



INDIVIDUAL WORKSHEETS WITH INTERACTIVE PROGRAMMING EXERCISES WITHIN THE HPI SCHUL-CLOUD

Individuelle Arbeitsblätter mit Interaktiven Programmieraufgaben
im Rahmen der HPI Schul-Cloud

SEBASTIAN SERTH

Sebastian.Serth@student.hpi.de

A Master's Thesis for attainment of the academic degree
Master of Science in IT-Systems Engineering

SUPERVISORS:

Prof. Dr. h.c. Hasso Plattner

Dr. Matthias Uflacker

Ralf Teusner, M.Sc.

Dipl.-Inf. (FH) Jan Renz

CHAIR:

Enterprise Platform and Integration Concepts

Hasso Plattner Institute

University of Potsdam

Potsdam, 6th May 2019

Sebastian Serth:

*Individual Worksheets with Interactive Programming Exercises
within the HPI Schul-Cloud*

*Individuelle Arbeitsblätter mit Interaktiven Programmieraufgaben
im Rahmen der HPI Schul-Cloud*

SUPERVISORS:

Prof. Dr. h.c. Hasso Plattner

Dr. Matthias Uflacker

Ralf Teusner, M.Sc.

Dipl.-Inf. (FH) Jan Renz

CHAIR:

Enterprise Platform and Integration Concepts

Hasso Plattner Institute

University of Potsdam

SUBMITTED ON:

6th May 2019

ABSTRACT

Modern computer science education in high-schools requires students to learn the basics of programming. In terms of content, teachers often incorporate existing videos, quizzes, and practical programming exercises from Massive Open Online Courses (MOOCs). However, teachers' options to adapt the content to their specific needs or to add their own material are currently limited. Based on a qualitative survey with thirteen teachers, we developed tools to extend these options. Our software prototype allows teachers to create their own interactive worksheets consisting of texts, videos, quizzes, and practical programming exercises. Additionally, teachers can embed and further customize existing exercises from MOOCs. Further, we enable teachers to gain deeper insights into the learning progress of their students by providing results from automated submission analysis. These data allow uncovering potential knowledge gaps and foster content-driven in-class discussions. In this thesis, we present findings from an evaluation with different school classes using our software. The concept was well received by students and teachers alike: Teachers noticed the possibility of a shift in their role from a lecturing instructor to an individual tutor, as students are enabled to learn at their own pace and receive specific, direct feedback based on automated unit tests. For the preparation of upcoming lessons, teachers valued the ability to analyze common mistakes of their students to uncover and discuss previously hidden problems. Interactive worksheets, as an integrated part of digital education, thus foster informed teacher interventions as part of an individualized student learning process.

ZUSAMMENFASSUNG

Moderner Informatikunterricht an Gymnasien umfasst das Erlernen von Programmierungsgrundlagen. Je nach Unterrichtsinhalt verwenden Lehrer dabei häufig bereits vorhandene Videos, Quizfragen und praktische Programmieraufgaben aus **Massive Open Online Courses (MOOCs)**. Dabei sind die Möglichkeiten der Lehrer, die Inhalte an ihre spezifischen Bedürfnisse anzupassen oder eigenes Material hinzuzufügen, momentan jedoch begrenzt. Um diese Möglichkeiten zu erweitern, haben wir einen Prototypen entwickelt, der die von 13 Lehrern in einer qualitativen Umfrage geäußerten Anforderungen erfüllt. Unsere Software ermöglicht es Lehrern ihre eigenen interaktiven Arbeitsblätter, bestehend aus Texten, Videos, Quizfragen und praktischen Programmieraufgaben, zu erstellen. Zusätzlich können sie vorhandene Übungen aus **MOOCs** einfügen und an ihre Bedürfnisse anpassen. Darüber hinaus erlangen Lehrer mithilfe von automatisierten Analysen der abgegebenen Programmieraufgaben ihrer Schüler einen erweiterten Einblick in deren Lernfortschritt. Diese Informationen erleichtern es, potentielle Wissenslücken bei Schülern aufzudecken und inhaltsorientierte Diskussionen innerhalb der Klasse anzustoßen. In dieser Arbeit stellen wir die Ergebnisse des Einsatzes unserer Software in unterschiedlichen Schulklassen dar. Das Konzept wurde von Schülern und Lehrern gleichermaßen gut angenommen: Die Lehrer erkannten die Chance, ihre Rolle vom ausschließlichen Dozenten zum individuellen Tutor zu wandeln, während Schüler in ihrem eigenen Tempo lernen können und direkte Rückmeldungen zu ihrem Fortschritt durch automatisierte Auswertungen erhalten. Für die Vorbereitung zukünftiger Unterrichtsstunden schätzten Lehrer die Möglichkeit, häufige Fehler ihrer Schüler auszuwerten, um so zuvor unerkannte Probleme aufzudecken und besprechen zu können. Interaktive Arbeitsblätter fördern individualisierte Lernprozesse, unterstützen Lehrer in der Unterrichtsgestaltung und sind somit ein wichtiger Bestandteil digitaler Bildung an Schulen.

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to everyone who supported me throughout my studies and especially while working on this thesis.

First of all, I want to thank Ralf for his excellent support, the detailed feedback and for all clever solutions he suggested for any unpredictable challenge I faced. Thanks for all the good advice you gave me—I will definitely keep them in mind!

Thanks a lot also to Jan, who regularly provided me with new ideas to extend my concept and offered me unique opportunities to gather feedback and test my prototype. Moreover, I want to thank the members of the openHPI and Schul-Cloud teams for their constant help. And, of course, Paul for his extraordinary assistance with edtr.io.

My special thanks go to all teachers and students involved as well as the members of the EPIC research group; be it interviewing, brainstorming, discussing, testing or evaluating the concept. A big thank to Kai, Marius, and Whitney for the last minute proofreading and the support in fine-tuning this thesis.

Last but not least, I want to thank my friends for a great time, and especially my family for their full support.

Thank you!

CONTENTS

1	INTRODUCTION	1
1.1	Motivation	1
1.2	Research Questions	4
1.3	Structure	4
2	BACKGROUND	7
2.1	Massive Open Online Courses	7
2.2	MOOCs Offered by the HPI with openHPI	8
2.3	Integration of Programming Exercises: CodeOcean	8
2.4	The HPI Schul-Cloud	11
3	RELATED WORK	13
3.1	Interactivity in Computer Science Lessons	13
3.2	Interactive Worksheets	14
3.3	Programming Exercises in K-12	14
3.4	Integrating MOOCs in Classes	15
3.5	Learning Analytics	15
3.6	Online Programming Resources	16
3.6.1	Programming Education for Individuals and Classrooms	16
3.6.2	Collaborative Code Editing Using Pair Programming	17
3.6.3	Browser-Based Code Execution Platforms	17
3.6.4	Technical Implementation of Web-Based Code Execution Platforms	18
4	CURRENT SITUATION IN SCHOOLS	21
4.1	Computer Science Education in K-12	21
4.2	Technical Equipment of Schools and Practical Implications	22
4.3	Content Distribution and Submission Handling in Computer Science Classes	24
4.4	Worksheets in Computer Science Classes	25
4.5	Educational IDEs Tailored for Beginners	28
4.6	Implications for Computer Science Teachers	30
5	CONCEPT	31
5.1	General Design of Interactive Worksheets	32
5.1.1	Different Views for Students and Teachers	34
5.1.2	Accessibility of Content for Students	36

5.2	Editing Programming Exercises in CodeOcean	37
5.2.1	Automated Feedback through Unit Tests for Exercises	38
5.2.2	Referencing Exercises from Worksheets	39
5.3	Deep Integration with the HPI Schul-Cloud	39
5.3.1	Pseudonymization	40
5.3.2	Customization of the CodeOcean Integration	40
5.3.3	Worksheet Sharing and Content from MOOCs	42
5.4	Implicit Submission Handling	42
5.4.1	Pre-Evaluation of Submissions	43
5.4.2	Time Traveling to Understand the Learner's Approach	43
5.5	Learning Analytics	43
5.5.1	Integration with External Systems	45
5.5.2	Summary for Teachers During Lessons	46
5.5.3	Comparison of Learners from a School Class to MOOC Participants	46
6	IMPLEMENTATION	49
6.1	Architecture of Worksheets with Practical Programming Exercises	49
6.2	Worksheet Editor: edtr.io	52
6.2.1	First-Party Multiple-Choice Quiz Plugin	54
6.2.2	Embedding Videos from openHPI	55
6.2.3	Integrating Programming Exercises through an iFrame with LTI	57
6.3	Introduction of the Teacher Role and Study Groups in CodeOcean	58
6.3.1	Features Available for Teachers	59
6.3.2	Automated Creation of Study Groups	60
6.4	Launching Programming Exercises with Different Configurations	61
6.4.1	Deep Linking with the LTI Standard	63
6.4.2	Introducing Feature Restrictions through LTI	64
6.5	Transmitting Results from CodeOcean Back to the HPI Schul-Cloud	64
6.5.1	Differences between Final Submissions and Intermediate Submissions	66
6.5.2	Using the Worksheet Editor to Forward Analytical Data	67
6.6	Per Exercise Dashboard for Teachers	67
6.6.1	Enabling Live Updates through WebSockets	68
6.6.2	Aggregating the Working Times of Students	70
6.6.3	Request for Comments within Study Groups	71
6.7	Learnings from the Implementation	72
6.7.1	Saving Session Information in a Cookie	73
6.7.2	Worksheets with Cross-Origin Frames	73

7	EVALUATION	75
7.1	Exploration of Requirements	75
7.1.1	Methodology	76
7.1.2	Results	76
7.1.3	Discussion and Interpretation of the Results	77
7.2	Students: Testing the Prototype	77
7.2.1	Methodology	78
7.2.2	Results	80
7.2.3	Discussion and Interpretation of the Results	82
7.3	Teachers: Experiences from Testing the Prototype	83
7.4	Teachers: Usability of the Prototype	84
7.4.1	Methodology	85
7.4.2	Results	85
7.4.3	Discussion and Interpretation of the Results	85
7.5	Overall Impression	87
8	OUTLOOK AND FUTURE WORK	89
9	CONCLUSION	91
A	APPENDIX: CONCEPT	93
A.1	Conceptual Wireframes	93
A.2	Related Concepts	101
B	APPENDIX: IMPLEMENTATION	105
B.1	Exemplary Worksheet	105
B.2	Architecture	107
B.3	Implementation Details: edtr.io	109
B.4	Implementation Details: CodeOcean	113
C	APPENDIX: EVALUATION	117
C.1	Exploration: Results of the Initial Survey	117
C.2	Students: Results of the meCUE	120
C.3	Students: Results of the Survey	122
C.4	Teachers: Results of the UEQ	127
	BIBLIOGRAPHY	129
	DECLARATION	137

LIST OF FIGURES

Figure 2.1	Implementation View in CodeOcean	9
Figure 2.2	The HPI Schul-Cloud Listing Topics	12
Figure 2.3	The Current Editing View for Teachers within the HPI Schul-Cloud	12
Figure 3.1	The Web-Based IDE Codeboard	18
Figure 4.1	A Traditional Worksheet for a Computer Science Lesson	26
Figure 4.2	Gerhard Röhner's <i>Java-Editor</i>	29
Figure 5.1	Modular Worksheet Editor	33
Figure 5.2	Wireframe of an Interactive Worksheet	35
Figure 5.3	The <i>FROG</i> Editor	37
Figure 5.4	Automated Hint Provided by CodeOcean	41
Figure 5.5	Wireframe of Key Metrics Below Each Part of a Worksheet	44
Figure 5.6	Wireframe of Learning Analytics within a Worksheet	44
Figure 5.7	Pseudonymization Concept Designed for the HPI Schul-Cloud	45
Figure 5.8	Wireframes of the Analytical Dashboard	47
Figure 6.1	Exemplary Worksheet in edtr.io	50
Figure 6.2	System Architecture of Interactive Worksheets	51
Figure 6.3	System Architecture of edtr.io	52
Figure 6.4	Multiple-Choice Plugin Written for edtr.io	55
Figure 6.5	UML Diagram of the StudyGroup and Related Classes	61
Figure 6.6	Live Dashboard Available to Teachers During a Lesson	68
Figure 7.1	meCUE : Class of the First Teacher	81
Figure 7.2	meCUE : Class of the Second Teacher	81
Figure 7.3	Book vs. Video: Class of the Second Teacher	82
Figure 7.4	UEQ : Benchmark	86
Figure 7.5	UEQ : Pragmatic and Hedonic Quality	87
Figure A.1	Sketch: Creating Programming Exercises	93
Figure A.2	Sketch: Creating Worksheets with Programming Exercises	94
Figure A.3	Sketch: Solving Exercises within Worksheets	95
Figure A.4	Sketch: Reviewing Student Submissions	96
Figure A.5	Sketch: Dashboard per Exercise	97
Figure A.6	Sketch: Dashboard per CodeOcean Tag	98
Figure A.7	Sketch: Dashboard per Worksheet	99
Figure A.8	Sketch: Unit Test Generator	100

