnline’s source code. In order to ensure modularity, we included the reused XLogoOnline functionality in a separate module, which uses a service to cooperate with the rest of the teachertool. If a new Pupil-Client A was introduced to our examination system, we would apply the same approach and include all of A’s functionality in a new module.

### 6.4.2. Automatic Correction

In the following we discuss how we implemented automatic correction in the teachertool. We first show how teachers can use the automatic correction. Afterwards, we elaborate on why correction of LOGO programs comes down to comparing images pixelwise. We then derive our 1-Neighbourhood policy for image comparison and elaborate on challenges faced while implementing this policy. To conclude the discussion, we provide the source code of our automatic correction in a simplified version and present a visualization of our algorithm.

### Teacher Control

A teacher can request the teachertool to correct a TaskSolution for him. During this process, the teachertool sets the values for Correctness, ScaleCorrectness, and AwardedPoints. All these values can be overridden, if the teacher disagrees with the system’s correction. This ensures that teachers do not lose control over the

⁵For Multiple Choice tasks we set the value of ScaleCorrectness equal to the value of Correctness, since the concept of ScaleCorrectness has no meaning for Multiple Choice tasks.
6.4. Correction of LOGO Exams

Listing 6.3: The two LOGO programs shown in this listing both draw a circle with radius 100, although they have very different implementations. To the human eye the two resulting circles look equivalent. All the same, the resulting images do not match in every pixel. Therefore, an image comparison algorithm implementing an exact match policy will return “no match” when comparing the outcomes of the two programs.

Correction Is Program Comparison All tasks that we consider have in common that pupils have to hand in a LOGO program\(^6\). This program must be compared to a master solution. Comparison of two LOGO programs comes down to comparing images, since LOGO programs are image representations. The question is which image representation we want to use in the comparison.

Limitations of Program Text Comparison Theoretically, it would be possible to compare the programs on textual basis. But we can not expect satisfying results from this because very different program texts can lead to the same resulting image. The two LOGO programs shown in listing 6.3 are an example for this problem. The two program texts have no similarity but the images resulting from them look equivalent.

Assumptions for Pixelwise Comparison We decided to compare LOGO programs by comparing their resulting images pixelwise. To simplify the discussion, we assume that the background of the resulting images is white and that strokes are not white. Further, we assume that the compared LOGO programs are syntactically correct and that each program draws at least one stroke.

Limitations of Exact Match Policy Comparing images pixelwise was also the approach used by Weibel when he integrated a judge into XLogoOnline [55]. Weibel implemented an exact match policy in his algorithm to determine whether two images are the same. His algorithm returns true if and only if two images match in every pixel. The exact match policy’s advantage is that false positives are avoided. The disadvantage is that it results in false negatives for seemingly easy tasks. For example in listing 6.3 we consider two LOGO programs that both draw a circle with radius 100. The two resulting circles look equivalent to the human eye but have small differences on the pixel level. Since these differences are not visible, a teacher would

---

\(^6\)We ignore Multiple Choice tasks for the rest of this section, since they are trivial to correct.
expect our algorithm to recognize that the two images are equivalent. In order to achieve this, we dismissed the exact match policy.

**k-Neighbourhood** To derive our algorithm, we introduce the concept of a k-Neighbourhood. In order to simplify the discussion, we assume that a pixel’s value is a real number. We then consider an image as a matrix $M \in \mathbb{R}^{m \times n}$ of pixels $p = M_{i,j}$, where $i \in \{1, \ldots, m\}$ and $j \in \{1, \ldots, n\}$. We will further refer to this image representation as pixel matrix. For $k \in \mathbb{Z} \geq 0$ we define $p$’s k-Neighbourhood $N_k(p)$ as follows:

$$N_k(p) = \{ M_{i',j'} \in M : \text{round} \left( \sqrt{(i - i')^2 + (j - j')^2} \right) \leq k \}$$

where \text{round} maps a real number to its closest integer

As an example of how a k-Neighbourhood can be used, we next discuss how the exact match policy can be described using a 0-Neighbourhood. This will lead us to the definition of the 1-Neighbourhood policy we applied in our algorithm.

**Exact Match with 0-Neighbourhood** According to the definition of a k-Neighbourhood, the 0-Neighbourhood of a pixel $p$ is \{ $p$ \}. The 0-Neighbourhood can be used to describe the exact match policy as follows: For every pixel in $image1$ the 0-Neighbourhood of $image2$’s corresponding pixel must contain a pixel of the same color. Formally, we consider $M^{(image1)} \in \mathbb{R}^{m \times n}$ and $M^{(image2)} \in \mathbb{R}^{m \times n}$. The exact match policy requires:

$$\left( p = M^{(image1)}_{i,j} \right) \land \left( q = M^{(image2)}_{i,j} \right) \Rightarrow \exists q' \in N_0(q) : \text{color}(p) = \text{color}(q')$$

where $i \in \{1, \ldots, m\}$ $\land$ $j \in \{1, \ldots, n\}$

**1-Neighbourhood Policy** We chose to relax the exact match policy and consider a 1-Neighbourhood instead of a 0-Neighbourhood. We call this strategy 1-Neighbourhood policy. Formally, we consider $image1$ and $image2$ as equivalent with respect to the 1-Neighbourhood policy, if the following holds for their respective pixel matrices $M^{(image1)} \in \mathbb{R}^{m \times n}$ and $M^{(image2)} \in \mathbb{R}^{m \times n}$:

$$\left( p = M^{(image1)}_{i,j} \right) \land \left( q = M^{(image2)}_{i,j} \right) \Rightarrow \exists q' \in N_1(q) : \text{color}(p) = \text{color}(q')$$

where $i \in \{1, \ldots, m\}$ $\land$ $j \in \{1, \ldots, n\}$

Intuitively, the 1-Neighbourhood policy requires that for each pixel $p$ in $image1$ the corresponding pixel $q$ in $image2$ must have an adjacent pixel $q'$ that has the same color as $p$.

**Pen Width in LOGO** To implement the 1-Neighbourhood policy, we employ LOGO’s concept of pen widths. By default, every stroke of a LOGO program is drawn with pen width 1. A stroke drawn with pen width 1 is one pixel wide. Setting the pen width to 3, results in strokes that are three pixels wide.
6.4. Correction of LOGO Exams

(a) This listing shows the LOGO program `fig`, which was used to draw the image shown in figure 6.6b.

```logo
to square1
repeat 4 [fd 200 rt 90]
end
to square2
repeat 4 [bk 200 rt 90]
end
to fig
setpc blue setpw 3 square1
setpc red setpw 1 square2
end
```

(b) This figure shows the image produced by executing the LOGO program `fig` provided in the listing in figure 6.6a.

Figure 6.6.: This figure visualizes the implementation challenge arising from different starting points of equivalent LOGO programs. Both `square1` and `square2` defined in the listing in figure 6.6a draw a square of side length 200. However, the two programs start to draw the square at different points. Therefore, the second square is not drawn upon the first square. To overcome this challenge, we introduce image clipping to our image comparison algorithm.

Idea of Our Algorithm Our algorithm to compare `program1` and `program2` conceptually works as follows: We execute `program1` with pen width 3 which results in `image1`. Then we execute `program2` with pen width 1 upon `image1`, which results in `image2`. If `program1` and `program2` are equivalent, then `image1` and `image2` look exactly the same because every stroke of `program2` is drawn upon a (thicker) stroke of `program1`. Therefore, we then compare `image1` and `image2` with an exact match policy.

Implementation Challenge Our algorithm’s high-level idea is simple but not trivial to implement. One reason is that we can not ignore the starting points of our programs under comparison. To explain this challenge, we consider the two programs given in the listing in figure 6.6a. Both programs draw a square of side length 200, but `square1` starts drawing at the square’s bottom left corner and `square2` starts drawing at the square’s upper right corner. The result of executing `square2` on top of the image resulting from `square1` is shown in figure 6.6b. The square drawn by `square2` does not lie upon the square resulting from `square1`. Therefore, our algorithm would fail to recognize that the squares are equal if we ignored the programs’ starting points.

Image Clipping To overcome the challenge of different starting points, we execute the compared programs separate from each other and clip the resulting images. Clipping an image `image1` means that we calculate the smallest possible rectangle `r` such that every stroke in `image1` is completely contained in `r`. The clipped version of `image1` is the part of `image1` that is inside `r`. For the rest of this section, we always consider images in their clipped version.
6. Implementation

Our Image Comparison Algorithm  In this paragraph we explain our image comparison algorithm. For this purpose, we show how it checks that a pupils program pupilprog matches the corresponding master solution solution. We first execute solution with pen width 3 which results in image1. Then we execute pupilprog with pen width 1 to obtain image2. Afterwards, we check for every non-white pixel in image2 that the corresponding pixel in image1 has the same color. If this check succeeds, we know that pupilprog draws no wrong strokes. Additionally, we have to ensure that pupilprog draws all correct strokes. To achieve this, we repeat the above process, but switch the programs. This means we execute pupilprog with pen width 3 to receive image3 and solution with pen width 1 to get image4. Then we check for every non-white pixel in image4 that the corresponding pixel in image3 has the same color. If this check succeeds too, we know that pupilprog draws all correct strokes. Therefore, we know that pupilprog draws the same image as solution with respect to the 1-Neighbourhood policy.

Image Dimensions  So far we assumed that the pixel matrices of two compared images have the same dimensions. Of course this is not automatically true. Since we consider images in their clipped version, we know that two images are not equivalent if they have different sizes. Therefore, we could just state that the images differ and consider the task done. But we also want our correction to recognize when two programs draw the same shape but in a different scale. For this purpose, we have to employ image scaling.