Figure A.9	Modular Worksheet Editor Offered by <i>tutory</i>	101
Figure A.10	The Digital Classroom within the HPI Schul-Cloud	101
Figure A.11	HPI Schul-Cloud Cockpit	102
Figure A.12	<i>regex101</i> Listing Matches for a Given RegEx	102
Figure A.13	The Web-Based <i>repl.it</i>	103
Figure B.1	Exemplary Worksheet on Java (Enlarged)— Part I	105
Figure B.2	Exemplary Worksheet on Java (Enlarged)— Part II	106
Figure B.3	System Architecture of Interactive Worksheets (Enlarged)	107
Figure B.4	System Architecture of <i>edtr.io</i> (Enlarged)	108
Figure C.1	Survey: Results of our Initial Survey—Part I	117
Figure C.2	Survey: Results of our Initial Survey—Part II	118
Figure C.3	Survey: Results of our Initial Survey—Part III	119
Figure C.4	meCUE : Benchmark with all Students	120
Figure C.5	Survey: Detailed Results of the First Class— Part I	122
Figure C.6	Survey: Detailed Results of the First Class— Part II	123
Figure C.7	Survey: Detailed Results of the First Class— Part III	124
Figure C.8	Survey: Detailed Results of the Second Class— Part I	125
Figure C.9	Survey: Detailed Results of the Second Class— Part II	126
Figure C.10	UEQ : Detailed Results as Table	127
Figure C.11	UEQ : Detailed Results Shown in Bars	128

LIST OF TABLES

Table 6.1	All New Application-Specific Parameters Sup- ported by <i>CodeOcean</i>	65
Table 7.1	Overview About Worksheets Tested in Lessons	78
Table 7.2	Overview of the NPS Achieved	80
Table B.1	Result of the Working Time Query	115
Table C.1	meCUE : Detailed Results of all Students	120
Table C.2	meCUE : Detailed Results of the First Class	121
Table C.3	meCUE : Detailed Results of the Second Class	121

LIST OF LISTINGS

Listing 6.1	Extract from the <i>Slate</i> Document Schema . . .	56
Listing 6.2	Concrete <i>Pundit</i> Policies Regarding Exercises .	60
Listing 6.3	<i>Pundit</i> Policies in CodeOcean Concerning Study Groups	62
Listing 6.4	HTTP Request Launching CodeOcean via LTI	63
Listing 6.5	<i>Slim</i> Template Rendering the <i>Request for Com-</i> <i>ments</i> Table Body	72
Listing 6.6	Method Call within the ActionCableHelper . .	72
Listing B.1	Extract of the Internal Document State in <i>Slate</i>	111
Listing B.2	Partial Data Model of edtr.io	112
Listing B.3	SQL Query to Aggregate Working Times . . .	115

PUBLICATION

Partial results of this thesis were submitted in advance for review as part of the following conference paper:

Sebastian Serth, Ralf Teusner, Jan Renz, and Matthias Uflacker. “Evaluating Digital Worksheets with Interactive Programming Exercises for K-12 Education”. In: *2019 IEEE Frontiers in Education Conference (FIE)*. Manuscript Submitted for Publication. Cincinnati, OH, USA: IEEE, 2019 [58]

PRIVACY NOTE

All personal data used as examples in this thesis, such as graphs on user activity or user names, are fictitious and are only shown for the demonstration of our concept.

ACRONYMS

API	Application Programming Interface
AWS	Amazon Web Services
BDSG	<i>Bundesdatenschutzgesetz</i> , the German <i>Federal Data Protection Act</i> [18]
BYOD	Bring Your Own Device
CSRF	Cross-Site-Request-Forgery
EU	European Union
GDPR	General Data Protection Regulation within the EU [14]
HPI	Hasso Plattner Institute ¹
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
ICT	Information and Communications Technology
ID	Identifier
IDE	Integrated Development Environment
IT	Information Technology
I/O	Input/Output
JSON	JavaScript Object Notation
K-12	primary and secondary education from kindergarten to 12 th grade
LMS	Learning Management System
LRS	Learning Record Store
LTI	Learning Tools Interoperability

¹ <https://hpi.de>

meCUE	m odular e valuation of key C omponents of U ser E xperience [43]
MINT	M athematics, I nformatics, N atural sciences and T echnology, a common abbreviation for STEM fields in Germany
MINT-EC	<i>Verein mathematisch-naturwissenschaftlicher Excellence-Center an Schulen e. V.</i> , literally “Association of mathematical-scientific Excellence Centers at Schools” ² — also see MINT
MOOC	M assive O pen O nline C ourse
MVC	M odel- V iew- C ontroller
NPS	N et P romoter S core [50]
OER	O pen E ducation R esource
PC	P ersonal C omputer
PDF	P ortable D ocument F ormat
RegEx	R egular E xpression
REPL	R ead- E val- P rint- L oop
REST	R epresentational S tate T ransfer
SPA	S ingle- P age A pplication
SQL	S tructured Q uery L anguage
STEM	S cience, T echnology, E ngineering and M athematics
UEQ	U ser E xperience Q uestionnaire [34]
UI	U ser I nterface
UML	U nified M odeling L anguage
URL	U niform R esource L ocator
USB	U niversal S erial B us
WYSIWYG	W hat Y ou S ee I s W hat Y ou G et, an acronym that mainly describes editors providing the result view while editing content
xAPI	E xperience A PI [70]

² <https://www.mint-ec.de>

INTRODUCTION

In recent years, the term *Digital Transformation* became popular to describe ongoing changes in society and organizations by redefining workflows and habits with the help of Information and Communications Technology (ICT) [37, 66]. While private households and businesses are able to adopt changing requirements rapidly, government agencies are often not as agile and require a longer time to adjust their processes and existing structures. In Germany, education is one of the public services financed by the government and is under supervision of government agencies, which is also defined in Article 7 of the German constitution *Basic Law* [4]. In practice, the German federal states define a syllabus and allocate funds for public schools resulting in a dependency of schools and teachers on the guidelines specified by the corresponding ministry. Technology-wise, public schools are chronically under-financed which led to the *DigitalPakt Schule* (literally “*DigitalPact School*”), a series of laws to support the *Digital Transformation* in schools¹. Its main purpose is to set up a modern, internet-connected infrastructure in schools to support learning with digital media. Consequently, many teachers are encouraged to include online content and the use of computers or tablets into their lessons. As part of digital education, more and more federal states in Germany also include mandatory computer science education into their syllabus, which is backed by the improved, technical infrastructure. This thesis concentrates on learning programming skills as part of the computer science education and illustrates how to leverage the new possibilities offered by digital media for teachers and students.

1.1 MOTIVATION

In primary and secondary education from kindergarten to 12th grade (K-12), teachers generally either present new topics to their students or practice and repeat previous subjects. For this, they have the choice between using an existing schoolbook or workbook with given content or creating their own worksheets. In the following, we concentrate on the worksheet scenario in German schools. When creating worksheets, teachers usually reuse existing content from schoolbooks or from other resources. Regardless of their sources, they are able to adapt the content to their needs when composing a new worksheet. This

¹ <https://www.bmbf.de/de/wissenswertes-zum-digitalpakt-schule-6496.html>

adaption is one of the biggest advantages teachers have: The freedom and possibility to tailor each part of the worksheet to fit their current situation in class to best support their students. Traditionally, these worksheets are distributed as paper copies by the teachers whenever needed. For teachers, this distribution form has some advantages as they know prior to the lesson that their material is copied and thus ready and that they do not need to rely on anything else (such as further technologies) to conduct their lesson.

Traditional, paper-based worksheets also have some general drawbacks when being compared to digital resources. Regardless of the environmental or financial costs, printed worksheets lack support for any form of interactivity or for the adaption of additional content to specific students. Many teachers strive for interactivity to foster active participation of their students. Thus, given the choice, today's teachers prefer using digital resources for their lessons if available. By doing so, they circumvent the missing interactivity and also get the benefit of links to other rich content (independent of the location, for example, a web resource).

For computer science classes, especially those teaching programming skills, additional drawbacks arise. Similar to other classes, teachers prepare content to focus their students' attention on the most important parts. While a figure, e. g., a schematic representation of a heart in biology classes, can be easily annotated by students on a sheet of paper, it's difficult to work with programming source code on traditional worksheets, particularly if students must execute the code to observe the program behavior. If the code is distributed in paper form, students are required to manually typewrite the given code which is time-consuming and error-prone. Depending on the possible mistakes students make, it might even be difficult for teachers to find the bug in a chaotic "rewrite" of their own template. A second issue occurs when the teacher decides to discuss results or when collecting students' work. As every individual learner developed own solutions and probably saved it locally on the Personal Computer (PC) in front of them, the teacher has no direct access to the source code files. However, the teacher needs access for collecting the submissions and thus must rely on the assistance of the students to copy all files required to compile and run the program to a provided location. This increases the complexity for all parties involved and the time effort for the teacher when compared to collecting paper-based submissions. However, the work created by students is only available in a digital format and thus students cannot submit their work on a sheet of paper to the teacher. Based on the collection of these files, the teacher might also ask students to present their work in front of the class using a projector. When having no access to the source files, some teachers might have the possibility to share a student's desktop on their machine and thus circumvent the need for copying the files manually. However, this

requires further software to be installed on the computers, compatible network architecture and is no viable solution for schools with a **Bring Your Own Device (BYOD)** policy.

A possible solution is bringing the cloud concept to schools. One approach is the *HPI Schul-Cloud*² (literally “*School-Cloud*”) being developed to fit the German education system with its strict legal requirements. It provides a web-based interface and native mobile applications to manage dates in a calendar, provide file storage or to form classes with different topics including a homework submission system. So far, it is used by more than 600 teachers and 2,000 students all over Germany. For non-computer science classes, the platform addresses some of issues raised, such as missing interactive content for example by integrating a GeoGebra plugin³ or the interactive whiteboard solution nexboard⁴. Currently, the *HPI Schul-Cloud* however lacks specific support for computer science classes.

Inspired by **Massive Open Online Courses (MOOCs)** [62], teachers wish to access tools used in e-learning environments for several reasons: First of all, teachers like the didactical approach and the methodology used in the courses. The course videos often use a different approach to explain abstract concepts than the one used in the schoolbook. In addition, online courses include interactive content, such as simple multiple-choice quizzes for self-assessment or interactive programming exercises. For the latter, the platforms usually provide a simplified web interface that is tailored for novices and offers additional support capabilities. For example, teachers appreciate that the code execution platforms used in **MOOCs** give direct feedback to students while working on the code. This direct feedback frees teachers from providing the same simple feedback over and over again and allows them to concentrate on students strongly struggling with the given task. In addition, these web-based code execution platforms are maintained by the respective platform provider eliminating administrative overhead for schools. If not using online tools, the **PCs** used in computer science classes need to have the corresponding language tools (such as a compiler or interpreter) to be installed in conjunction with correctly configured developer tools. In comparison, everything that is required to participate in **MOOCs** is a computer with a modern web browser and a stable internet connection.

2 <https://schul-cloud.org>

3 GeoGebra is an interactive geometry software used to visualize graphical elements — <https://geogebra.org>

4 <https://nexenio.com/nexboard>

1.2 RESEARCH QUESTIONS

Given the content available through MOOCs and the provided tools, some teachers currently embed resources from these e-learning tools in their classes. However, MOOC platforms are not tailored for this use case as they were built with a different design goal in mind. This results in two substantial drawbacks: (1) Teachers are unable to adapt the course material hosted online to their needs and the knowledge level of their students—the content is not editable and feels *static*. In addition, (2) the MOOC platforms prevent teachers from getting insights into the learning progress of their students, making teachers dependent on the feedback of their students.

To tackle these drawbacks and improve teachers' current situation in educating students the basics of programming, we address the following research questions:

- RQ1. How can we enable teachers to reuse and adapt exercises (e. g., from MOOCs) and create their own interactive worksheets?
- RQ2. Which tooling support do teachers need to help students struggling with given programming exercises?
- RQ3. How can we leverage learning data to enhance teaching effectiveness and help the teacher to achieve lesson goals?
- RQ4. Which support do students need to get individual help, either from their fellow students or from their teacher?
- RQ5. Which of the results gathered from the questions above can be transferred to the general MOOC context?

1.3 STRUCTURE

In this thesis, we focus on interactive worksheets with programming exercises in the context of the HPI Schul-Cloud as described in [Section 1.1](#). Before diving deeper into our concept, we give some background information in [Chapter 2](#) on the HPI Schul-Cloud, MOOCs and programming exercises in general with their respective use cases. [Chapter 3](#) introduces related work in six categories, mostly focusing on MOOCs, learning analytics and programming education. Moreover, the chapter introduces existing tools in this field and shortly highlights their advantages and disadvantages.