Challenges Concerning Scaling  If we receive two images of different sizes, we have to rescale one of them before comparing them. For this we can not use our used representation of the pixel matrix. The reason for this is that we only want rescaling to affect the strokes’ lengths but not their widths. Otherwise we could not use our comparison algorithm on rescaled images, since there we assume that strokes always have width 1. Performing rescaling directly in LOGO program text would be difficult, since we can not assume that the scaling factor is the same for the height and width of an image. This means that we not only have to consider the length of strokes but also their slope. Therefore, we need a third image representation.

SVGImage  SVGImage is XLogoOnline’s intermediate representation for LOGO programs before they are drawn to the canvas. This means that every LOGO program is mapped to an SVGImage in XLogoOnline. This SVGImage is then drawn to the canvas. We decided to rely on SVGImages for image scaling. An SVGImage represents a stroke s as pair of endpoints: $s = ((x_1, y_1), (x_2, y_2))$. This represents a stroke from $(x_1, y_1)$ to $(x_2, y_2)$.

Scaling of SVGImage  To scale image2 to the size of image1, we first calculate the scale factors $s_h$ for the height h and $s_w$ for the width w as follows:

$$s_h = \frac{h_{\text{image1}}}{h_{\text{image2}}}, \quad s_w = \frac{w_{\text{image1}}}{w_{\text{image2}}}$$

Afterwards, we consider the center point $c = (x_c, y_c)$ of our image. $c$ corresponds to the center point of the smallest circle that surrounds all strokes. To transform an arbitrary stroke endpoint
6.4. Correction of LOGO Exams

If \( p_{\text{old}} = (x_{\text{old}}, y_{\text{old}}) \) is an endpoint of \( \text{image2} \) and \( p_{\text{new}} = (x_{\text{new}}, y_{\text{new}}) \) the corresponding endpoint in \( \text{image2} \)’s scaled version, we compute:

\[
\begin{align*}
    x_{\text{new}} &= c_x - (c_x - x_{\text{old}}) \cdot s_w \\
    y_{\text{new}} &= c_y - (c_y - y_{\text{old}}) \cdot s_h
\end{align*}
\]

This procedure is applied to all stroke endpoints in \( \text{image2} \) to obtain the scaled version of \( \text{image2} \).

Summary

We have discussed all aspects of our image comparison algorithm on a detailed level. To conclude this section, we put the single pieces together. For this purpose, we visualize our image comparison algorithm in figure 6.7. Additionally, we provide a simplified version of the source code corresponding to the correction of LOGO programs in listing 6.4.
6. Implementation

(a) This listing shows the LOGO programs used to draw the image shown in figure 6.7b.

```logo
1 to prog1
2  fd 4  rt  90  fd 11  rt  90
3  fd 4
4  end
5
6 to prog2
7  fd 4  rt  90  fd 11  rt  90
8  fd 4  rt  90  fd 1
9  end
10
11 to fig
12  setpw 3 setpc red prog1
13  setx 0 sety 0
14  setpw 1 setpc black prog2
15  setx 0 sety -20
16  setpw 3 setpc blue prog2
17  setx 0 sety -20
18  setpw 1 setpc green prog1
19  end
20
21 (b) The drawing in this figure was produced by executing the LOGO program fig displayed in the listing in figure 6.7a. The figure heavily zooms in on the image resulting from fig.

Figure 6.7.: In this figure we visualize how our image comparison algorithm works. We first execute prog1 with pen width 3 (drawn red for better visualization). Then we execute prog2 with pen width 1 (drawn black) upon the result of prog1. Because every black stroke lies upon a red stroke, we say that prog2 draws no wrong strokes. Then we switch the programs, that is, we draw prog2 with pen width 3 (drawn blue). Afterwards, we execute prog1 (green) upon the result of prog2. Because every green stroke lies upon a blue stroke, we say that prog2 draws all correct strokes. Therefore, our algorithm considers prog1 and prog2 as equivalent with respect to the 1-Neighbourhood policy. However, prog1 and prog2 would not be equivalent with regard to the exact match policy.
public correctTask(thickProgram, thinProgram, first=true) {
    var isCorrect = false;
    var isScaleCorrect = false;

    // execute first program with pen width 3 and clip resulting image
    var penWidth = 3;
    var thickImage = executeProgram(thickProgram, penWidth);
    thickImage = clipImage(thickImage);

    // execute first program with pen width 1 and clip resulting image
    penWidth = 1;
    var thinImage = executeProgram(thinProgram, penWidth);
    thinImage = clipImage(thinImage);

    // check that the sizes of the images match
    if (checkSameSize(thickImage, thinImage) {
        // check that the thin image lies on the thick image
        isCorrect = checkThinImageOnThickImage(thickImage, thinImage);
        isScaleCorrect = isCorrect;
    } else {
        // rescale the thin image before comparing the two images
        thinImage = rescaleThinImage(thinImage);
        isScaleCorrect = false;
        if (checkSameSize(thickImage, thinImage)) {
            // check that the thin image lies on the thick image
            isCorrect = checkThinImageOnThickImage(thickImage, thinImage);
        }
    }

    var result = null;
    if (!first) {
        // we are in the recursive call (see below)
        result = new Tuple(isCorrect, isScaleCorrect);
    } else {
        // if we arrive here and isCorrect = true, then we know that thinProgram
        // draws no wrong strokes. To ensures that it also draws all correct
        // strokes, we recursively call correctTask and switch the programs
        result = correctTask(thinProgram, thickProgram, false);
        result[0] = result[0] && isCorrect;
        result[1] = result[1] && isScaleCorrect;
    }
    // return the result that informs about correctness and scaleCorrectness
    return result;
}

Listing 6.4: In this listing we present a simplified version of the code that corrects a pupil’s LOGO program. When calling correctTask we set thickProgram to the pupil’s program, thinProgram to the master solution and first to true.
6. Implementation
Evaluation

In this chapter we evaluate our system. We start by explaining the setup used for this evaluation. We then focus on functional aspects of our system and describe our experiences while working with the system. This includes improvements that were implemented after testing our system. Afterwards, we discuss how our system’s backup mechanism could be used to approach interesting research questions.

7.1. Setup

In this section we describe the setup in which we evaluated our system. We used a programming contest in a primary school to test our examination system. We first explain the context in which this contest was carried out and then present the tasks that were part of this contest.

Contest in ABZ Course  We tested our system in a LOGO course organized by ABZ. The course was integrated into the regular classes of a fifth grade with ten male and nine female pupils. All pupils were between ten and twelve years of age. During the LOGO course the pupils got used to working with XLogoOnline and learned the most important concepts of LOGO programming. In the last course session we used our examination system to carry out a programming contest.

Tasks in the Contest  Each of the contest’s tasks required pupils to write LOGO programs to draw a certain figure. In figure 7.1 we present the images pupils had to draw in the contest. The full contest including all task descriptions provided to pupils is given in appendix B.
7. Evaluation

Figure 7.1: In this figure we present the tasks that pupils had to solve in the programming contest we used to test our system.
7.2. Functional Evaluation

In this section we discuss the functional aspects of the insights we gained during our system’s evaluation. For this purpose, we focus on the challenges we overcame while working with the system and ignore aspects that worked correctly from the start. We start this section with reviewing the preparations for the contest. Then we reflect on aspects related to carrying out the contest. Finally, we discuss our experiences while correcting the contest.

7.2.1. Exam Preparations

In this subsection we give an overview of the most important aspects related to preparing the contest. We start by showing how we ensured that the contest is not biased to our system’s advantage. Further, we discuss an issue related to the order of tasks inside an exam.

**Task Sources** We did not invent new tasks for the contest but reused tasks from two sources. The contest’s first three tasks are taken from Weibel’s Bachelor Thesis [55]. The rest of the tasks are taken from contests carried out in previous LOGO courses organized by ABZ. We decided to reuse tasks in order to reduce the risk that we adjust the contest to our system’s strengths.

**Task Order** The only issue we found in the teachertool during the contest’s creation was related to task ordering. When a teacher retrieved an exam from the server, all tasks of an exam were downloaded in parallel. In the teachertool tasks were then displayed in the order in which they had arrived. We wanted to enable the teacher to fix the task order and therefore adapted our implementation to support this.

7.2.2. Carry Out Exam

In this subsection we discuss conclusions resulting from carrying out the contest. We start by explaining why our mechanism of handing in tasks proved to be pupil-friendly enough. Afterwards, we show that our backup mechanism proved very useful. Additionally, we discuss the concept of teacher confirmation we introduced in section 4.2.2 and motivate the changes with regard to course management which we implemented after the contest. Then we elaborate on issues arising from missing browser compatibility. We conclude this subsection by reflecting on the overall success of carrying out the contest.

**Handing In Tasks** Before the contest we were concerned that our mechanism to hand in solutions could be overly complicated. To avoid the situation that pupils forget to hand in their solutions, we reminded them to do this at the end of the contest. However, it turned out that almost all of them had already done this before. Visualizing handing in of tasks with colors turned out to be a good reminder. We conclude that the way we implemented task handing in is convenient for pupils.
7. Evaluation

**Backup Mechanism**  Our backup mechanism was used multiple times during the contest. No pupil lost more than a few seconds of working progress during the contest. Pupils accidentally left the exam environment more frequently than we had anticipated. Using appropriate countermeasures, we think that it will be possible to reduce the number of times the exam is left. One possibility would be to require a confirmation before closing or refreshing XLogoOnline during an exam. We leave this improvement to future work.

**Teacher Confirmation**  The mechanism pupils currently use to enroll into a course was only implemented after the contest. The main motivation was that the contest showed that our old mechanism was inconvenient. In our old mechanism pupils were required to explicitly confirm a teacher¹ and the teacher had to add pupils to the course manually. We had pupils register one week in advance in order to prepare the course. When we wanted to start the contest only twelve of the nineteen pupils remembered their password. The remaining pupils had to set up a new account. Consequently, we had to manually add all pupils with a new account to the course again. Using the new mechanism this problem is solved, because pupils can enroll into a course *without* the need for further actions by the teacher. Therefore, it is not an issue anymore if a pupil forgets the password of his normal account.

**Browser Compatibility**  We found several little bugs in our implementation. Most of them were related to browser compatibility issues. The most severe example was that XLogoOnline did not display a task’s description image when running in Microsoft Edge. The reason was that the constructor of the TypeScript class `File` which XLogoOnline uses for displaying an image was not supported by Microsoft Edge² [12]. This problem affected five pupils. To ensure that they also have access to the tasks’ description images, we painted them on the blackboard.

We were able to carry out the contest in the way we planned to and did not find general limitations. This shows that our mechanisms for distributing exams and collecting all pupils’ solutions work. It also indicates that our system is ready to be used in its target environment. Therefore, we generally consider the day of the contest to be a success.

### 7.2.3. Correction

In this subsection we elaborate on our experiences while correcting the pupils’ solutions for the contest. In this process we emphasize challenges we faced and how we improved our automatic correction to overcome these challenges.

¹When *Tom* added *Paola* to a course for the first time, *Paola* was asked if she knew *Tom* and she had to explicitly confirm this.

²We leave fixing this issue to future work.
7.2. Functional Evaluation

This picture displays the image pupils were requested to draw in the contest’s second task.

**Rounding Errors**

Our system rejected almost all solutions to task 2, which requested pupils to draw the image shown in figure 7.2. In the following, we will explain why this issue occurred.

**Image Width**  As we explained in section 6.4, in our image comparison algorithm we first check that the two compared images have the same dimensions. This check was executed on the SVGImage. In order to perform this check, we first calculated an image’s width $w$. For this purpose, we defined $x_{\text{max}}$ to be the largest x-coordinate occurring in any of the images strokes and $x_{\text{min}}$ the smallest x-coordinate occurring in any of the images strokes. We then calculated $w$ as:

$$w = x_{\text{max}} - x_{\text{min}}$$

**Image Size Comparison**  In order to compare the sizes of two images $\text{image1}$ and $\text{image2}$, we first compared their corresponding widths $w_{\text{image1}}$ and $w_{\text{image2}}$. In this comparison we did not require an exact match, but allowed a difference of 0.1 percent of a pixel’s size. The problem was that even such a small error could result in a different number of pixels for $\text{image1}$ and $\text{image2}$ in the resulting pixel matrices. For example if $w_{\text{image1}} = 10.0001$ pixels and $w_{\text{image2}} = 9.9999$ pixels, then our method for comparing the images’ sizes returned that the two images have the same width. But our image comparison algorithm assumes that the two compared images’ sizes match in the pixel matrix and this is not the case here, as $\text{image1}$’s pixelmatrix is 10 pixels wide and $\text{image2}$’s pixelmatrix is 9 pixels wide.

**Image Comparison**  Since we employ image scaling before image comparison, our image comparison method can assume that the two compared images have the same size. Therefore, our image comparison method considers images with different numbers of pixels generally as non-matching. Thus, to fix the issue we had to get rid of the rounding errors initially.

**Our Solution**  We solved the issue by comparing the image dimensions not on the SVGImage as before but on the corresponding pixel matrices.
7. Evaluation

Figure 7.3: This figure displays the problem of the added pen width at the head and tail of a stroke. The red bar is a stroke width 3 and the black dots are this stroke’s endpoints. Obviously the stroke is longer than the distance of its endpoints.

Multiple colors

Our image comparison algorithm worked well for images that only contain black strokes. With multiple colors our algorithm provided too many false negatives, that is it rejected too many pupils’ solutions which we considered as correct in the manual correction. In the following, we will explain why this issue happened.

Overlapping Pixels The reason for this issue was that two strokes that share an endpoint overlap in one pixel. If the two strokes have different colors, then p’s color corresponds to the color of the more recently drawn stroke. If the pupil and the teacher choose a different stroke order, this results in one non-matching pixel. According to the 1-Neighbourhood policy introduced in section 6.4, this should not be a problem as there is always a pixel with the correct color in a 1-Neighbourhood.

Pen Width The aspect of overlapping pixels only becomes a problem when we deal with pen widths that are larger than the default, as we do in our algorithm. The reason is that the additional pen width used in our algorithm also affects a stroke’s length. In particular, a stroke of length 100 and pen width three actually has length 102 in the resulting image. But still the strokes endpoints have a distance of 100. The length addition is added beyond the endpoints. In figure 7.3 we visualize this problem.

Too Large Overlap Because a strokes pen width also affects its length, strokes with pen width three that share one endpoint have an overlap of $9 = 3 \cdot 3$ pixels in the resulting image. This makes our image comparison algorithm fail, as we demonstrate in figure 7.5.

Our Solution We fixed this issue by computing the number of coordinates in which strokes of different colors overlap. This number corresponds to the number of errors our image comparison method tolerates before rejecting a pupils solution.

Our Solution’s Limitation Our way to fix the issue has the disadvantage that it considers a fixed number of errors instead of particular error occurrences. This means that our algorithm only considers how many pixels are allowed to be wrong, but not which pixels. To explain this, we consider Task 6 of the contest, which requested pupils to draw the image shown in figure 7.4.
7.2. Functional Evaluation

Figure 7.4: This picture displays the image pupils were requested to draw in the contest’s sixth task.

For this image, in our solution we calculate that there are 8 points in which strokes of different colors overlap. Therefore, our image comparison algorithm allows 8 wrong pixels. However, these 8 errors are allowed to occur anywhere in the image. They are not required to be in the points where strokes of different colors overlap. Therefore, if Paola hands in exactly the same program as Tom used in the master solution, she can draw a line of length 8 anywhere in the image and her program will still be considered as correct. This problem gets even worse if we consider an image with for example 100 points in which strokes of different colors overlap. This problem could be solved by not only considering how many pixels are allowed to be wrong, but which pixels. We leave this improvement to future work.

Focus on Avoiding False Negatives We implemented our solution in the teachertool, although we known about its limitation. Our motivation to do this is that we focus on avoiding false negatives. It is worse if the teachertool rejects a correct program than if it accepts a wrong program.
7. Evaluation

(a) This listing shows the LOGO program `image` and its subprograms used to draw the image shown in figure 7.5b.

```logo
1 to teacherline
2 setpc red fd 100
3 setpc blue fd 100
4 end
5
6 to pupilline
7 setpc green bk 100
8 setpc yellow bk 100
9 end
10
11 to image
12 rt 90
13 setpw(3) teacherline
14 setpw(1) pupilline
15 end
```

(b) This figure shows different phases of the LOGO program `image`’s execution (`image` is defined in the listing in figure 7.5a). The top image shows the canvas after the execution of line 15. The middle image shows the full picture drawn by `image`. The bottom image shows a close-up of the area where our algorithm finds an error. The problematic pixel $p$ is the one contained in the black square.

**Figure 7.5:** This figure demonstrates the problem of overlapping strokes, which have different colors. We executed `teacherline` with pen width 3 and drew `pupilline` with pen width 1 upon the result, as our algorithm does when comparing the two programs. For better visualization, we used the colors red and blue in `teacherline` and the colors yellow and green in `pupilline`. Our algorithm would return that `pupilline` contains no wrong strokes, if there is a red pixel in every yellow pixel's 1-Neighbourhood and a blue pixel in every green pixel’s 1-Neighbourhood. However, for pixel $p$ this is not true, since $p$ is yellow and its 1-Neighbourhood contains no red pixel. The reason is that the red pixels in $p$’s 1-Neighbourhood are covered by blue or yellow pixels. This is due to the fact that `teacherline` is executed with pen width 3 and therefore the overlap of the red and the blue stroke is too large.
7.2. Functional Evaluation

Figure 7.6: These figures display an example of the problem arising from white strokes in image clipping. To visualize the problem, we set the background color to gray. The red border denotes the clipped image. Although the images shown in figure 7.6a and in figure 7.6b appear to be the same on a white background, the clipped images have different sizes because of the white stroke in figure 7.6b. Our automatic correction thus first scales the image in figure 7.6a to the version shown in figure 7.6c. Then the image in figure 7.6b and the image in figure 7.6c are compared and they do not look similar anymore on a white background.

White Strokes

The biggest remaining issue arises from the usage of white strokes. White strokes on a white background are invisible to the human eye, but they are still there. Therefore, they are considered in image clipping, which we do on the SVGImage. In figure 7.6 we present an example for this problem. The obvious solution seems to be to exclude white strokes in image clipping. However, this alone does not solve the problem since for example a black stroke might be covered by a white stroke. In this case we have to exclude the covered black stroke too, otherwise we modify the resulting image. The issue is complicated even further by the fact that a black stroke must not share and endpoint with the white stroke covering it. Therefore, the approach of excluding white strokes might be difficult to implement working on the SVGImage. A more promising approach would be to clip an image using its pixel matrix and not the SVGImage. We leave this improvement to future work.

Teacher Control

Leaving teachers the possibility to override our system’s choice turned out to be a very good decision. Among the pupils’ solutions were multiple programs that our system correctly rejects although the pupil came close to a correct solution. Because a teacher can override the system’s decision, he could reward the pupil with partial points in such cases. A good example for such a scenario is presented in figure 7.7.

---

3For simplicity we assume in this discussion that the canvas’ background is white. Since XLogoOnline allows users to change the background color of the canvas, this assumption can not be made when implementing a fix to the described issue.

4Black strokes below white strokes are quite common. The number one motivation for pupils to use white strokes is to cover wrong black strokes.
Figure 7.7: This figure displays a screenshot taken during correcting the contest in the teachertool. At the top there is a pupil’s solution and below the corresponding master solution. The difference between the two images is visible by eye as the pupil’s image is rotated some degrees to the right compared to the master solution. Therefore, our system’s choice to reject the pupil’s solution is correct. All the same the pupil demonstrated almost all skills required to solve the task and the teacher might want to acknowledge this. The teacher can do this by overwriting the number of awarded points or even the decision about the correctness.
### 7.3. Research Potential of Backup Mechanism

**Successful Correction**

With the improvements described in this section, we were able to correct the complete contest in the teachertool. For the most part, the teachertool corrected exercise as we would manually. The only exceptions were due to the described limitation concerning white strokes. For those examples we manually overruled the teachertool’s correction. This way we could ensure that the contest’s performance overview presented by the teachertool correctly mirrors the real performance of the pupils. We conclude that our semi-automatic correction meets the requirements we imposed on it.

We arrived at the end of our functional evaluation and now turn towards another aspect: In the following section we discuss how our system’s backup mechanism could be used in research.

#### 7.3. Research Potential of Backup Mechanism

In this section we show that our system’s backup mechanism has a great potential to be used for examining interesting research questions. One such question is whether there are general differences in how boys and girls approach programming tasks. We present metrics that can be used to approach this question and show how we can extract the corresponding data out of our backups.

**Backup Granularity**  
Our system’s backup mechanism can be used to get interesting insights about the process of programming. Upon every action of a pupil during an exam XLogoOnline checks if the backup on the server is recent enough or if a new backup has to be saved. The time difference $\delta$ between consecutive backups is configurable in XLogoOnline. This enables us to get very fine-grained insights. If we set $\delta = 0$, we could even use the backups to create a video showing a pupil’s screen during an exam.

**Automated Backup Evaluation**  
If we configure our backup mechanism in an infinitely fine-grained way, we get the same information as if we recorded a pupil’s screen. The advantage of our backups over such a recording is that our backups have a format which allows automated evaluation. Backup states are numbered increasingly and saved in JSON format on the server as shown in listing 6.2.

**Differences Between Boys and Girls**  
The research question we are going to use as an example in this section considers differences in how boys and girls approach programming tasks. The hypothesis we consider is that boys employ a more exploratory approach while girls rely more on planning. This hypothesis was advanced by Prof. Juraj Hromkovič and his research group independent of our project. In the context of LOGO programming, the hypothesis would suggest that boys are less reluctant to try new approaches to draw a figure. They care less if such a new approach fails. Girls, on the other hand, carefully plan how they want to achieve a certain drawing before starting to code.
7. Evaluation

<table>
<thead>
<tr>
<th>Metadata</th>
<th>creation_time</th>
<th>creation_time</th>
<th>creation_time</th>
</tr>
</thead>
<tbody>
<tr>
<td>- id</td>
<td>- id</td>
<td>- id</td>
<td></td>
</tr>
<tr>
<td>- tasks:</td>
<td>- tasks:</td>
<td>- tasks:</td>
<td></td>
</tr>
<tr>
<td>- task1:</td>
<td>- task1:</td>
<td>- task1:</td>
<td></td>
</tr>
<tr>
<td>- editor</td>
<td>- editor</td>
<td>- editor</td>
<td></td>
</tr>
<tr>
<td>- history</td>
<td>- history</td>
<td>- history</td>
<td></td>
</tr>
<tr>
<td>- task2</td>
<td>- task2</td>
<td>- task2</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>Content</td>
<td>backup_1</td>
<td>backup_2</td>
<td>backup_3</td>
</tr>
</tbody>
</table>

**Figure 7.8.** In this figure we present a schematic visualization of a backup list. Each backup in the list corresponds to a backup file. A backup’s content is in the format as shown in listing 6.2. The file metadata additionally contains the corresponding file’s creation time. All backups in a backup list correspond to the same pupil and backup \( i \)’s creation time is later than backup \( i - 1 \)’s creation time.

**Previous Work** Weibel also considered the same hypothesis in his Bachelor Thesis [55]. We will build upon his findings throughout this section. In particular, we adapted his set of metrics used to approach the hypothesis.

**Usage of Contest Data** For all metrics we are going to present in this section we evaluated the data of our contest. Since only nineteen pupils took part in this context the results we present have no statistical relevance. Our intention is to provide a proof of concept that shows that we can extract the corresponding data from our backups.

7.3.1. CS-Score

In this subsection we present the metric CS-Score and show why it can be interesting with regard to our hypothesis. We then explain how the CS-Score can be extracted from our backups. Finally, we present a listing containing a python method to extract a task’s CS-Score for a pupil.

**Motivation** Our hypothesis states that girls plan more carefully when approaching a LOGO task. This suggests that girls arrive less often in a situation where they have to start over again. In the context of LOGO, starting over again is achieved by using the command \( \text{cs} \). This command clears the screen and resets all parameters to the default value. For example the pen color is set to black and the background color to white.

**Definition** A task’s CS-Score is defined per task as the number of times a pupil used the command \( \text{cs} \). According to the hypothesis we would therefore expect that boys have a higher average CS-Score than girls. CS-Score is a metric that was already presented by Weibel [55].
def compute_cs_score(backups_list, task):
    total_num_cs = 0
    history_length = 0
    old_backup = None
    for backup in backups_list:
        h = get_history_length(backup, task)
        if old_backup is not None:
            if h < history_length:
                # if the current backup’s history length is shorter than the saved
                # history length, then ct was executed between old_backup and backup.
                # Therefore, we add num_cs of old_backup to our total_num_cs
                total_num_cs += get_num_cs(old_backup, task)
            history_length = h
        old_backup = backup
    if old_backup is not None:
        total_num_cs += get_num_cs(old_backup, task)
    return total_num_cs

Listing 7.1: In this listing we present the simplified version of a method to compute a task’s CS-Score for a pupil. We assume that backups_list contains the complete list of backup files associated to the pupil as shown in figure 7.8.

How to Extract CS-Score If we assume that the last backup contains the complete history for every task, the CS-Score is easy to extract. Then we consider the last backup and extract the history of the task we want to consider. This history contains an array of all commands executed by the corresponding pupil as shown in listing 6.2. To compute the CS-Score, we iterate over this array and count how many of its elements contain the command cs.

The Command ct The assumption that the last backup contains the complete history for every task might be wrong. The reason for this is the LOGO command ct, which clears the history. If this command has been used by the pupil, then we might miss usages of cs. Therefore, in a rigorous approach it is necessary to iterate over all backups and figure out whether ct has been used. Since ct results in an empty history, ct itself is never contained in the history. Thus, we have to iterate over all backups and for each of them record the length of the history. Whenever the length of the history decreases between consecutive backups, ct must have been used⁵.

Proof of Concept In listing 7.1 we present a python method that computes a task’s CS-Score. We used this method to compute the average CS-Score of all pupils for all tasks in our contest. The results are presented in figure 7.9a.

⁵When ct is execute in XLogoOnline we always save a backup to the server. This ensures that we do not miss a usage of ct. Otherwise if Paola’s backup contained one entry and she executed ct plus one additional command before saving backup_i+1, we would miss a usage of ct.
7. Evaluation