For developing our concept of interactive worksheets, we interviewed several high-school teachers about their usage and expectations of worksheets and digital resources. The insights we gathered are summarized in [Chapter 4](#) and form the foundation of requirements for our concept. Especially those teachers already using digital resources reg-

ularly in their lessons, reported from advantages and disadvantages of their current solutions. [Chapter 5](#) elaborates in detail on our concept of inlining interactive content, especially programming exercises, in a worksheet with detailed analytics for teachers. As part of the concept, we explain our research journey based on various iterations of our prototype. The following [Chapter 6](#) presents implementation-specific details of our worksheet editor with embedded programming exercises. It outlines the general architecture used to create a seamless user experience with special respect to the worksheet itself and learning analytics.

We used our prototype to get feedback from students and teachers by giving them access to our tool. The results of our evaluation based on the usage of our prototype within four regular lessons are described in [Chapter 7](#). Throughout the chapter, we also discuss how students and teachers evaluate our concept and the prototype. Based on the evaluation, we It also includes additional feedback from teachers evaluating our concept without using it with their students. The thesis closes by showing directions of future work ([Chapter 8](#)), separated in short- and long-term suggestions. Finally, [Chapter 9](#) summarizes the most important findings and also revisits the research questions by providing a short answer for each.

BACKGROUND

This thesis integrates interactive worksheets into the HPI Schul-Cloud and focuses on programming education. It is inspired by requirements teachers expressed when using MOOCs in their lesson. [Section 2.1](#) summarizes the concept of MOOCs shortly. As our university is one of the MOOC providers in Germany offering entry-level programming courses, we are regularly contacted by teachers expressing their interest in using our content (including our practical exercises) in their classes. Within this chapter, we provide an overview about the existing offers and details about the MOOC platform ([Section 2.2](#)) and the web-based code-execution platform used in programming courses ([Section 2.3](#)). Further, [Section 2.4](#) introduces the HPI Schul-Cloud with its current design goals and the existing architecture intended to provide a unified cloud solution for all schools in Germany.

2.1 MASSIVE OPEN ONLINE COURSES

In 2011, an innovation with global impact debuted in the US: Sebastian Thrun, professor at Stanford University, published one of his lectures on the Internet free of charge and thereby reached an audience of almost 160,000 people [13]. He was the first to reach such a large audience with his teaching activities. Nowadays, the mass distribution of learning content via online services has been established and gained popularity. The concept of making learning content accessible to a broad audience via the Internet is called **Massive Open Online Course (MOOC)**. Usually, these courses contain small learning sequences on a given topic and run in a fixed period of time. During this period, users have the opportunity to watch learning videos, participate in self-assessment quizzes or gain points through weekly homework assignments. Further, MOOCs provide a forum to allow learners to ask questions and discuss the learning content. The activity of a learner including progress information and scores is commonly referred to as learning data. Upon successful completion of the course, each participant receives a certificate stating the course and the respective score. MOOCs are offered by universities and private institutions on many topics and designed for various audiences. While some are designed for novices, others target experts or are offered in cooperation with universities to be part of an academic degree.

2.2 MOOCS OFFERED BY THE HPI WITH OPENHPI

As one of the first universities in Germany, the Hasso Plattner Institute (HPI) launched its MOOC platform called *openHPI*¹ in September 2012 [41]. openHPI offers Information Technology (IT)-related courses for different age groups and covers a broad range of topics. One of the tracks is dedicated on programming and involves the programming languages Python, Java and Ruby in different courses. Especially the Python and Java courses are designed for beginners and are targeted at high-school students. The courses require no prior knowledge and guide learners through the first steps in the corresponding programming language. Besides video and quizzes, these courses also include practical programming exercises. Therefore, learners are encouraged to write small programs and observe their behavior. The exercises are presented in a web-based programming environment called CodeOcean², which is described in more detail in the following Section 2.3. These courses teaching Java and Python did not only attract students directly but also caught attention from high-school teachers. They expressed their interest in using the courses with their students. To comply with the time requirements teachers have, the existing courses were repeated in a school edition with an extended time schedule. Instead of unlocking new content on a weekly basis, these repetitions implemented a three week time horizon for each module (previously called a week). However, as this thesis elaborates, this adaption is not sufficient for many teachers and thus prevents a broad usage of MOOCs in classes. In 2019, the programming courses debuted on mooc.house³, the white-label platform⁴ of openHPI, as school editions with extended module lengths.

2.3 INTEGRATION OF PROGRAMMING EXERCISES: CODEOCEAN

The programming courses offered on openHPI include practical programming exercises that are provided on a web-based code execution platform called CodeOcean [60]. Figure 2.1 provides an example of an exercise being edited and graded in CodeOcean. The platform has many advantages for learners and teachers alike:

- (1) CodeOcean offers a predefined view on the exercises with scaffolded source code including syntax highlighting and allows editing these files directly within the browser.

¹ <https://open.hpi.de>

² <https://codeocean.openhpi.de>

³ <https://mooc.house>

⁴ A white-label platform is a hosted service targeting businesses that are interested in offering the same services under their brand to their customers.

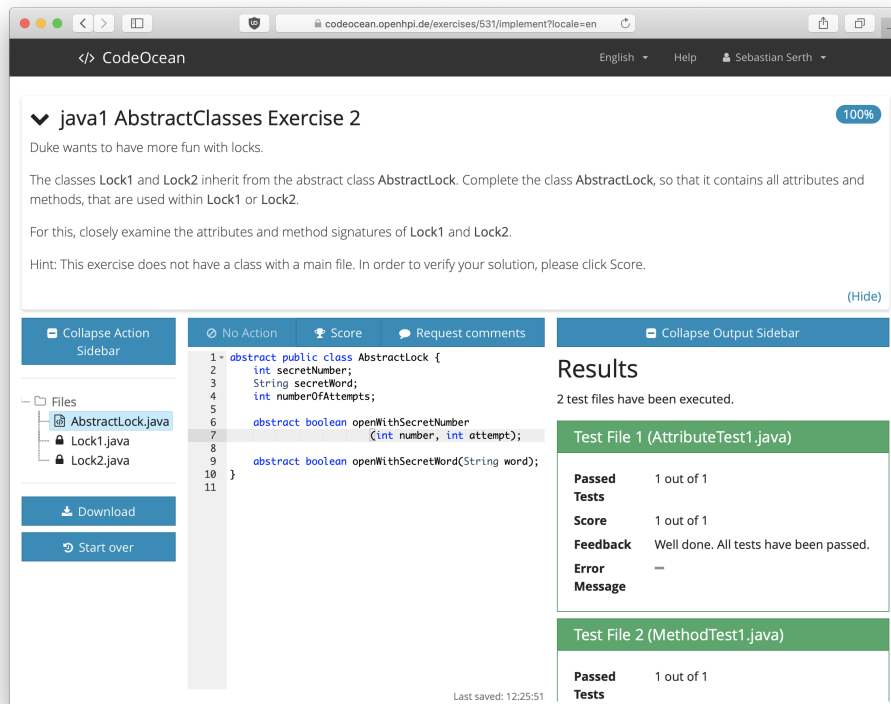


Figure 2.1: Implementation view of an exercise in CodeOcean. The exercise consists of a description (top), multiple files (left), the editable source code (center) and test results (right).

- (2) The platform allows executing the code on the server and streams the output back to the learner's web browser. Together with the customized view, learners are supported to get up with the programming in no time instead of taking care of the development environment. Learners do not need to download and install any compiler or runtime. Thus, MOOC instructors minimize technical help requests about correct machine setup and can focus forum posts on the content provided in the course.
- (3) CodeOcean includes unit tests to provide feedback for learners and score their code. A unit test is defined as a program that either runs the learner's code in a pre-defined way and compares the provided result with an expectation or the unit test parses the student's source code and matches it against an exercise-defined string. While the code of the unit tests are hidden, learners can run the unit tests at any time and get instant feedback whether they passed or failed. If the unit tests fail the result is shown together with an error message defined by the MOOC instructors. On the one hand, this feedback helps people to help themselves and provides learners with a hint of their mistake. On the other hand, the automated scoring using unit tests is required to indicate progress for the learners. In the context of a MOOC with thousands of active learners, a manual review by the instructors is not feasible

and peer-review of source code has not been implemented in CodeOcean so far [60].

- (4) When MOOC instructors with admin privileges use CodeOcean in their programming MOOCs, they get basic insights about the student performance of the given tasks. For example, CodeOcean tracks the working time of students per exercise (together with the achieved scores) and already offers some statistics, such as the average working time and score enabling an anomaly detection for items being difficult for learners (i. e., based on the time spent to achieve a full score).
- (5) In CodeOcean, learners can ask questions about their program directly within the platform and in context of their current program. Usually, MOOC platforms provide a forum to discuss questions. While this concept also works great for source code in general outside of a MOOC (cf. StackOverflow⁵), it is an additional barrier for novices to summarize their problem externally. To understand the problem, contextual information is generally of help for others to provide the current solution. When using a dedicated forum, learners are required to provide as much information as necessary to reproduce the issue which beginners might find difficult to identify. As a result, they might copy too few or too much information. In addition, early iterations of the Java courses showed that learners did not format their source code appropriate in forum posts (but as plain text)⁶, making it difficult to read. With *Request for Comments*, CodeOcean provides a built-in feature to ask a question in the context of an exercise, thus lowering the barriers to get help [64]. CodeOcean presents the learner's source code and error message together with the question to fellow students and allows them to add a comment specifically to one line of code. Hence, the previously described issue is solved with a dedicated forum.

CodeOcean is mainly used in the context of MOOCs (such as those offered on openHPI and mooc.house) and has been used by more than 42,000 users as of May 2019. CodeOcean is a stand-alone tool implementing the Learning Tools Interoperability (LTI) standard⁷ to be used in various learning scenarios. By offering an LTI interface, it is accessible from MOOC providers as well as other providers, such as the HPI Schul-Cloud (see Section 2.4). CodeOcean itself cannot be used directly by learners or other users than the MOOCs instructors or administrators.

⁵ <https://stackoverflow.com>

⁶ <https://open.hpi.de/courses/javaeinstieg2015/pinboard>

⁷ <http://www.imsglobal.org/activity/learning-tools-interoperability>

2.4 THE HPI SCHUL-CLOUD

As described in [Chapter 1](#), the *Digital Transformation* is an ongoing process in many organizations and the society in general. However, many German schools are unable to profit from the *Digital Transformation* so far as most of them do not have a suitable IT-supported learning environment. The HPI Schul-Cloud aims to close this gap and develops a cloud infrastructure specifically meeting the legal and organizational requirements within German schools [39]. It is built upon a scalable micro-service architecture by the HPI in cooperation with the *Verein mathematisch-naturwissenschaftlicher Excellence-Center an Schulen e. V.*, literally “Association of mathematical-scientific Excellence Centers at Schools” (MINT-EC). Within the HPI Schul-Cloud, students can access content provided by their teachers within topics as shown in [Figure 2.2](#), submit (homework) assignments and organize themselves with a personal calendar. Teachers might post news or upload files for their upcoming lessons or browse through existing content in an existing *Lern-Store* (literally “Learn-Store”). Moreover, the HPI Schul-Cloud allows editing documents collaboratively and supports external tools, such as bettermarks⁸, a math learning tool.

Content-wise, the HPI Schul-Cloud allows teachers to prepare topic pages and publish these for later use during lessons. [Figure 2.3](#) visualizes the current editing experiences for teachers. In addition to static documents, the HPI Schul-Cloud integrates interactive tools, such as a GeoGebra⁹ or neXboard¹⁰. So far, these plugins are user-agnostic and either do not preserve changes or synchronize their current state for all connected users. The submission system available in the HPI Schul-Cloud allows students to submit text or upload files to be graded by their teacher.

⁸ <https://de.bettermarks.com>

⁹ GeoGebra is an interactive geometry software used to visualize graphical elements — <https://geogebra.org>

¹⁰ neXboard is an interactive whiteboard solution — <https://nexenio.com/nexboard>

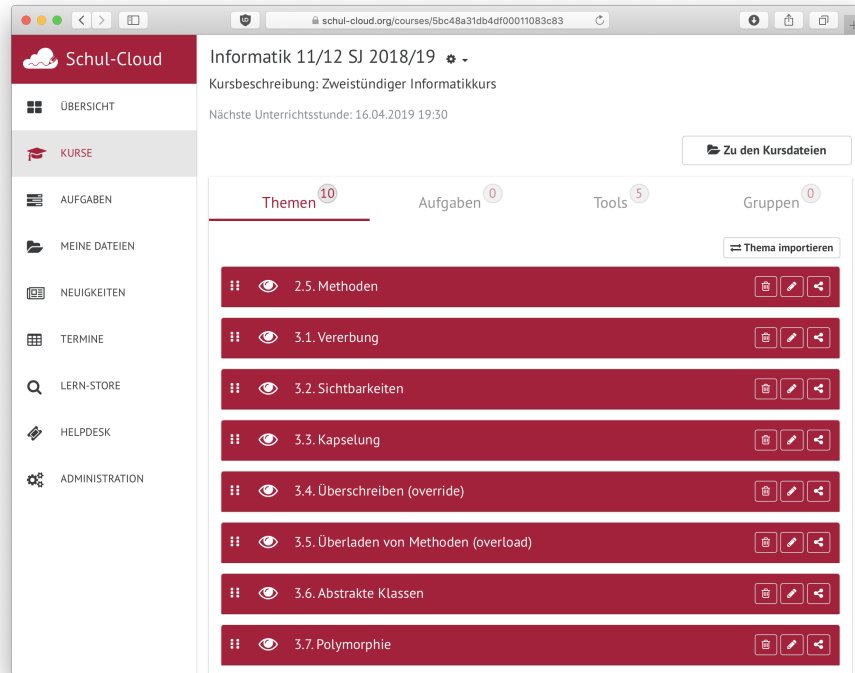


Figure 2.2: The HPI Schul-Cloud listing different topics within a computer science course. Teachers use topics to group learning materials and structure their courses.

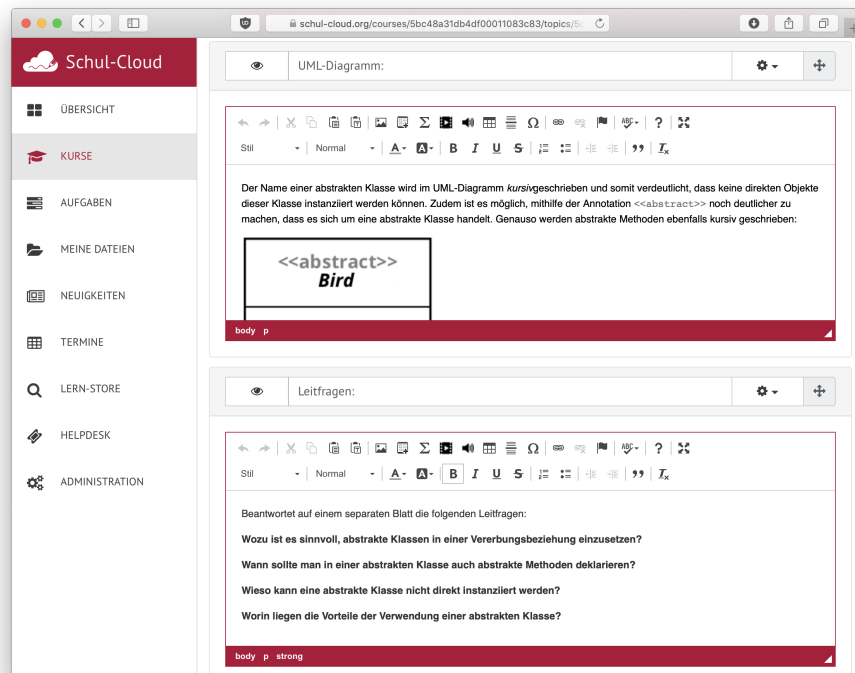


Figure 2.3: The current editing view for teachers within the HPI Schul-Cloud. The content is separated in different blocks which are either text-based or include interactive elements.

RELATED WORK

This work is based on existing research in the context of interactive elements using [ICT](#) equipment in different school lessons and prior research on how to embed content from [MOOCs](#) in hybrid classes. Furthermore, we highlight different didactical approaches on how to use programming exercises in [K-12](#) education. The following paragraphs give a short overview of the work this thesis builds on in two main categories: Ongoing research ([Section 3.1](#) to [3.5](#)) and existing solutions ([Section 3.6](#)).

The related work in the research category begins with work in the context of interactive elements in computer science lessons ([Section 3.1](#) and continues in [Section 3.2](#) with interactive worksheets. As a key component of computer science education is programming, [Section 3.3](#) summarizes insights about programming exercises in [K-12](#). [Section 3.4](#) details the integration of [MOOCs](#) in classes and [Section 3.5](#) completes the research category with details on learning analytics.

Prior to analyzing the current situation in schools (see [Chapter 4](#)), [Section 3.6](#) introduces some of the tools available for programming education and provides background information on the existing approaches.

3.1 INTERACTIVITY IN COMPUTER SCIENCE LESSONS

In 2013, Othman et al. stated that computer science classes “contain numerous abstract concepts that cannot be easily explained using traditional educational methods” [47]. The authors compare different forms of online interactive content that can be used in addition to traditional learning methods, such as simulations, tutorials or quizzes. This is in line with the findings from Merchant et al. who describe that the use of games, simulations and virtual worlds improve the learning outcome gains in [K-12](#) [42]. Existing [MOOCs](#), especially those designed for a younger audience, make use of this strategy and tell a story that requires the help of the learner to solve a given problem [21]. Whenever it comes to interactive elements, Merchant et al. also highlight the importance of the individual experience. Compared to group interactivity, situations which actively require the involvement of every single learner enhance the student performance. Beauchamp and Kennewell further describe the advantages of using [ICT](#) as it supports

the learner influence on content and methodology [5]. However, they also state that the usage of those resources needs to be well embedded in the lesson to push the use of technology in the background while focusing on the desired content.

3.2 INTERACTIVE WORKSHEETS

While little research is available specifically for the use of interactive worksheets in computer science classes, the concept has been tested in other areas. In 2000, Leslie-Pelecky showed that interactive worksheets in an introductory physics course increased the student/teacher interaction [35]. The main cause is the way how worksheets and embedded tasks are structured: They follow a common goal and, if designed appropriately, build on each other, starting with easier tasks. Usually, students work in the same order as the exercises appear on the worksheets. As soon as students start struggling, the teacher gets an impression which exercise is causing the students' problems and thus can intervene orally. Another advantage of interactive worksheets is described by Blayney and Freeman: Students are put in the position to learn at their own speed and pace [8]. This supports the individualization of the learning process, especially with an interactive worksheet that gives direct feedback to the learner. The researchers describe this as a possible mechanism to support students where they need the most help. With respect to the level of interactivity and feedback, Blayney and Freeman also concluded in a previous paper that better exercises than multiple-choice quizzes are needed and that it is important to provide frequent feedback to learners [7]. While multiple-choice quizzes are technically easy to verify, they ask only for a subset of the content learned and do not allow the students to demonstrate their understanding of more complex content, such as approaches or methodologies.

3.3 PROGRAMMING EXERCISES IN K-12

For computer science classes with a focus on programming education, an important part of the learning process is to apply the knowledge learned by writing source code (amongst other competences, such as modeling). Besides simple **Read-Eval-Print-Loop (REPL)** (such as the one offered on repl.it¹, depicted in Section A.2 as Figure A.13), teachers wish that their students learn to code small programs. A good way to do so is described by Isomöttönen et al., who propose to provide students with scaffolded source code [26]. Besides that, scaffolding also helps students to make progress in a shorter period of time and

¹ <https://repl.it/languages>

abstracts more advanced concepts for novices. Other approaches are discussed by Hubwieser et al., as they raised many questions regarding the content and methodology of programming education [25]. In this thesis, we tackle some of these questions, especially those with respect to the tools and the environment required to support teachers in their teaching practice by reusing and customizing content, e. g., available from MOOCs.

3.4 INTEGRATING MOOCS IN CLASSES

The majority of teachers who embedded MOOCs in their classes were satisfied with the results. This is due to the excellent content available and the use of other teaching approaches, according to research by Griffiths et al. in 2014 [20]. However, Israel showed that fitting existing courses that are not tailored for in-class usage is a huge challenge, for example regarding the engagement of students and the learning effectiveness [27]. A potential downside is described by Griffiths et al.: When comparing traditional face-to-face only lessons and hybrid lessons using MOOCs, student satisfaction is lower in hybrid lessons, while overall learning results remain on an equal level [20]. Counter measurements to prevent a decrease of satisfaction need to be taken by the teacher, for example by providing time within lessons to discuss the online content and to answer upcoming questions. Caulfield et al. have shown that students prefer this direct communication for questions initiated by them over asking for help in an online community as typically available in a MOOC [10]. Teachers also benefit from the in-class discussions as they allow them to get an understanding of the current progress and possible problems of their students. Generally, teachers have no direct way to access student activity in MOOCs [27] as MOOC platforms offer only two roles: MOOC instructors and participants. The role of a teacher or teaching assistant is currently not represented. Also, additional requirements imposed by school usage, such as stretched course runtimes and awareness of school holidays are not represented in most MOOCs.

3.5 LEARNING ANALYTICS

Getting insights about the learning data is achieved with so-called learning analytics and is not only valuable for teachers to perform grading-related tasks, but primarily of interest for them to improve their lessons and support students who need extra help [38, 45]. Teachers can use the information provided through the platform to prepare upcoming lessons, for example by including a repetition to tackle existing gaps in understanding of a topic. Learning Analytics, such as the

individual learning path, the time spent, or points achieved also allow for predicting students' performance, as described by Williamson in 2016 [68]. Concerning programming exercises, Blikstein describes that the frequency of code runs and source code changes allows inferring the coding behavior, opening an additional window into students' cognition and approach [9]. Berland et al. describe that these results can either be used by a teacher (as described above) or to enable automated feedback to the learners [6]. Besides the increase in self-awareness of possible problems, Long and Siemens found out that showing personalized progress information is motivating for learners when compared to learning goals or peers [36]. Current research, for example as conducted by Rohloff et al., evaluates how to integrate learning objectives and corresponding feedback into MOOCs [56].

3.6 ONLINE PROGRAMMING RESOURCES

Besides MOOCs, other online resources are available to learners interested in a programming language (see Section 3.6.1). These might either target individual learners or are designed to support learning in a group, either within a company or in schools. Some specifically support collaborative code editing (cf. Section 3.6.2) often present in schools due to missing school computers. Other offers have specific use cases in mind not directly connected to help learning the basics of a programming language. Instead, they focus on coding challenges or recruiting processes (Section 3.6.3). When executing user-generated code from within a browser, several technical solutions exist, which are briefly characterized in Section 3.6.4.

3.6.1 Programming Education for Individuals and Classrooms

Individual Learners might use open books, such as predecessors of current compendiums of a specific language (for example *Java ist auch eine Insel* [65], literally "Java is also an island"), available free of charge on the web². While teaching the basics of a language, these (open) books usually miss a dedicated forum or web-based code execution platform. Some platforms, such as codecademy³, CodingBat⁴ or SoloLearn⁵ combine textual explanations with a guided code editor, multiple-choice quizzes and a forum. In addition, platforms, such as Treehouse⁶ (fee required) also feature video explanations and optionally one-on-one support. These offers differ from MOOCs by the

² e. g., on Java: <http://openbook.rheinwerk-verlag.de/javainsel/>

³ <https://www.codecademy.com>

⁴ <https://codingbat.com/>

⁵ <https://www.sololearn.com>

⁶ <https://teamtreehouse.com>

flexible time schedule and allow unlocking new content after solving previous content sections. With respect to team education, Treehouse offers a team plan on request featuring individual progress reports and adaptable learning goals.

3.6.2 Collaborative Code Editing Using Pair Programming

If a school is unable to provide one PC for each student during computer science lessons, two students are required to share a single computer. For programming assignments, the approach to work collaborative in pairs of two might even be advantageous and is also used intentionally within the software industry where it is known as *Pair Programming*. It defines a process, where one person actively types code as the driver and the other, called the navigator, watches out for mistakes and acts as a brainstorming partner [46]. As shown by Nagappan et al. in 2003, pair programming helps students in an introductory computer science class to perform better (compared to solo learners) within a less frustrating class experience [46]. The educational platform Qualified⁷ supports students to work in *Pair Programming* and constantly captures snapshots of the code edited by students to help a teacher in understanding the development process retrospectively.