```python
def compute_planning_phase_duration(backups_list, task):
    planning_duration = 0
    old_backup = backups_list[0]
    for backup in backups_list:
        history_length = get_history_length(backup, task)
        editor_content = get_editor_content_length(backup, task)
        if editor_content > 0 || history_length > 0:
            # if we arrive here, we have the first backup containing changes
            # related to the current task. Thus, we compute the time difference
            # to backup’s predecessor and break out of the loop.
            # save_time returns a backup file’s creation time.
            planning_duration = save_time(backup) - save_time(old_backup)
            break
        old_backup = backup
    return planning_duration
```

**Listing 7.2:** In this listing we present the simplified version of a python method to extract a task’s planning phase duration. We assume that `backups_list` contains the complete list of backup files associated to the pupil as shown in figure 7.8.

7.3.2. Planning Phase Duration

In this subsection we discuss the metric “planning phase duration” and focus on why it can be interesting with regard to our hypothesis. After motivating and defining the metric, we explain how a pupil’s planning phase duration for a task can be extracted from our backups. Finally, we provide a listing that contains a python method to extract a pupil’s planning phase duration for a task.

**Motivation** Our hypothesis states that girls plan more carefully when approaching a LOGO task. This suggests that girls in average devote more time to thinking before starting to code than boys. This planning phase is visible in our backups and its duration can be measured. According to the hypothesis we assume that the average planning phase duration is higher for girls than for boys.

**Definition** In order to measure the planning phase duration, we use the fact that backups are only saved when the pupil has performed an action. This means that we can look for the first backup `backup_i` that contains changes related to a particular task `t`. We then consider `backup_i`’s predecessor `backup_{i-1}`. We define the duration of a pupil’s planning phase for task `t` as the time elapsed between saving `backup_{i-1}` and `backup_i`.

**Recognizing Task Switch** To extract a pupil’s planning phase duration for a particular task `t`, we have to determine the first backup that contains changes related to `t`. This means we have to discover the first backup, in which `t` has either a non-empty history or a non-empty editor content.
7.3. Research Potential of Backup Mechanism

Measure Elapsed Time  After we have found the index $i$ of the backup that contains the first change related to a particular task $t$, we compute the time elapsed between saving $backup_{i-1}$ and $backup_i$. For this purpose, we use that a backup file’s metadata contains the corresponding file’s creation time. Therefore, we only have to compute the respective backups’ time difference and find the desired result.

Proof of Concept  In listing 7.2 we present a python method that computes a pupil’s planning phase duration for a task. We used this method to compute the average planning phase duration for all tasks and all pupils in our contest. The results are presented in figure 7.9b.

7.3.3. History Length

In this subsection we present the history length as metric and discuss why it can be interesting with regard to our hypothesis. We first define the metric and relate it to a similar metric used by Weibel. We then explain how the history length can be extracted from our backups. Finally, we provide a listing that contains a python method to extract a pupil’s history length for a task.

Motivation and Definition  Our hypothesis states that boys approach programming tasks in a more exploratory way than girls. This could suggest that boys execute code more often and in smaller chunks than girls. If this is true, it has to be visible in our backups. In particular, we can extract the history length to observe the number of times that code is executed. We define a task’s history length to be the number of elements in the task’s history. According to our hypothesis we suppose that the average history length is higher for boys than for girls.

Comparison to Weibel  Weibel also used a metric called history length in his thesis, but our definition differs from his [55]. Weibel considers the total number of LOGO commands found in the history. For example if the history contains the element $fd 200 rt 90$, Weibel adds 2 to the history length because the element contains two LOGO commands. In the same example we would only add 1 to the total, since we count the number of history elements. Thus, our definition corresponds to the number of times a pupil executes a code sequence, while Weibel’s definition corresponds to the number of instructions that are executed.

Extracting the History Length  If we assume that the last backup contains every task’s complete history, the history length is easy to compute. For every task a backup contains the history, which is an array. This array’s length corresponds to the history length, as we defined it. As we already explained in section 7.3.1, the assumption that the last backup contains every task’s complete history might be wrong. The reason is that a pupil might have cleared his history using the LOGO command $ct$. This issue has to be considered in the same way as for the CS-Score in order to avoid errors in our results.

Proof of Concept  In listing 7.3 we present a python method that computes a task’s history length. We used this method to compute the average history length of all pupils and all tasks in
7. Evaluation