3.6.3 Browser-Based Code Execution Platforms

With Codeboard⁸, a web-based IDE is available that supports teachers in creating scaffolded exercises and collecting student submissions. Moreover, as shown in Figure 3.1, Codeboard enables students to edit and run code directly in the browser. Other platforms that allow writing and in most cases running code in a web browser are either coding games, recruiting tools or offer further learning material for a deep dive on a specific topic. CodinGame⁹ is one of the sites offering puzzles and mini-games for experienced developers to improve their coding skills through writing a few lines of source code. This might come handy for developers seeking a new job. In most cases, hiring candidates have to pass one or more online coding challenges (or traditionally interviews) to show off their capabilities. HackerRank¹⁰ and Codility¹¹ are two examples of platforms that support the recruiting process by providing tasks together with a runtime in the browser. Similar to the student/teacher relationship, companies have similar

7 <https://www.qualified.io/for-education>

8 <https://codeboard.io>

9 <https://www.codingame.com>

10 <https://www.hackerrank.com>

11 <https://www.codility.com>

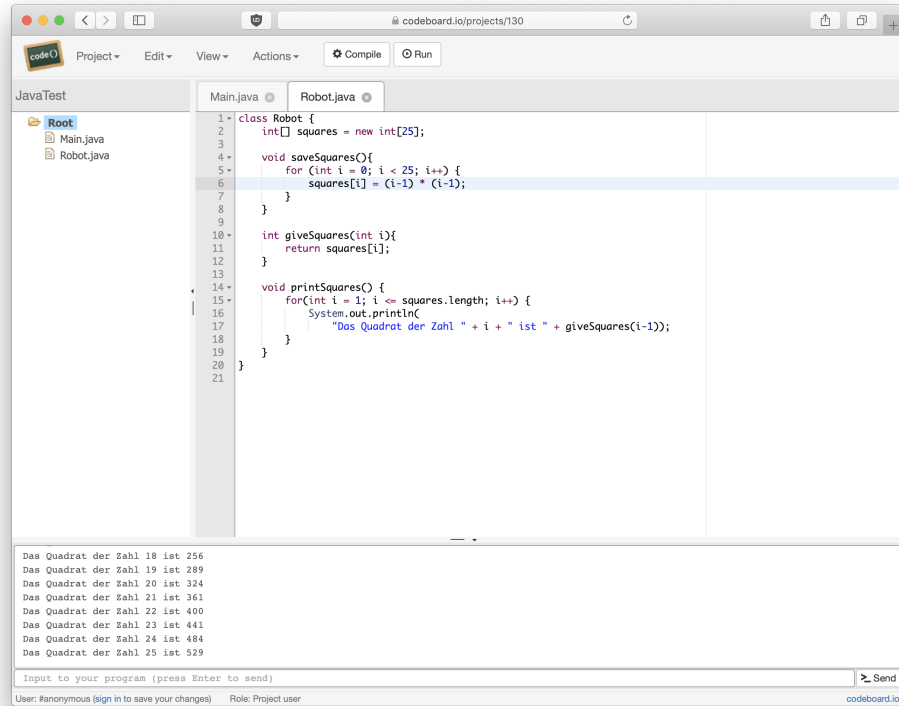


Figure 3.1: The web-based IDE Codeboard allows learners to create and run programs without installing local software [15]. Codeboard can be used as a stand-alone tool or as part of a MOOC. Similar to a traditional Integrated Development Environment (IDE), it structures code into projects and allows learners to save them online.

requirements, e. g., regarding the submission handling and automated correctness checks via unit tests. Qualified also supports screening candidates and offers experienced learners the opportunity to further strengthen their skills with Codewars¹². The boundary between pure learning material (more and more often with interesting challenges) and educational coding games is increasingly blurred.

3.6.4 Technical Implementation of Web-Based Code Execution Platforms

In traditional development scenarios, developers execute source code (after compilation / interpretation) locally on their machine or a remote system they have access to. Most IDEs support local and remote run targets and are thus allowed to start new processes. Using web browser, however, starting system processes is not desired (due to the impact on security) and therefore restricted with the sandboxing all major browsers implement. As a result, code execution, as done by traditional software development, is unavailable. Instead the code

¹² <https://www.codewars.com>

execution platforms mentioned in the previous sections have to use one of the following two approaches:

- (1) Code execution in the web browser: Modern browsers include a JavaScript engine that allows them to run JavaScript code or WebAssembly¹³ as part of rendering a website. Some platforms, such as PlayCode¹⁴ use the integrated JavaScript engine to execute user-generated code. As a result, the code stays and runs locally on the user's machine, similar to traditional development processes. The platform provider does not need to care about an infrastructure for code execution, security implications or scaling factors. However, this approach is limited to JavaScript and does not allow the native execution of code written in other languages. To circumvent that restriction, other platforms, such as TryRuby¹⁵ use a transpiler to generate JavaScript code from another language, in this case Ruby. However, transpilers are not available for any combination of two programming languages and might miss advanced features of the source language. Moreover, JavaScript in the browser still has limitations, e. g., regarding File Input/Output (I/O) due to the browsers sandboxing.
- (2) Remote code execution on a server: Instead of executing code in the user's browser, another approach is to send the user-generated code together with the scaffolded code to an execution server and to stream the I/O operations back to the user's browser. This approach is more flexible than the browser-based code execution and, generally speaking, available for all programming languages. This is advantageous for users as it is independent of the performance of their own devices and provides the full capabilities of the respective language. For platform providers, this approach raises questions on the security aspects and sandboxing of user-generated code. Therefore, providers typically either use a serverless function where the source code is executed on a third-party server, such as Amazon Web Services (AWS) Lambda¹⁶. Or, they execute code on their servers using a Docker¹⁷ container, which provides an isolation of (user) applications using a sandboxed environment. In both cases, users are technically enabled to perform arbitrary I/O operations, if not limited otherwise by the platform provider.

13 WebAssembly is low-level byte code designed for the use in web browsers — <https://webassembly.org>

14 <https://playcode.io/online-javascript-editor>

15 <https://ruby.github.io/TryRuby/>

16 <https://aws.amazon.com/lambda/>

17 <https://www.docker.com>

CURRENT SITUATION IN SCHOOLS

To assess the current situation in schools, we conducted interviews with thirteen computer science teachers in K-12 education, two principals and five students. In these interviews, we gathered insights on the approaches teachers use in typical lessons, the resources they use and the equipment that is available for them during (computer science) lessons. The following sections summarize the findings: [Section 4.1](#) gives an overview about computer science education in K-12, especially in Germany, and [Section 4.2](#) outlines the technical equipment available in schools today. Based on the resources available to teachers, [Section 4.3](#) focuses on distribution forms for teachers and also considers education platforms designed for schools. In addition to online resources, many computer science teachers use own worksheets with learning materials and exercises, even though they face issues with integrating source code (see [Section 4.4](#)). Regarding the didactical approaches within computer science education, [Section 4.5](#) highlights common tools and IDEs used in K-12 programming lessons. Finally, [Section 4.6](#) formulates key requirements students and teachers have on technology support in computer science classes.

4.1 COMPUTER SCIENCE EDUCATION IN K-12

In Germany, the sixteen federal states are responsible for the financial and content-related orientation of education in their state (as shortly described in [Chapter 1](#)). Consequently, the ministries of education define in their syllabus whether schools should teach computer science at all, which types of schools offer computer science education and what students learn at which age. As of 2019, computer science education is still not mandatory for students in all federal states, which stands in contrast to the ongoing *Digital Transformation* and resulting claims of the society and politicians [33]. Due to the low prevalence, many computer science teachers are unsure about the materials and didactical approaches to use. They find the materials available to them, such as schoolbooks or recommendations from the ministries, incomplete or see major potential for improvement. In addition, many schools do not have schoolbooks for students in computer science classes or describe those as outdated. These are some of the reasons why teachers tend to use other resources than books and invest much time to prepare content themselves. Known

from other lessons, teachers create worksheets with the content they wish to teach in the upcoming lessons.

As schoolbooks might not be available (or heavily outdated), teachers search the web for suitable resources. In the German area, many teachers reported to (re-)use content from **Open Education Resources (OERs)** *as-is* or in a slightly modified way. As one of those, the open schoolbook *inf-schule*¹ is common and is capable of replacing traditional schoolbooks. Similar to a printed version, it does not feature interactive elements but includes exercises and offers sample solutions for teachers. Furthermore, computer science teachers exchange (parts of) their content or make it available for the interested public. For example, the content published by Tino Hempel on his personal website² is known by many teachers and students. Additionally, some teachers use **MOOCs** and combine their lessons with e-learning resources (known as *blended learning*). [Section 2.1](#) and [Section 3.4](#) give background information on **MOOCs** and discuss some of the advantages and disadvantages teachers have with blended learning.

4.2 TECHNICAL EQUIPMENT OF SCHOOLS AND PRACTICAL IMPLICATIONS

More than any other field, computer science education depends on the technical equipment of the schools. Without internet-enabled **PCs** provided by the schools with the appropriate development tools installed, computer science education is almost unfeasible. In everyday life at school, many issues arise:

- (1) The equipment in schools is, generally speaking, mostly inadequate. Both of the principals we interviewed reported to have too few **PC** pools available for the demand of teachers, especially with respect to the class sizes. They further indicated that their internet connection is not designed for the number of concurrent users they have, resulting in a poor speed or an unreliable connection.
- (2) The technical infrastructure at schools requires constant administration; otherwise it will likely get outdated. Due to the usage of many different students and teachers, unpredictable problems typically arise over time. School principals and computer science teachers describe difficulties in finding a responsible **IT**-administrator taking care of the systems. While only few schools have a dedicated administrator, most other schools are mainly managed by one of the computer science teachers. Due to the personnel bottlenecks, changing requirements on the software installed on **PCs** are difficult to implement. Moreover, technical stability is a key factor:

¹ <https://inf-schule.de>

² <https://tinohempel.de>

With some lessons only lasting about 45 minutes, no time should be wasted by unresponsive computers or other technical issues.

- (3) Teachers (and students) usually have insufficient user rights to install software on school computers. Even if they have the required permissions, most PCs are configured to reset themselves to a predefined state with each reboot.
- (4) Similar to other fields, students get homework in their computer science classes and might need to finish a commenced exercise. Of course, they want to continue working on their last progress. This implies two requirements: First, they need access to a comparable development environment on their own PC at home and second, they need access to their previously edited files. The software installation on students' computer might fail with an unexpected error. Copying the required files at the end of a lesson seems trivial, but teachers reported missing Universal Serial Bus (USB) drives or forgotten passwords when trying to send the files via email.
- (5) BYOD policies are difficult to implement due to the financial impact implied for parents and the organizational support required by school administrators if students should use local applications on their PCs. Teachers are not in favor of a BYOD policy as they fear a control loss: On school computers, only educational software is installed (minimizing the possible distraction) and teachers usually have the possibility to lock or inspect the status of a PC, e. g., by using *veyon*³. Using local applications on an own device within computer science lessons might be attractive for students, but imposes technical requirements on the software to feature a local development environment.

For these reasons, teachers prefer solutions making a viable trade-off between the school-provided IT and external resources. Additionally, teachers consider whether and how students are enabled to continue their work at home. As a result, most teachers decide between one of the following three options: First, they could rely on portable software students bring with their own USB drives as this enables students to save their progress on the USB drive and continue working with the same tools on a home PC. A pre-defined set of tools to work from a USB drive is offered by Tino Hempel with his collection *Informatik on Stick* (literally "*Informatics on Stick*")⁴. Second, teachers could use server solutions with web-based development environments (as described in Section 3.6). Third, they could work with traditional software installations and only handle submissions and templates via a USB drive, a network share or another solution (as outlined in the following Section 4.3).

³ <https://veyon.io>

⁴ <https://tinohempel.de/info/info/IoStick/index.html>

4.3 CONTENT DISTRIBUTION AND SUBMISSION HANDLING IN COMPUTER SCIENCE CLASSES

Similar to the distribution of traditional hard copies in lessons, teachers require an adequate way to share digital content with their students. Especially computer science teachers express the need to distribute information to and collect results from their students without relying on printouts. Therefore, an increasing number of schools provide platforms for their students and teachers to support learning with digital resources. Following the definition from Kerres, the integration of digital media (such as those platforms) to distribute learning content is the first step towards e-Learning [30]. Most platforms used in German schools today can be characterized as so-called Learning Management Systems (LMS). According to Paulsen, LMS usually feature organizational services (to manage students, form groups or handle communication) and content services (to distribute learning materials or [homework] submission handling) [49]. Common learning platforms used in Germany include the open-source LMS Moodle⁵ and ILIAS⁶, usually hosted by the schools themselves. Both systems are designed for broad use cases and are also deployed at universities. Other systems focus on schools with their specific requirements, such as the cloud-based product *itslearning*⁷ or the on-premise solution *IServ*⁸. Recently, Microsoft and Google joined with own offers, such as *Microsoft Teams*⁹ and *Google Classroom*¹⁰. However, these are not widely used in German schools due to privacy concerns expressed by State Commissioners for Data Protection and Access to Information, i. e., from Rhineland-Palatinate (Rheinland-Pfalz) [11]. As a consequence, other LMS, such as *Moodle*, *itslearning* and *IServ* are predominant. Furthermore, the HPI Schul-Cloud is one of the newest LMS and aims to become a centralized hub for external services with the deep integration of existing learning content through a Lern-Store (as mentioned in Section 2.4).

Regarding the content services, all platforms allow teachers to upload learning materials and to create exercises for which students can upload own solutions. Due to the general design that covers common requirements from all school fields, no solution offers special support for programming submissions. Students mainly have two choices for transmitting their source code: They could either paste the code in a text box (regardless of the presentation for the teacher) or upload the files manually. In addition, teachers could avoid the platforms

⁵ <https://moodle.org>

⁶ <https://www.ilias.de/en/>

⁷ <https://itslearning.com/>

⁸ <https://iserv.eu>

⁹ <https://www.microsoft.com/education/products/teams/default.aspx>

¹⁰ <https://edu.google.com/products/classroom/>

for submission handling and use a traditional network share or **USB** drive. All three solutions have several shortcomings:

- (1) Textual submission: Teachers need to copy and paste the solutions of their students manually and save the resulting files locally on their computer. In the second step, they can compile and run the code to observe the programmed behavior. Even though some steps might be automated by an **IDE**, the teachers we interviewed find these steps annoying and error-prone.
- (2) File upload: When handling each submission individually, downloading and running the students' source code is still a hassle. The downloaded files might either have unique names making it more difficult to handle with automation tools or have the same names, which might lead to name clashes in the teacher's download folder. For some languages, such as Java, the file name of a class is important and unintended changes should be avoided.
- (3) Network file share / **USB** drive: Some teachers prefer to use a network file share or a **USB** drive to collect submissions as students help in organizing their own solutions. However these approaches have the potential disadvantage that students might mutually copy or alter their submissions. Preventing that through explicit file permissions requires additional preparation of teachers.

Furthermore, some programs might consist of two or more source files where at least parts of the sample code should not to be edited by the students. In such a case, teachers might want to ensure that their (unaltered) version of the respective source code file is used, making manual file handling even more complex. As executing student-generated source code is time-consuming, teachers mostly refuse to do so. They either only provide feedback from reading the code or do not collect submissions at all. In the latter case, they can only encourage students to point out problems themselves. However, our interviewees reported that usually the more reserved students will not seek help themselves. *GitHub Classroom*¹¹ uses the version-control system *Git*¹² to handle submissions but requires knowledge about the functioning of a version-control system which most high-school teachers do not introduce to their students.

4.4 WORKSHEETS IN COMPUTER SCIENCE CLASSES

Independent of the possibilities teachers have to distribute content through a **LMS**, many computer science teachers prefer preparing worksheets as known from other fields. **Figure 4.1** depicts one of those traditional worksheet with a programming assignment designed by a

¹¹ <https://github.com/education/classroom>

¹² <https://git-scm.com>

Info 12.1LK Soose
Zusatz-Übung
S. 1

1. Eine selbst definierte Klasse
 Es soll eine Variante des Knobelspiels "Stein, Schere, Papier" programmiert werden. Der Spieler erhält zu Spielbeginn einen Bonus von 10 Punkten. Zunächst entscheidet man sich für einen Spielzug: Stein, Schere oder Papier. Wird die Schaltfläche *OK* betätigt, nimmt der Computer ebenfalls einen Spielzug vor. Dieser Spielzug wird ausgegeben, beide Spielzüge werden verglichen¹ und der Punktstand wird neu berechnet und ebenfalls ausgegeben. Verliert der Spieler diese Runde, wird die Punktzahl um einen Punkt vermindert, sonst um einen Punkt heraufgesetzt. Sind 20 Punkte erreicht, wird eine Meldung ausgegeben, dass das Spiel gewonnen wurde, bei 0 Punkten erfolgt ein Hinweis auf das Verlieren des Spiels.
 Die Auswertung des eingegebenen Zuges soll in der selbst definierten Klasse *TSpiel* erfolgen.

a) Erstellen Sie die fehlende **Unit USpiel**, die die Klassendefinitionen enthält.
 Die notwendigen Dateien finden Sie im Ordner *Knobeln* auf *AlleUser\Soose*, wo sich auch ein Testprogramm (EXE) befindet. Alle weiteren Informationen bzw. Vorgaben entnehmen Sie den folgenden Texten und Schaubildern. **Der private- und der public-Bereich bleiben in USpiel leer!**
Beachten Sie bitte, dass der Computerzug als String vom Spielobjekt geliefert wird. Bei der Methode handelt es sich also um eine Funktion!

b) Die Eigenschaften *Punktstand*, *Computerzug* und *Spielerzug* (alle vom Typ *Byte*) werden innerhalb der Klasse *TSpiel* als **privat** markiert. Dadurch kann auf diese von außen nicht zugegriffen werden. Wie muss die Klassendefinition geändert werden, wenn die *Unit UKnobeln* nicht verändert werden soll? Schreiben Sie dazu eine **Unit USpiel2** und begründen Sie die Änderungen!

```

unit uKnobeln;
interface
uses SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics, Controls,
Forms, Dialogs, StdCtrls, ExtCtrls, Menus, USpiel;
type
TFAnwendung = class(TForm)
LPunkte: TLabel;
EPunkte: TEdit;
RGAuswahl: TRadioGroup;
BOKay: TButton;
LComputerzug: TLabel;
EComputerzug: TEdit;
Button1: TButton;
procedure FormCreate(Sender: TObject);
procedure BOKayClick(Sender: TObject);
procedure FormDestroy(Sender: TObject);
procedure Button1Click(Sender: TObject);
private
Spiel : TSpiel;
end;
var
FAnwendung : TFAnwendung;
implementation
{$R *.DFM};
procedure TFAnwendung.FormCreate(Sender: TObject);
begin
Spiel:=TSpiel.create;
Spiel.init(10);
RGAuswahl.ItemIndex:= -1;
EPunkte.Text:=IntToStr(Spiel.Punktstand);
end;
procedure TFAnwendung.BOKayClick(Sender: TObject);
begin
if RGAuswahl.ItemIndex = -1 then
showmessage('Kein Spielzug angegeben! ');
else
begin

```

Klasse *TSpiel*

Eigenschaften

- Punktstand (Byte)
- Computerzug ∈ [0,1,2] (Byte)
- Spielerzug ∈ [0,1,2] (Byte)

Methoden

- Spielbeginn (der Spieler erhält z.B. 10 Punkte)
- Computerzug bekannt machen (als String)
- Spielzüge auswerten

Stein, Schere, Papier

Spielzug
 Stein OK
 Schere Ende
 Papier

Computerzug:

Punktstand:

1 Schere schlägt Papier, Papier schlägt Stein, Stein schlägt Schere

Figure 4.1: A traditional worksheet for a computer science lesson [59]. It is intended for high-school students in their second-last year and focuses on creating a custom class for the rock-paper-scissors game. The worksheet includes an assignment to create the custom class based on a simplified UML diagram of that class. Moreover, the remaining program is scaffolded and printed on the page together with a screenshot of the running program.

high-school teacher. For the creation of worksheets, our interviewees highlighted the possibilities to (re-)use fitting content from websites and other resources. Besides identifying the aspects they like and dislike about their current situation, we also asked them to share some

of their current worksheets with us. Finally, we distilled three main issues of traditional worksheets regardless of the distribution form:

- (1) Lack of interactivity: Some use videos or small online games, such as Pac-Man, to motivate students to work on a given task related to what they've experienced in the interactive content (e. g., a path-finding algorithm for ghosts in the context of Pac-Man). To allow students to work on the most challenging part, teachers then provide a framework with a few source code files. In this scenario, they distribute the source code via a file server together with a **Portable Document Format (PDF)** of the worksheet with embedded links to an instance of the online game or a video hosted online (e. g., on YouTube).
- (2) Lack of adaptability: Others use content from **MOOCs**, e. g., the programming courses on Java and Python available on openHPI with programming exercises in CodeOcean. While teachers like the approach used in these online courses, they wish to have the opportunity to customize exercises or to add additional references to the schoolbook available to the class. When it comes to changing the exercises, teachers want to adapt the exercise description and the scaffolded code, as well as the automated feedback the platform provides to learners. In their experience, students focus on achieving a maximum score while sometimes losing focus of the actual learning goals. Consequently, teachers might either accept exercises as they are or manually copy them and lose key features of the platforms, such as automated feedback.
- (3) Lack of usability: Some other teachers remarked on the disadvantages of traditional submission systems based on uploading and downloading files by hand. They require too many manual steps to be practical for use in lessons.

Analyzing the shared worksheets, we further derived other needs not directly mentioned by the interviewees. For example, some teachers distributed their worksheets as editable text documents instead of **PDFs** as they included open questions that should be answered by the students in-line on the worksheets. These teachers need support to provide each student with an individual, editable copy of the digital worksheet. Others choose to put the content in **LMS**, such as a *Moodle* instance, that also allows content creation and submission handling. With an **LMS**, open text questions might be asked together with simple multiple-choice quizzes allowing the automated generation of feedback for students. According to the teachers we interviewed, *Moodle* is seen as powerful, but also has a reputation for being too complicated for all parties involved. Concerning computer science classes, they still do not fit all the requirements teachers have, hence dedicated assistance is needed. By looking at the exercises included in

the worksheets, we also learned that teachers sometimes wish to provide a full working source code example allowing students to observe the execution flow or a programming pattern used. Presenting the code on PDFs or printed worksheets without the possibility to execute it, gives the teacher control over the level of detail and granularity but requires some level of imagination from students to understand what the given code snippet does. Providing the full source code as downloadable files enables students to execute it but reduces the influence a teacher has on the presentation of the relevant code in question as, for example, no code could be hidden.

All current solutions used by teachers in-class to distribute content and assignments have some drawbacks. Most of them are caused by the decoupled source code and other learning materials, resulting in the need for a context switch when working on a given topic. If some of those drawbacks are addressed, for example by using MOOCs and their integrated programming environments, teachers have to accept their decreased capabilities in editing the provided content and the lack of access to their students' submissions and progress. Consequently, there is currently no option that fully meets the needs of teachers.

4.5 EDUCATIONAL IDES TAILORED FOR BEGINNERS

Besides organizational difficulties, computer science teachers face challenges in terms of content and didactics (as outlined in Section 4.1). A major decision they make is about the IDE used in class. In many cases, teachers prefer educational IDEs (as described in the following) over traditional, feature-rich IDEs due to a simpler feature set targeting novices. Examples for educational IDEs include *BlueJ*¹³, which the authors describe as “a powerful graphical shell / REPL for Java”. It also includes a graphical representation of classes and objects. *Greenfoot*¹⁴ builds on top of *BlueJ* and provides a graphics / simulation Application Programming Interface (API). It therefore helps students to visually understand the behavior they programmed. The teachers we interviewed appreciated the simplification offered by both but also noted a missing link to “real-world” software development due to the abstractions included. Other tools support specific tasks: *Processing*¹⁵ for example features simple 2D and 3D visualizations in the context of a “flexible software sketchbook” according to the website. In this regard, it is similar to Jupyter Notebooks¹⁶, an interactive, web-based

13 <https://bluej.org>

14 <https://greenfoot.org>

15 <https://processing.org>

16 <https://jupyter.org>

REPL with support for visualizations. *Thonny*¹⁷ is an open-source Python programming environment inspired by traditional **IDEs** that is designed for beginners and features an integrated debugger. The most comprehensive educational **IDE** used by teachers is the *Java-Editor*¹⁸ by Gerhard Röhner that also integrated an **Unified Modeling Language (UML)** editor. On the one hand, teachers like the many features but on the other hand, they also find them confusing for novices. Additionally, the *Java-Editor* also visualizes code blocks and an integrated debugger, as shown in [Figure 4.2](#). Most educational **IDEs** are stand-alone applications and cannot be used through a web browser even though many teachers would prefer a cloud solution due to decreased maintenance efforts and access from any internet-enabled device.

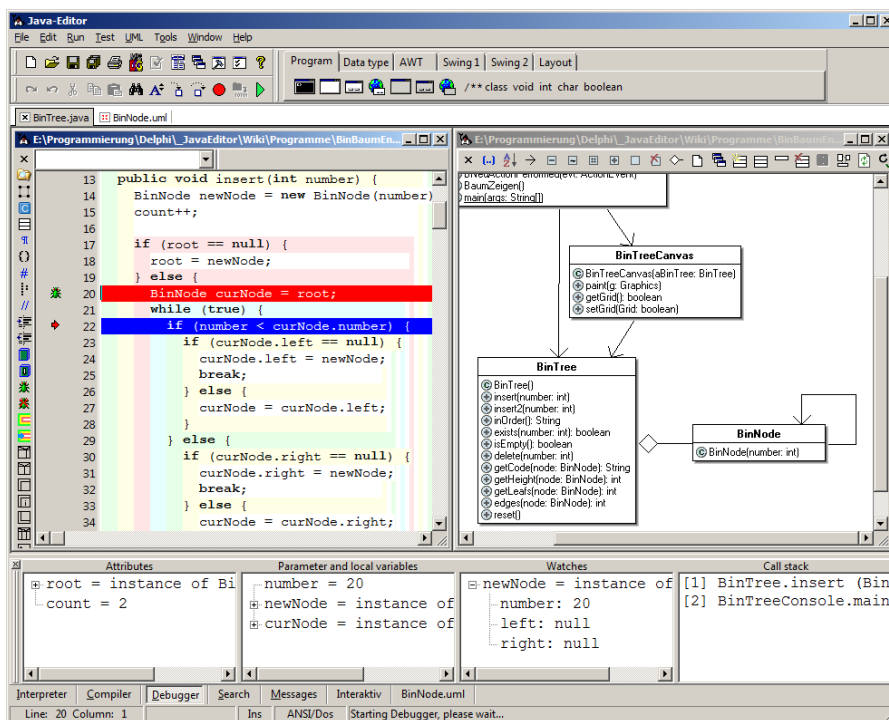


Figure 4.2: Gerhard Röhner’s *Java-Editor* debugging an object-oriented Java program [57]. The editor shows the source code on the left (with additional visualization of code levels), the corresponding **UML** diagram on the right and debugging information at the bottom.