```python
def compute_history_length(backups_list, task):
    total_history_length = 0
    history_length = 0
    old_backup = None
    for backup in backups_list:
        h = get_history_length(backup, task)
        if old_backup is not None:
            if h < history_length:
                # if the current backup’s history length is shorter than the saved
                # history length, then ct was executed between old_backup and backup.
                # therefore we add history_length to our total_history_length
                total_history_length += history_length
                history_length = h
                old_backup = backup
            if old_backup is not None:
                total_history_length += history_length
    return total_history_length
```

**Listing 7.3:** In this listing we provide the simplified version of a python method to extract a task’s history length out of our backups. We assume that `backups_list` contains the complete list of backup files associated to the pupil as shown in figure 7.8.

our contest. The results are presented in figure 7.9c.
7.3. Research Potential of Backup Mechanism

(a) This figure shows the average CS-Score per task of the pupils who took part in the contest. The blue plot shows the average CS-Score of male pupils and the orange plot shows the average CS-Score of female pupils.

(b) This figure shows the average planning phase duration per task of the pupils who took part in the contest. The blue plot shows the average planning phase duration of male pupils and the orange plot shows the average planning phase duration of female pupils.

(c) This figure shows the average history length per task of the pupils who took part in the contest. The blue plot shows the average history length of male pupils and the orange plot shows the average history length of female pupils.

Figure 7.9.: In this figure we present the results of extracting the discussed metrics from the backups we gathered during the contest. In all plots the error bars represent the standard deviation. All results are not statistically relevant.
7. Evaluation
Conclusion

In this chapter we present our conclusion. Additionally, we discuss some limitations of our system and provide ideas how our system could be extended in future work.

8.1. Conclusion

During this project we designed, implemented, and tested an examination system for the web-based programming environment XLogoOnline. Teachers can use our examination system to design exams and distribute them to their pupils, who can solve such an exam directly in XLogoOnline. After the exam has finished a teacher can collect all pupils’ answers and use our system to correct them.

In this section we first summarize our system’s design. Afterwards, we discuss how our system solves our problem statement’s three sub problems: Designing exams, carrying out exams in the pupils’ known environment, and correcting exams. We will now discuss these aspects and our system’s design in more depth.

Design We implemented our system as a distributed system consisting of three cooperating parts: A server, a Pupil-Client, and a Teacher-Client. For this project we focused on XLogoOnline as Pupil-Client and our teachertool as Teacher-Client. The teachertool can be used to create exams, which can then be solved by pupils directly in XLogoOnline. All pupils’ answers are then automatically collected in the teachertool in order to correct the exam. Our server stores resources like exams, tasks, and solutions. Our server is further used for communication between XLogoOnline and the teachertool, as those two can not directly communicate with each other in our model.
8. Conclusion

**Creating Exams** Our system comfortably enables teachers to create typical LOGO programming exams, as we demonstrated in the programming contest described in chapter 7. Our system supports swift and convenient task creation by providing multiple task templates. Further, we prepared our system for the integration of a task pool, which will act as source of example exam tasks for teachers in the future. Additionally, we implemented our examination system in such a way that it can be extended to provide exams for other programming IDEs, as for example WebTigerJython, in the future.

**Carrying Out Exams** Our examination system enables teachers to carry out an exam in XLogoOnline for multiple pupils at the same time. In our programming contest described in chapter 7 we demonstrated that this mechanism works in the desired manner. This way we showed that our examination system is ready to be used in its target environment.

**Correcting Exams** Our examination system supports teachers in correcting a LOGO programming exam. Our system supports comparing a pupil’s LOGO program to the corresponding master solution in order to check that the pupil’s version draws the requested image. A teacher can instruct our system to correct a LOGO exam automatically. Afterwards, the teacher can overwrite every decision taken by the system, if he does not agree with it. In our programming contest described in chapter 7 we were able to correct all pupils’ answers within the teachertool. After the correction, the teachertool successfully displayed a summary of the pupils’ performance. This shows that our semi-automatic correction is ready to be used in its target environment.

**Goals Satisfied** We were able to reach all our goals defined in section 1.3. We designed, implemented, tested, and integrated an examination system to be used with XLogoOnline. Further, we succeeded in solving our problem in such a way that our examination system can easily be extended to support carrying out exams in other programming IDEs. Therefore, we consider our project as a success.

**8.2. Limitations and Implications**

We generally consider our system’s design and implementation as successful, because we were able to achieve all goals we defined for it. All the same our system has a few limitations:

- **Separation of History and Editor**: The first limitation is due to the separation of the editor and the history in XLogoOnline. This can lead to the case that the teacher does not receive all information he requires to correct a pupil’s solution.

- **Offline Mode**: We can not provide offline support for our examination system. This implies that teachers and pupils have to be online when working in our system.

- **Multiple Colors**: Our image comparison algorithm has a limitation when images contain overlapping strokes of different colors. This can lead to the case the our automatic correction marks wrong programs by pupils as correct.
8.2. Limitations and Implications

- **White Strokes**: Our image clipping method has the limitation that it can not recognize white strokes. This can lead to cases where our automatic correction rejects correct programs by pupils.

In the following subsections we discuss these limitations in more details.

### 8.2.1. Separation of History and Editor

In this subsection we explain the limitation arising from the separation of the editor and the history in XLogoOnline. We assume that a pupil can reexecute his history at the time of handing in a task. Most importantly, this means that a pupil must not change the implementation of programs used in his history (after the most recent occurrence of `cs`). If this assumption is violated, then the teacher might consider a pupil’s solution as wrong even if the correct image is visible on the pupil’s canvas at the time of handing in. We will next explain this issue in more detail and provide an example to demonstrate its implication.

**The Assumption**  In section 6.3.2 we explained how a pupil can hand in a task. If Paola hands in `task1`, Tom receives the history and the editor associated to Paola’s solution for `task1`. The purpose of this is that Paola’s solution can be reexecuted in the teachertool. In this functionality there is a hidden assumption: Paola must be able to reexecute her history on an empty canvas and receive the same image as she sees on her canvas at the time of handing in. However, this assumption can be wrong as we demonstrate in the following example.

**Example: How the Assumption Can Fail**  To explain how the considered assumption can fail, we consider `task1` that requires pupils to write a program drawing a square. We then consider a sequence of four events:

1. *Paola* uses the editor to write a program `square`, which correctly draws a square.
2. *Paola* executes `square`, which results in a square drawn on her canvas.
3. *Paola* clears her editor.

At the moment of handing in `task1` there is a square on *Paola’s* canvas and therefore *Paola* has correctly solved `task1`. However, since *Paola* cleared her editor before handing in, she could *not* reexecute her history on an empty canvas and receive the same image she sees at the time of handing in.

**Consequence**  The consequence of the violated assumption in the previous example is that also in the teachertool we can not reexecute *Paola’s* solution. The teachertool considers *Paola’s* history and sees that a program `square` has been executed. But neither `square`’s definition nor its implementation is contained in *Paola’s* editor. Therefore, the teachertool considers `task1` as unsolved by *Paola*. *Tom* will arrive at the same conclusion, since he also just sees *Paola’s* empty editor.
Possible Solution  This limitation’s nature is not theoretical but arises from the way XLogoOnline and the teachertool are implemented. The underlying problem is that the editor and the history are completely independent in XLogoOnline. The history only contains the name of executed programs, but not their implementations. However, to reexecute the history we require the programs’ implementations and not just their names. Currently, we assume that the implementations are contained in the editor, but this might not be true. And even if the editor contains an implementation for a program included in the history, this implementation might have changed since the program was executed. It would be possible to resolve the issue, if the history would not only save the name of every executed program but also the corresponding implementation.

8.2.2. Offline Mode

Since our system heavily relies on communication between different computers we can not provide an offline mode for our examination system. This limitation directly arises from the system requirements and is not a consequence of our particular design. Our system requirements discussed in chapter 3 state that multiple pupils must be able to take the same exam in XLogoOnline at the same time. These pupils operate at different physical computers and therefore we have to synchronize operations between multiple computers. This requires communication between the computers of the teacher and the pupils.

It would have been possible to provide an offline mode, that offers partial functionality. For example we could have implemented our system in a way that allows teachers to correct an exam without being online. However, we decided against providing an offline mode because in order to offer offline support we would have had to choose a different technology. Most importantly, this would have implied that we can not reuse code from XLogoOnline, which we did in our automatic correction. Therefore, we only ensured that no problems arise from a temporarily lost network connection. We achieved this by implementing our backup mechanism, which we introduced in section 6.3.2.

8.2.3. Multiple Colors

In section 7.2.2 we discussed that in the automatic correction an issue arises from points in which strokes of different colors overlap. In this subsection we highlight the limitation arising from our solution to this issue.

Summary of the Issue  If an image drawn by a LOGO program contains points, where strokes of different colors overlap, then the order in which these strokes are drawn is relevant. The pixel in which these two strokes overlap has the color of the more recently drawn stroke. Our image comparison method amplifies this problem because it uses a larger pen width, which enlarges the area in which strokes overlap.
8.2. Limitations and Implications

**Summary of our Solution**  As explained in section 7.2.2, we solve this issue by computing the number of points where strokes of different colors overlap. This corresponds to the number of errors our image comparison method allows when comparing a pupil’s program to the corresponding master solution.

**Our Solution’s Limitation**  We showed that this solution is not optimal because our image comparison method allows these wrong pixels to occur *anywhere* in the image. The underlying problem is that our image comparison algorithm only knows *how many*, and not *which*, pixels are allowed to be wrong.

**Consequence**  Consequently, our image comparison method might accept pupils’ programs which are wrong. For example if an image contains 100 points in which strokes of different colors overlap, then our image comparison method allows 100 errors. But if the pupil draws all strokes in the same order as the corresponding master solution, these 100 error can occur at another place in the image. For example the pupil could then add an additional stroke of length 100 to his image and our image comparison method would still accept his solution.

**Possible Solution**  A solution to this limitation would be to adapt our image comparison method such that it not only considers *how many* pixels are allowed to be wrong. Instead it should consider *which* pixels are allowed to be wrong.

8.2.4. White Strokes