¹⁷ <https://thonny.org>

¹⁸ <http://javaeditor.org>

4.6 IMPLICATIONS FOR COMPUTER SCIENCE TEACHERS

The previous sections describe learnings from the interviews we had with thirteen computer science teachers, two principals and five students. While the requirements teachers described are similar, the approaches they choose to fulfill these differ. We extracted the following requirements for tools used in computer science classes:

- (1) Teachers wish to mix existing content (e. g., from open books and from MOOCs) and use the most fitting learning materials for their students.
- (2) Teachers want to adapt the learning materials to their needs and add own content.
- (3) When providing practical exercises, the task should be linked to the source code.
- (4) The programming environment should be simplified to be suited for beginners but must not limit language features.
- (5) Submission handling is crucial for students and teachers in computer science lessons and should be supported by the programming environment.
- (6) Access to the programming environment should be simple and independent of the device used.
- (7) Any tool used for teaching in school needs to be technically stable and in compliance with legal regulations (i. e., data protection laws).

In this thesis, we develop a concept (see [Chapter 5](#)) that is based on these requirements. Some tools, for example open book websites, provide some of the advantages (such as serving content), but teachers cannot link to those from development environments or edit the content directly. IDEs on the other hand provide the development tools but without linking to the content and, in most cases, without submission support. Traditionally, teachers create worksheets with assignments (as known from other fields) and accept some of the shortcomings due to missing tools and time. Therefore, our approach is to build a novel kind of digital worksheets with embedded programming exercises to tackle these issues and provide a seamless experience for students and teachers.

CONCEPT

Our university is one of the [MOOC](#) providers in Germany offering free entry-level programming courses online and is regularly contacted by teachers expressing their interest in using the content (including practical exercises) in their classes. From these contacts and from our initial interviews, we know that teachers often wish to further provide own content and adapt exercises to their specific needs. Therefore, we developed a concept that allows teachers to reuse and adapt content from [MOOCs](#) regardless of its type (e. g., introductory texts, videos, quizzes or programming exercises) and enrich it with their own exercises or other references.

In the following sections, the concept of interactive worksheets with programming exercises is introduced:

- (1) The general design of interactive worksheets, which is described in [Section 5.1](#), is based on a modular editor with support for different content types on a single webpage. The proposed worksheet editor follows a **What You See Is What You Get (WYSIWYG)** editing approach with customized views for teachers and students. Interactive worksheets support multiple content types including rich text, embedded images and videos as well as multiple choice quizzes and interactive programming exercises.
- (2) In order to find or create programming exercises for use with their class in an upcoming lesson, teachers need access to CodeOcean. Inspired by [MOOCs](#) and web-based code execution platforms (see [Section 3.6.3](#)), teachers appreciate automated feedback that is given to students as it helps learners to identify mistakes. Hence, teachers want to include automated feedback for their own exercises as well. [Section 5.2](#) introduces the implications these requirements have to CodeOcean and outlines how CodeOcean exercises could be integrated into worksheets.
- (3) Similar to other materials teachers use during their lessons, students access worksheets (and thus programming exercises) through the [HPI Schul-Cloud](#). As outlined in [Section 5.3](#), CodeOcean exercises are linked from the [HPI Schul-Cloud](#), which is required to recognize returning learners and allow proceeding previous work. Additionally, CodeOcean depends on information about students and their membership in classes to provide teachers with meaningful learning analytics of their students. Transmitting user information from the [HPI Schul-Cloud](#) to the CodeOcean

server must follow a privacy-by-design approach and is limited to pseudonymized user information.

- (4) Further, we tackle the problem of explicit submission handling in our concept. As previously described in [Section 4.3](#), teachers find collecting source code from students time-consuming and the following correction cumbersome and annoying. To address these drawbacks, [Section 5.4](#) describes an integrated submission handling process that minimizes the overhead for students and teachers. Programming submissions are saved automatically to prevent unintended data loss and serve as a foundation for learning analytics.
- (5) While teachers find it beneficial to access submissions manually for grading-related tasks, it is no viable approach for them to get a quick overview, especially during lessons. Hence, access to automated learning analytics is required, which is introduced in [Section 5.5](#). Teachers wished to get insights about the working time in correlation with the progress achieved by their students. Based on the requirements expressed by teachers, the learning analytics allow a simple comparison to [MOOC](#) participants and feature a summary for teachers to be used during lessons.

5.1 GENERAL DESIGN OF INTERACTIVE WORKSHEETS

Our vision is that teachers are enabled to mix existing editable content in worksheets with their own additions. Traditionally, teachers create those to provide students with tailored learning material adapted to the current situation of the class. In paper-based worksheets, common components are text, images and graphics. Special editing tools support teachers in creating those paper-based worksheets. *tutory*¹, for example, is a web platform dedicated to edit worksheets intended to be distributed as printouts to students. *tutory* focuses on editing worksheets and includes (design) templates for definitions, images, references and tables as well as tasks with examples for multiple choice quizzes, clozes and crossword puzzles. The resulting documents can be shared with other teachers and downloaded as static [PDF](#) for printing. Thus, students work with hard copies and the worksheets do not contain any interactive elements. For computer science lessons, the worksheet editor *tutory* lacks specific support, so that source code cannot be highlighted and needs to be inserted as plain text (an example is given in [Section A.2](#) with [Figure A.9](#)).

Similar to *tutory*, our concept also includes a [WYSIWYG](#) editor with different types of components which can be freely re-ordered to match the teacher's needs. [Figure 5.1](#) provides an example of the

¹ <https://tutory.de>

editor we include in our concept without interactivity yet. Especially for computer science classes with students having access to PCs, an interactive worksheets is beneficial as reasoned in Section 4.4. Besides static text and images, a digital version allows teachers to include videos and interactive multiple choice quizzes as known from MOOCs and LMS, such as Moodle. Furthermore, the interactive worksheets embed a web-based code execution platform featuring the desired programming exercises. Combining learning materials, e. g., through textual descriptions or videos, with programming exercises on a single web page, helps students to revisit explanations while working on the programming exercises. Moreover, teachers have the opportunity to share interactive worksheets and programming exercises themselves with colleagues.

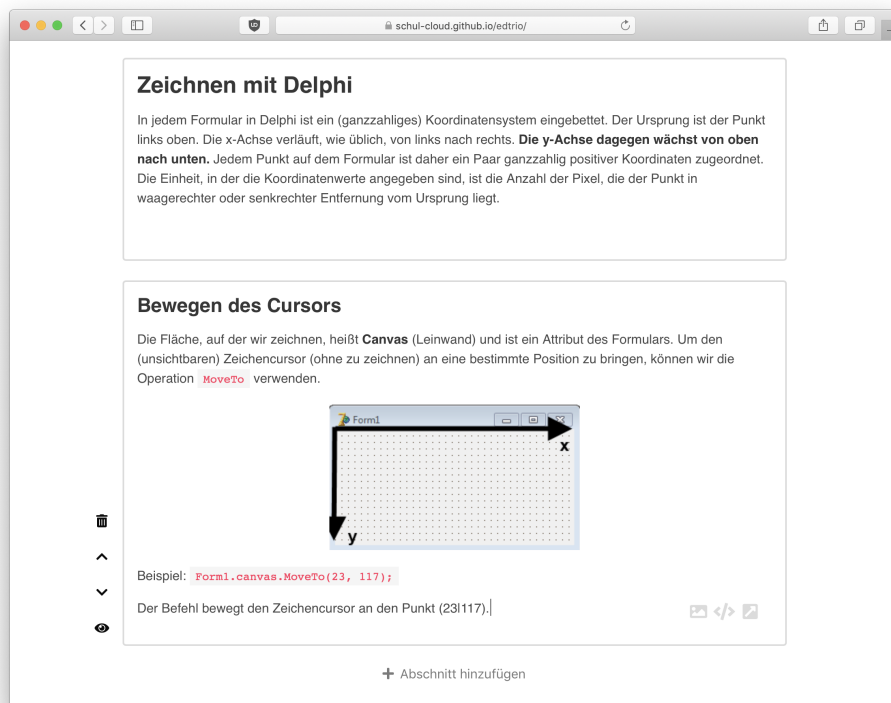


Figure 5.1: Modular worksheet editor as part of our concept without interactive elements. The example introduces recursion in the programming language Delphi [1]. In addition to *tutory*, the editor allows highlighting source code.

While it is easy for teachers to adapt a text on a worksheet (worst case by copying it manually and subsequently changing it), it gets trickier with other content types, regardless of the editor used. Videos can be downloaded and embedded on other sites but editing requires manual labor and some familiarity with video editing tools. In fact,

none of the teachers we interviewed was interested in editing a video. Either, they embed the full video as it is, or they do not use it. Even specifying a range of the video and thus embedding only a short, continuous part is rarely used. Some thought about recording their own videos but did not find enough time to prepare those. Therefore, we concluded the need to support embedding videos from MOOCs and also videos from common video hosting platforms, such as YouTube² or Vimeo³. For quizzes, two major use cases were distinguished: Either (1) self-tests for the learner or (2) graded or ungraded assignments with results being checked by the teacher.

Major challenges for teachers are the embedding and customization of programming exercises from MOOCs. For that, we decided to show exercises on the same page as other elements to reduce context switches and to fit our worksheet analogy. As a foundation of our concept, we built on top of the existing work that was done in the context of the HPI Schul-Cloud. This brings us several advantages, as it provides a home for the content creation and helps teachers currently working with the HPI Schul-Cloud to try our prototype in their lessons. Further, the research described in this thesis is coordinated with the future development of the HPI Schul-Cloud and their roadmap. Offering a free combination of learning materials, it is possible to replace other forms of worksheets whether analog, PDF- or text-based. For each content type, a plugin is provided. The default set of plugins offered by our worksheet editor includes those required to display images, embed videos or create multiple-choice quizzes. Figure 5.2 visualizes a document created with our concept of an interactive worksheet editor. It uses the default set of plugins and further embeds a practical programming exercise to consolidate concepts learned.

5.1.1 *Different Views for Students and Teachers*

Teachers and students use the same tool to access the worksheets. While teachers have access to the edit mode, students are limited to the view mode that prevents changes to the structure of the document but allows usage of interactive elements within the worksheet. The edit mode for teachers follows the WYSIWYG approach and thus provides teachers with the look and feel of the final document. Due to the concept of providing sample solutions for interactive worksheets (allowing students to get automated feedback), teachers can also preview the document from the students' perspective. This change in perspective helps teachers to distinguish which information is visible to students. Besides hiding solutions, no major difference between the

² <https://youtube.com>

³ <https://vimeo.com>

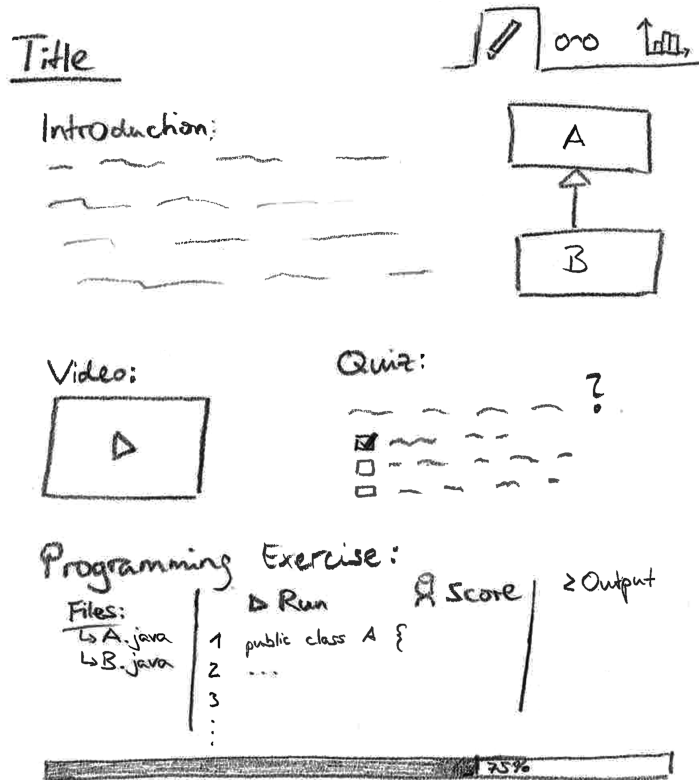


Figure 5.2: Wireframe of an interactive worksheet. It contains a textual introduction to a topic, a visualization (in form of an [UML diagram](#)) and a video. Additionally, it features a multiple-choice quiz and a programming exercise. At the bottom, the overall progress (in terms of the score achieved) is shown. The drawing illustrates the teacher role so that a switch to the preview mode and analytic results is possible (top right corner).

editing and viewing context exists. The control elements within the editor are of secondary importance and are hidden unless needed.