In section 7.2.3 we explained that our automatic correction currently can not appropriately handle images that contain white strokes. In this subsection we highlight the limitation arising from this.

**Summary of the Issue**  Our image comparison method handles images that contain white strokes incorrectly in some cases. The reason is that white strokes are not visible on a white background, but are considered in image clipping. Image clipping denotes the process of finding the smallest rectangle \( r \) such that all strokes of an image are completely contained in \( r \). Instead of this, image clipping should find the smallest rectangle in which all *visible* strokes are contained.

**Consequence**  Before comparing two images, our image comparison method ensures that the two considered images have the same size. If a pupil’s solution is correct, then his image should have the same size as the corresponding master solution after clipping. However, if the pupil used white strokes in his LOGO program in contrast to the master solution, then the two clipped images might have different sizes. If this is the case, the pupil’s solution is rejected, even if the two images look the same.
8. Conclusion

Possible Solution This limitation arises from the fact that we perform image clipping on the SVGImage, as explained in section 6.4.2. The limitation could be resolved by adapting image clipping to operate on the pixel matrix instead of the SVGImage.

8.3. Future Work

In this section we present ideas how our system could be extended in future work. We start with improvements to make our system even better in the current setup. To conclude the section, we also present extension ideas that would adapt our system to a bigger context.

8.3.1. Improvements of Current System

In this subsection we present ideas how our system could be further improved in the current context. We dedicate one paragraph to each improvement idea.

Task Pool We built our system with the idea of a shared task pool in mind. This pool should be accessible by all teachers and contain tasks that can be used in exams. We did not implement this task pool as part of our project, but we prepared the system such that it can easily be introduced. If we were to continue with our project, the task pool would be the next feature to introduce.

Reusing Tasks in Multiple Exams An exam contains a set of tasks, which pupils have to solve. Currently, those tasks have to be created at the same time as the exam itself is created. However, it would be better if our system enabled to include already existing tasks into a new exam. This feature is intertwined with the introduction of a task pool and would improve our system.

Support For White Strokes In section 7.2.3 we explained that our automatic correction currently can not appropriately handle images that contain white strokes. This limitation could be resolved in future work by adapting image clipping to operate on the pixel matrix instead of the SVGImage.

Leaving ExamMode During the programming contest we described in chapter 7 we realized that pupils accidentally leave the ExamMode more often than we anticipated. Therefore, our system would benefit from mechanisms that prevent pupils from closing or reloading XLogoOnline without an additional confirmation.

Stroke Width Our image comparison mechanism assumes that pupils execute their programs with pen width one. Since pupils rarely know that the pen width is configurable\(^1\), this assump-

\(^1\)This functionality is not included in the teaching materials provided by ABZ.
tion is legitimate. Still it would be better to get rid of the assumption. This would be possible by considering the pen width chosen by the pupil as thin in our image comparison. The pen width corresponding to thick execution in our algorithm could be achieved by adding 2 to the pupil’s chosen pen width.

**Early Handing In of Exams**  XLogoOnline does not explicitly provide the feature that pupils can hand in their exam before the teacher stops it. Implicitly this feature is supported, since pupils could just close the browser in which XLogoOnline is executed. However, in future work early handing in of exams could be supported explicitly to provide a clean solution for this issue.

**Automatic Handing In of Tasks**  In section 6.3.2 we discussed why pupils have to hand in their answers manually. The disadvantage of this feature is that pupils could forget to hand in a task, which is equivalent to not solving a task. In future work our system could be improved as follows: When an exam is stopped, our system could automatically hand in all tasks that were never handed in during the exam. This feature can not be implemented in XLogoOnline, since XLogoOnline is only informed after an exam is stopped and not before it is stopped. However, the feature could be implemented in the server. The server could extract the required information from a pupil’s most recent backup and include it in the pupil’s solution.

**Overlapping Strokes**  In section 7.2.2 we explained the issue arising from points in which strokes of different colors overlap. We also explained that we solved this issue by computing the number of such points and allowing the corresponding number of wrong pixels. We then showed that this solution is not optimal because these wrong pixels can occur anywhere in an image. The underlying problem is that our image comparison algorithm only knows how many pixels are allowed to be wrong. In future work, our algorithm could be improved such that it also considers which pixels are allowed to be wrong.

## 8.3.2. Extensions to Our System

**Automatic Exam Generation**  In section 2.4 we discussed learning taxonomies. The intention behind this was to work towards automatic exam generation. We did not yet implement automatic exam generation as part of our project. However, we still like the idea of improving our system to support automatic exam generation. For this purpose, we intended to categorize tasks based on criteria derived from the considered learning taxonomies. We could then randomly select a task of each category to be included in a newly generated exam. A teacher could then request an automatically generated exam and adapt it, such that it meets his expectations.

**Homework Mode**  Our system could be extended to support homework assignments. Implicitly homework assignments are already supported, since an exam lasts until the teacher stops it. Therefore, a homework assignment could be seen as exam that lasts for a week. However,
8. Conclusion

for homework it could be beneficial if pupils had XLogoOnline’s full functionality available\(^2\). We already prepared support for a *HomeworkMode* similar to the *ExamMode* on the server. Providing this support also in XLogoOnline would improve our system.

**Observing Performance Across Multiple Exams** Our system’s model allows to compare a pupil’s performance across multiple exams in the teachertool. However, this functionality is not implemented yet and could improve our system.

**Adding Pupil-Clients** We heavily emphasized modularity and generality when designing our model. The intention of this was that it should be as easy as possible to add additional Pupil-Clients to the system. With this effort we mainly targeted the web-based IDE WebTigerJython that is currently being developed at our chair. Neither the teachertool nor the server would require changes in their implementation in order to carry out an exam in a Pupil-Client different than XLogoOnline. Only the additional Pupil-Client would have to be prepared for carrying out an exam. This could be done in similar way as we did it for XLogoOnline. Also the teachertool’s semi-automatic correction framework would not have to be changed in order to support an additional Pupil-Client. We would only have to *extend* the teachertool with a module taking care of the automatic correction of exams for the new Pupil-Client.

**MOOC** In section 2.5 we talked about the MOOC learning model. We believe that this model has potential to be used in main school programming education. Since our tool can automatically correct LOGO tasks, it would be possible to get the teacher out of the loop\(^3\), particularly if our system also supported automatic exam generation. This would enable us to provide MOOC-like learning courses in our system. This idea is meant as an alternative to the current setup with a teacher and pupils and not as a replacement. For this purpose, also elements of the system PAT introduced in section 2.4.3 could be used [25].

\(^2\)In the *ExamMode* some features like loading LOGO programs from disk are locked for pupils.

\(^3\)For example Weibel’s judge for XLogoOnline did not rely on the existence of a teacher [55].
Questionnaire

In this section we present the version of our questionnaire distributed to teachers to be filled out.
Einleitung

Use Case


Durchführung  Sobald die Schüler Platz genommen haben, öffnen sie XLogoOnline, melden sich mit ihrem Login an und öffnen in XLogoOnline per Knopfdruck den integrierten Prüfungsmodus. Darauf werden sie gebeten, den Prüfungscode abzuschreiben, den die Lehrperson zuvor an die Tafel geschrieben hat. Anschliessend meldet XLogoOnline, dass es für die Prüfung bereit ist. Sobald alle Schüler bereit sind, startet die Lehrperson die Prüfung, indem sie in der Lehrerumgebung den zugehörigen Knopf drückt. Dadurch erscheinen die Aufgaben im XLogoOnline bei den Schülern.

Die Schüler können nun die Aufgaben lösen. Dabei können sie beliebig zwischen den Aufgaben hin und her navigieren. Jedes Mal wenn sie eine früher begonnene Aufgabe erneut öffnen, können sie dort weitermachen, wo sie zuletzt aufgehört haben. Dabei wird der Fortschritt der Schüler einmal pro Minute automatisch gespeichert und an die Lehrperson gesendet. Kurz vor Ablauf der Zeit erhalten die Schüler eine Meldung mit der restlichen Zeit und sobald die Zeit abgelaufen ist, wird der aktuellste Stand der Schüler gespeichert und an die Lehrerumgebung gesendet. Die Schülerbildschirme informieren über das Ende der Prüfung und zeigen danach wieder die Startseite von XLogoOnline an.


Figure 1: Schematische Darstellung des Systems

A. Questionnaire
automatischen Bewertung und der manuell von der Lehrperson durchgeführten Korrektur berechnet die Lehrerumgebung pro Schüler eine Gesamtpunktzahl. Anhand dieser kann die Lehrperson frei eine Note verfügen.

**Fragebogen**

**Pädagogik**

3. Würden Sie es begrüßen, wenn Sie mit der Lehrerumgebung auch Prüfungsreihen erstellen könnten?

Ja: Nein: Egal: Ich verstehe die Frage nicht:

4. Sollte das System die Lehrperson warnen, wenn der Verdacht auf Betrug besteht (z.B. abschreiben von auf dem Desktop gespeicherten Programmen)?

Ja: Nein: Egal: Ich verstehe die Frage nicht:

Konfigurierbarkeit

5. Um Prüfungen zu gestalten erhalten Sie Zugriff auf eine Aufgabensammlung der ABZ. Wie wichtig ist es Ihnen zusätzlich eigene Fragen verwenden zu können?

Unwichtig: Eher unwichtig: Eher wichtig: Wichtig:
6. Angenommen Sie wollen eigene Fragen für die Prüfung verwenden. Sollen diese Fragen künftig in die Aufgabensammlung der ABZ aufgenommen und anderen Lehrpersonen zur Verfügung gestellt werden?

Ja: Nein: Egal: Ich verstehe die Frage nicht:


10. Welche der folgenden beiden Systembewertungen sagt Ihnen mehr zu und weshalb?

A. Im System kann bis ins kleinste Detail alles konfiguriert und personalisiert werden. Allerdings braucht es etwas Einarbeitungszeit, um das Beste aus dem System heraus zu holen.

B. Das System ist sehr intuitiv und man kann sofort damit arbeiten. An einigen Punkten muss aber mit unveränderbaren Voreinstellung vorlieb genommen werden.
Allgemeine Fragen zum Setup

11. Was sind Ihre Erwartungen an ein Prüfungssystem? Was soll es für Sie übernehmen, was nicht?

12. Gibt es Aspekte in der Beschreibung, bei denen Sie bezweifeln, dass sie in der Praxis umsetzbar sind?

13. Gibt es Aspekte in der Beschreibung, mit denen Sie nicht einverstanden sind?
14. Würden Sie so ein System benutzen? Wenn nein, wieso nicht?

Weiteres

16. Würden Sie die Prüfung lieber komplett selbst korrigieren?

   Ja:    Nein:    Egal:    Ich verstehe die Frage nicht:
17. Haben Sie Bedenken betreffend des Datenschutzes? Wenn ja, welche?

18. Würden Sie es alternativ zum Login bevorzugen, wenn die Schüler zu Beginn der Prüfung nur ihren Namen eingeben müssten? Vorteil: Die Schüler müssen sich kein Passwort merken. Nachteil: Das Login könnte auch für andere Zwecke verwendet werden, zum Beispiel um Programme der Schüler online zu speichern, sodass sie als Hausaufgabe daran weiterarbeiten könnten.

   Ja:  Nein:  Egal:  Ich verstehe die Frage nicht:

19. Soll das System alternativ zu Punkten eine andere Feedbackmethode verwenden? Wenn ja, was für eine?
A. Questionnaire

20. Was möchten Sie noch sagen?
Contests

B.1. Contest Used to Test Examination System

In this section we present the contest we used during our system’s evaluation.
**Contest**

1. Zeichne mit einem Programm das folgende Bild:

```
repeat 4 [ bk 100 fd 200 rt 90 ]
```
2. Zeichne mit einem Programm das folgende Bild:

```
repeat 3 [ rt 90 fd 200 repeat 5 [ fd 200 lt 90 ] ]
```

3. Zeichne mit einem Programm das folgende Bild:

```
repeat 5 [ repeat 6 [ fd 200 rt 90 ] rt 180 ]
```
4. Das Programm `HALBKREISRECHTS HALBKREISLINKS HALBKREISRECHTS` soll die folgende Figur zeichnen:

Schreibe die Unterprogramme `HALBKREISRECHTS` und `HALBKREISLINKS`, so dass das oben angegebene Programm richtig läuft.

```
sol

to HALBKREISRECHTS
  repeat 45 [ fd 8 rt 4 ]
end

to HALBKREISLINKS
  repeat 45 [ fd 8 lt 4 ]
end

to sol
  HALBKREISRECHTS HALBKREISLINKS HALBKREISRECHTS
end
```

5. Vervollständige das Programm `AUFGABE5` so, dass es das obige Bild zeichnet:

```
to AUFGABE5
  repeat 9 [ repeat 7 [ fd 150 rt 90 ] ]
end
```

```
to AUFGABE5
  repeat 9 [ repeat 7 [ fd 150 rt 90 ] rt 90 ]
end
```

```
STERN

to STRAHL
  setpc 4 fd 100 setpc 1 fd 200 pu bk 300 pd
end

to STERN
  repeat 8 [STRAHL rt 45]
end
```

MUSTER 200

to SECHSECK :GR
repeat 6 [fd :GR rt 60]
end
to MUSTER :GR
repeat 3 [SECHSECK :GR rt 120] ht
end
In this section we present several examples of contests used in previous LOGO programming courses offered by ABZ.
Du darfst die Aufgaben in beliebiger Reihenfolge lösen. Suche dir zuerst die Aufgaben aus, die dir am meisten Spaß machen.

1. Zeichne das folgende Bild.

2. Zeichne das folgende Kreuz.

3. Schreibe ein Programm für die folgende Figur.


Du darfst die Aufgaben in beliebiger Reihenfolge lösen. Suche dir zuerst die Aufgaben aus, die dir am meisten Spass machen.


```
50 50
```

2. Zeichne mit einem Programm das folgende Kreuz.

```
50 50
```

3. Programmiere die folgende Zeichnung.

```
100 100 20
```

4. Schreibe ein Programm zum Zeichnen des folgenden Ovals.

```

```


```

```


```
:GR
```
Beispiel mit 6 Sechsecken. 
Du darfst die Aufgaben in beliebiger Reihenfolge lösen. Suche dir zuerst die Aufgaben aus, die dir am meisten Spaß machen.

1. Zeichne das folgende Bild.

2. Zeichne das folgende Kreuz.

3. Schreibe ein Programm für die folgende Figur.


Du kannst die Aufgaben in beliebiger Reihenfolge lösen. Beginne mit denjenigen Aufgaben, die dir am meisten Spass machen.
(Die blauen Masseinheiten in den Figuren müssen nicht gezeichnet werden.)

1. Zeichne mit einem Programm das folgende Bild:

```
  200
    100
    100
    100
```


```
80
80
```

(Die blauen Strecken sollen nicht gezeichnet werden.)

```
50
100
```

4. Das Programm

```
HALBKREISRECHTS
HALBKREISLINKS
HALBKREISRECHTS
```

soll die folgende Figur zeichnen:

```
  |   |
---+---+
  |   |
```

Schreibe die Unterprogramme `HALBKREISRECHTS` und `HALBKREISLINKS`, so dass das oben angegebene Programm richtig läuft.
(Die Grösse der Halbkreise kannst du wählen.)

5. Vervollständige in beiden Teilaufgaben das Programm so, dass es das obige Bild zeichnet:

```
a) repeat 9 [ repeat 7 [fd 40 rt 90] ... ]
b) fd 40 rt 90 fd 360 rt 90 fd 40 rt 90 fd 40 repeat 8 [rt 90 fd 40 ... ]
```

6. a) Schreibe ein Programm `SECHSECK:GR`, das ein regelmässiges Sechseck mit frei wählbarer Seitenlänge `:GR` zeichnet.

```
:GR
:GR
```

b) Verwende `SECHSECK:GR` als Unterprogramm, um ein Programm `MUSTER:GR` zu schreiben, welches das folgende Bild ergibt:

```
:GR
```

Zusatzaufgabe

7. Schreibe ein Programm, das die folgende Figur zeichnet:

```
  |   |
---+---+
  |   |
```

Die Grösse der Quadrate und die Anzahl der schwarzen Quadrate sollen frei wählbar sein. Rote Quadrate hat es immer gleich viele wie schwarze.
B. Contests
In this section we provide a cheat sheet for XLogoOnline, which is available at https://xlogo.inf.ethz.ch/Cheatsheet.pdf. It also contains an introduction of the most important LOGO commands.
XLogoOnline – 
Befehlsübersicht

In diesem Dokument werden alle Logo-Befehle aufgelistet, welche in XLogoOnline verfügbar sind. Beachten Sie, dass nicht alle davon für den Unterricht an Primarschulen geeignet und notwendig sind.

Grundbefehle

FD zahl  Bewege die Schildkröte eine gewisse Anzahl von Schritten vorwärts
BK zahl  Bewege die Schildkröte eine gewisse Anzahl von Schritten rückwärts
RT winkel  Drehe die Schildkröte an der Stelle nach rechts um einen gegebenen Winkel.
LT winkel  Drehe die Schildkröte an der Stelle nach links um einen gegebenen Winkel.
CS  Lösche die Zeichnung und setze die Schildkröte zurück ins Zentrum

fd 100 rt 90 bk 200 lt 360/4 fd 125.5 cs

Arbeiten mit Farben

SETPC farbe  Die Stiftfarbe wird auf die gegebene Farbe festgelegt.
SETCSC farbe  Die Hintergrundfarbe wird auf die gegebene Farbe festgelegt.

Benutzt werden die folgenden Farb-Wörter, Zahlen oder Farbwerte \([R \ G \ B]\) der additiven Farbmischung:

<table>
<thead>
<tr>
<th>Name</th>
<th>Zahl</th>
<th>[Rot Grün Blau]</th>
</tr>
</thead>
<tbody>
<tr>
<td>black</td>
<td>0</td>
<td>[0 0 0]</td>
</tr>
<tr>
<td>red</td>
<td>1</td>
<td>[255 0 0]</td>
</tr>
<tr>
<td>green</td>
<td>2</td>
<td>[0 255 0]</td>
</tr>
<tr>
<td>yellow</td>
<td>3</td>
<td>[255 255 0]</td>
</tr>
<tr>
<td>blue</td>
<td>4</td>
<td>[0 0 255]</td>
</tr>
<tr>
<td>magenta</td>
<td>5</td>
<td>[255 0 255]</td>
</tr>
<tr>
<td>cyan</td>
<td>6</td>
<td>[0 255 255]</td>
</tr>
<tr>
<td>white</td>
<td>7</td>
<td>[255 255 255]</td>
</tr>
<tr>
<td>darkgray</td>
<td>8</td>
<td>[128 128 128]</td>
</tr>
<tr>
<td>lightgray</td>
<td>9</td>
<td>[192 192 192]</td>
</tr>
<tr>
<td>darkred</td>
<td>10</td>
<td>[128 0 0]</td>
</tr>
<tr>
<td>darkgreen</td>
<td>11</td>
<td>[0 128 0]</td>
</tr>
<tr>
<td>darkblue</td>
<td>12</td>
<td>[0 0 128]</td>
</tr>
<tr>
<td>orange</td>
<td>13</td>
<td>[255 200 0]</td>
</tr>
<tr>
<td>pink</td>
<td>14</td>
<td>[255 175 175]</td>
</tr>
<tr>
<td>purple</td>
<td>15</td>
<td>[128 0 255]</td>
</tr>
<tr>
<td>brown</td>
<td>16</td>
<td>[153 102 0]</td>
</tr>
</tbody>
</table>

setpc red fd 200 setsc [128 0 255]

Kommentare

# text  Zeilen, welche mit einem Hashtag (#) beginnen, werden als Kommentar verstanden und nicht ausgeführt. Nur im Editor (links) sind Kommentare erlaubt.

fd 100 rt 120
# fd 100 rt 120 --> wird nicht ausgeführt
fd 100 rt 120

Stift-Manipulation

SETPW zahl  Setze Stiftbreite auf gegebenen Wert. Standard ist 1 (Achtung: intern effektiv 2, da sonst Aliasing-Effekte auftreten)
PU  Hebe den Stift hoch, die Schildkröte fährt, ohne zu zeichnen.
PD  Setze den Stift ab - ab sofort werden wieder Linien gezeichnet.
PE  Radiergummi-Modus: Gezeichnete Linien werden ausradiert (übermale mit Hintergrundfarbe)
PPT  Beendet Radiergummi-Modus
WASH  Lösche alle bisher gezeichneten Linie. Die Hintergrundfarbe bleibt erhalten, ebenso die Position der Schildkröte.

fd 100 rt 90 fd 100 setpc green wash
setpw 5 fd 100 pu fd 100 pd wash

Schildkröten-Manipulation

HT  Die Schildkröte wird versteckt. Sie steht an derselben Stelle und kann zeichnen, allerdings sieht man sie dabei nicht.
ST  Zeige die Schildkröte, falls versteckt.
SETX x  Setze die x-Koordinate der Schildkröte. Ursprung ist jeweils \((0,0)\), im Zentrum
SETY y  Setze die y-Koordinate der Schildkröte
SETXY x y  Setze x- und y-Koordinate der Schildkröte
HOME  Setze die Schildkröte zurück auf \((0,0)\), ohne dabei die gezeichneten Linien zu löschen.

SETHEADING z  Setze die Ausrichtung z der Schildkröte manuell fest.

<table>
<thead>
<tr>
<th>Ausrichtung</th>
<th>Wert</th>
</tr>
</thead>
<tbody>
<tr>
<td>oben</td>
<td>0</td>
</tr>
<tr>
<td>rechts</td>
<td>90</td>
</tr>
<tr>
<td>unten</td>
<td>180</td>
</tr>
<tr>
<td>links</td>
<td>270</td>
</tr>
</tbody>
</table>

ht fd 100 st setx 200 fd 100 home
fd 100 setheading 90 fd 100
Arithmetische Funktionen

**RANDOM max** Generiere eine zufällige natürliche Zahl im Bereich zwischen 0 und max-1.

**MOD a b** Berechne den Rest der Division von a durch b. Bsp: 21 modulo 5 ist 1 denn: 21 = 5*4 + 1.

**POWER a b** Potenziere b mit a: Berechne $b^a$

**SQRT a** Zeie die Wurzel von a: Berechne $\sqrt{a}$

**LOG a** Logarithmus von a zur Basis 10: $\log_{10}(a)$

**ABS a** Berechne den absoluten Wert von a (falls a ein negatives Vorzeichen hat, berechne $a \times (-1)$ um den positiven Wert von a zu erhalten.

**SIN a** Berechne $\sin(a)$

**ARCSIN a** Berechne $\sin^{-1}(a)$

**COS a** Berechne $\cos(a)$

**ARCCOS a** Berechne $\cos^{-1}(a)$

**TAN a** Berechne $\tan(a)$

**ARCTAN a** Berechne $\tan^{-1}(a)$

fd 200 rt 90 fd 200 rt 135 fd sqrt 2*40000

Mathematische Konstanten

**PI** Eine auf Maschinengenauigkeit exakte Darstellung der Zahl pi (3.1415)

**E** Eine auf Maschinengenauigkeit exakte Darstellung der Eulerzahl e (2.7182)

Interaktion mit der History

**PRINT zahl** Schreibe eine Zahl in die History

**PRINT [text]** Schreibe Text in die History

**CT** Leere die History

print [pi hat den Wert:] print pi ct

Neue Befehle definieren


<table>
<thead>
<tr>
<th>Ohne Parameter</th>
<th>Mit Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>TO NAME befehle END</td>
<td>TO NAME :PARAMETER befehle END</td>
</tr>
<tr>
<td>TO QUADSEITE100 fd 100 rt 90 END</td>
<td>TO QUADRAT :GR repeat 4[fd :GR rt 90] END</td>
</tr>
</tbody>
</table>

Kontrollstrukturen

*Ohne die typischen Kontrollstrukturen (Schleifen und bedingte Ausführung)* werden Programme stets Befehl-für-Befehl von oben nach unten ausgeführt. Schleifen und bedingte Ausführung ändern dies.

Schleifen

Mit Schleifen können Befehle mehrmals wiederholt werden. Logo kennt zwei Schleifen-Arten:

- **REPEAT zahl [befehle]** Wiederhole die Befehle in den eckigen Klammern so oft wie die gegebene Zahl es verlangt.
  repeat 4 [fd 100 rt 90]

- **WHILE [abbruchbedingung] [befehle]** Wiederhole die gegebenen Befehle solange die gegebene Bedingung erfüllt ist. Achtung: Programme deren Bedingung immer erfüllt ist, werden in Folge niemals stoppen.
  while [:a>0] [fd 100 rt 90 make "a :a-1]

Bedingte Ausführung

Mit dem Keyword IF können wir selektiv Teile unseres Codes ausführen oder nicht. Es gibt zwei Arten dieser bedingten Ausführung (wenn-dann und wenn-dann-sonst):

- **IF (bedingung) [befehle]** WENN-DANN: Führe die Befehle in Klammern nur dann aus, wenn eine gegebene Bedingung erfüllt ist.
  make "a 1 if[:a=0] [fd 100]

- **IF (bedingung) [befehle]** WENN-DANN-SONST: Wenn die Bedingung erfüllt ist, führe den ersten Code-Block aus, sonst den zweiten.
  make "a 1 if[:a=0] [fd 100] [bk 100]

Ausführungssteuernde Befehle

**STOP** Breche die Ausführung an dieser Stelle ab. Ausserhalb einer Schleife wird das Programm vollständig abgebrochen. *Bug* In verschachtelten Schleifen bricht nur die Schleife ab.

**WAIT zahl** Pausiere die Ausführung. Hier gilt: wait 100 pausiert die Ausführung während 1 Sekunde.

**MAKE "a b** Definiere eine Variable mit dem Namen a und dem Wert b. Diese Variable kann anschließend verwendet werden durch :a. Variablen sind ausschliesslich verfügbar in ihren Definitionsbereichen (dem jeweiligen Befehl)

make "x 100 repeat 50 [fd :x make "x :x+1] repeat 100[fd 99 bk 99 wait 10 wash fd 10]
C. XLogoOnline Cheat Sheet
Bibliography


Bibliography


Bibliography


Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

A Computer-Based Examination System for XLogoOnline

Authored by (in block letters):

For papers written by groups the names of all authors are required.

Name(s):

Eisebach

First name(s):

David

With my signature I confirm that
- I have committed none of the forms of plagiarism described in the 'Citation etiquette' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date

Winterthur, 30.08.2019

Signature(s)

David Eisebach

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.