During lessons, teachers and students access the worksheet simultaneously. Each student gets an individual instance of the document so that resuming work is possible. Within the worksheet, teachers have access to submissions of their students ([Section 5.4](#)) and learning analytics insights ([Section 5.5](#)). The access to submissions can either be used for grading tasks or to discuss solutions and emerging problems during a lesson. As shown by Janke, a dedicated presentation view supports teachers in using a projector to show student results in front of the class [28].

5.1.2 Accessibility of Content for Students

If designed appropriately, the worksheets contain all information required for students to proceed in learning the topic and solving the embedded assignments. The worksheets are self-contained and separate topics from each other. Together with additional resources (e. g., a programming language reference), students can learn at their individual pace. However, teachers might sometimes wish to have fixed checkpoints to compare results or discuss problems before students proceed with working on the assignments. Traditionally, teachers would either split the worksheet into multiple parts or ask students to ignore sections of it until it is used later during the lesson. With the interactive worksheet however, teachers get control over the visibility of content blocks on the worksheet. One or more elements in the editor, such as text, videos or programming exercises, form a content block that teachers can hide or unhide while students are working with the worksheet. Thus, teachers have a higher level of control during their lessons and can adapt changing situations rapidly: Teachers could prepare a component with additional content and use it only if they think students will benefit from it. Whenever desired, teachers can control the visibility of new content and navigate to other sections of the document. Arndt introduces the concept of a Schul-Cloud Cockpit to provide teachers with a detailed overview of the student progress together with the current tasks visible to them (prototype shown in [Section A.2](#) with [Figure A.11](#)) [2].

Besides teacher-unlocked components, some interviewees proposed auto-unlocking of content based on progress of previous sections. For example, a worksheet might contain an introduction and two programming exercises on a given topic. Only if both assignments were solved by the student, a final multiple-choice quiz could become visible to recapitulate learnings from the practical programming exercises. To minimize student frustration whenever new content appears after finishing a task, an overall progress indicator is present on the worksheet. Combined with the direct control teachers have about the visibility of content, many different learning scenarios can be created for use during lessons and at home as part of student homework. Auto-unlocking new content based on student progress is also part of the research conducted by Håklev et al. to support collaborative learning scenarios. Their tool *FROG*, which is shown in [Figure 5.3](#), implements Orchestration Graphs that “depict the structure (what is done when by whom)” of a learning scenario and thus can be used to enable auto-unlocking of new content [23]. Overall, three visibility options are available: Always visible, teacher-controlled and auto-unlocked through previous assignments.

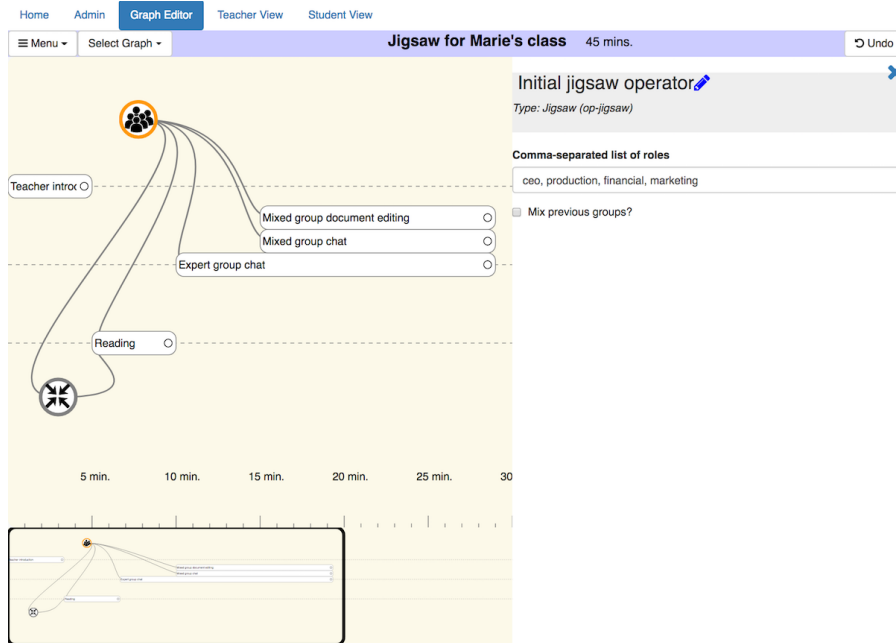


Figure 5.3: The *FROG* editor used to create an Orchestration Graph for a lesson [22]. The lesson planned includes time for Reading, which students need to finished before they continue with the next assignment.

5.2 EDITING PROGRAMMING EXERCISES IN CODEOCEAN

In addition to a suitable content editor, teachers also need direct access to a code execution platform, in our case CodeOcean. Up to now, CodeOcean was designed with MOOCs in mind, resulting in a simple authentication model of administrators drafting exercises or managing the platform and students using the system to solve exercises without any backend access. In CodeOcean, each exercise comprises one or more files visible or editable by the user, which already contain some scaffolding, and hidden unit tests to check the correctness of the student's submission. The result of a test-run is then used to provide automated feedback to students and to direct them to the mistakes they made. For the desired use in class, this is not sufficient as it completely omits the teacher role. Therefore, our concept implies that CodeOcean is extended to support a third role, called teacher. It allows teachers to (1) create new exercises that (technically) do not differ from existing exercises created by MOOC instructors and (2) browse a list of public exercises and clone those if desired. A wireframe outlining the process of editing exercises is shown in Section A.1, Figure A.1. Given these changes, CodeOcean can become the starting point for teachers to build a repository of exercises. These might then be shared using CodeHarbor, a platform designed to find, share and

improve exercises which are fully compatible with CodeOcean [61]. For learners belonging to a class the teacher educates, the teacher is enabled to view learning analytics and access student submissions on demand.

5.2.1 Automated Feedback through Unit Tests for Exercises

During our initial interviews, we found that providing the same tools that MOOC instructors use to create exercises including unit tests for automated learner feedback, is not sufficient and suitable for teachers. Many teachers we consulted either did not know about unit tests at all or admitted that they are unable to write their own tests. As a result, none of our interviewees is willing to invest the time and effort to create tests for their own exercises. On the other side, teachers described the feedback given to learners on the basis of the tests as one of the biggest advantages CodeOcean offers when compared to other solutions.

Therefore, the usability of the platform with own exercises might be further improved with CodeOcean offering a (simple) unit test generator (fully outlined in Figure A.8, Section A.1). By analyzing exercises used in MOOCs, we found a few general test cases that repeated over and over again: In Java, typical tests compare (1) the command line output, (2) the content of a source file or (3) a return value from a method against a specified **Regular Expression (Regex)**. Most parts of a unit test are similar and related to getting the desired string or executing the code. We are confident that teachers benefit from a test generator that simply asks them what to compare against which expected result and auto-generates the code required to perform this task. Writing a **Regex** that is general enough to include all possible solutions but as strict as possible to prevent mistakes being misidentified as solution is difficult, as MOOC instructors acknowledged. Therefore, teachers need further assistance in finding the correct **Regex**. Some tools, such as *regex101*⁴, allow users to provide a **Regex** together with multiple examples and highlight the matched parts visually (an example is provided in Section A.2 with Figure A.12) [44]. While being a help mainly for professionals, users still need to write the **Regex** themselves. Other approaches include generating the **Regex** from examples, an approach explored by Bartoli et al. to generate a **Regex** based on a provided pair of strings and desired matches [3]. In this thesis, we concentrate on simple **Regex** proposed by the unit test generator with a few options to specify the case-sensitivity or whether a **Regex** should match the string fully or partially.

⁴ <https://regex101.com>

5.2.2 Referencing Exercises from Worksheets

Programming exercises for worksheets need to be created on CodeOcean before being embedded into a document. Therefore, teachers need access to CodeOcean outside of the worksheet editor to create new exercises, browse existing ones and try them out for themselves. Once they are satisfied with an exercise, they can embed it into a worksheet (cf. [Figure A.2](#) in [Section A.1](#)). The embedding is either done by manually copying the exercise Identifier (ID) to the worksheet editor or by selecting the corresponding exercise from a list offered by the integration of CodeOcean in the worksheet. Teachers might either include a specific exercise or let CodeOcean recommend a fitting exercise (through a mechanism called *Proxy Exercise*). When learners access one of those exercises for the first time, they are served with a specific exercise that is most suitable for them, e. g., to repeat a concept they previously struggled with. The exercise recommendation system was introduced by Teusner et al. for bonus exercises in programming MOOCs and also helps teachers to support their students best with targeted and individualized worksheets [64]. Therefore, worksheets with recommended (bonus) exercises and automated feedback are a first step towards internal differentiation as further described in general by Arndt in his Master's Thesis [2].

5.3 DEEP INTEGRATION WITH THE HPI SCHUL-CLOUD

The interactive worksheets are integrated into the [HPI Schul-Cloud](#) which frames the content available through worksheets. As outlined in [Section 2.4](#), the [HPI Schul-Cloud](#) is structured in courses that have multiple content topics. Each worksheet is assigned to a specific topic and is therefore accessible via that topic. Furthermore, the editor uses the surrounding infrastructure as offered by the [HPI Schul-Cloud](#) eliminating the need for explicit authentication and authorization. Arndt describes the general integration concept regarding the persistence of documents and separation of users in more detail [2]. As a result of his concept, the worksheets are user-aware and can associate progress with individual learners. Within the worksheets, programming exercises are provided by CodeOcean. For meaningful learning analytics within CodeOcean, the course context of a learner is required which is therefore passed by the [HPI Schul-Cloud](#) to CodeOcean. Hence, CodeOcean can form study groups including the teacher and all students of the course automatically with no need of manual work.

5.3.1 Pseudonymization

From the very beginning, the [HPI Schul-Cloud](#) was designed in accordance with German privacy protection laws in mind, such as the *Bundesdatenschutzgesetz*, the German *Federal Data Protection Act* ([BDSG](#)) and the European **General Data Protection Regulation** ([GDPR](#)). For example, Article 5 of the [GDPR](#) introduces the *data minimisation* principle to restrict the use of personal data to a minimum required to provide a service [14], which the German [BDSG](#) further restricts [18]. Consequently, the [HPI Schul-Cloud](#) minimizes the transmission of personal data to third parties, including interactive components in worksheets. If some person-related data is required, for example to allow proceeding previous work, a pseudonymous [ID](#) is used that differs for each user and service combination. In particular, the [HPI Schul-Cloud](#) does not expose user or email information to any third party without previous consent.

When students access a programming exercise in CodeOcean embedded into a worksheet, the same mechanism applies: CodeOcean only obtains a pseudonymous [ID](#) to identify returning users. For learning analytic insights offered to the teacher, the pseudonymization is problematic. As long as CodeOcean only has access to an [ID](#), the platform is unable to display real student names to teachers in the analytics. Therefore, a de-pseudonymization approach is needed and should ideally revert the pseudonymization within the teacher's web browser or through a proxy to prevent names being processed by the CodeOcean server [51]. In their paper, Renz and Meinel focus on generalizable de-pseudonymization strategies, which only require dedicated support for the pseudonymized usage of external tools, such as CodeOcean but no major architectural changes for the de-pseudonymization [51].

5.3.2 Customization of the CodeOcean Integration

In our concept, the integration of CodeOcean exercises within the [HPI Schul-Cloud](#) is highly customizable and thus supports teachers in various different teaching scenarios. Without any further explicit settings, all features available in CodeOcean are enabled for students working on a programming exercise. For some teaching scenarios, however, high-school teachers do not want students to use the full potential offered by CodeOcean. For example, some teachers wish to provide students with a non-modifiable version of a program. Thus, students can execute the code and observe the programmed behavior without accidentally changing it. By combining a read-only (but executable) version of exemplary source code together with an

exercise on the same worksheets, students are encouraged to apply knowledge they got from the example in their own code.

Furthermore, some of our interviewees wished to disable the ability of CodeOcean to display test results or scores to prevent that students lose the actual learning goals (cf. [Section 4.4](#)). Each of the main functionality in CodeOcean, such as posting a *Request for Comment* or running the code is individually configurable by teachers. If desired, teachers can combine multiple of these settings to minimize the tool support offered by CodeOcean. By disabling many of the features offered by CodeOcean and combining them with the submission handling, the way is paved for teachers to use interactive worksheets as (graded) homework assessments. Based on the interviews we had with teachers, the main features that need to be disabled are those to run, download, and score the source code as well as additional help offered by CodeOcean through automated hints or details about failed tests. [Figure 5.4](#) depicts the automated hints as an example of a feature teachers wish to control for their students. A wireframe outlining the integration is included as [Figure A.3](#) in [Section A.1](#).

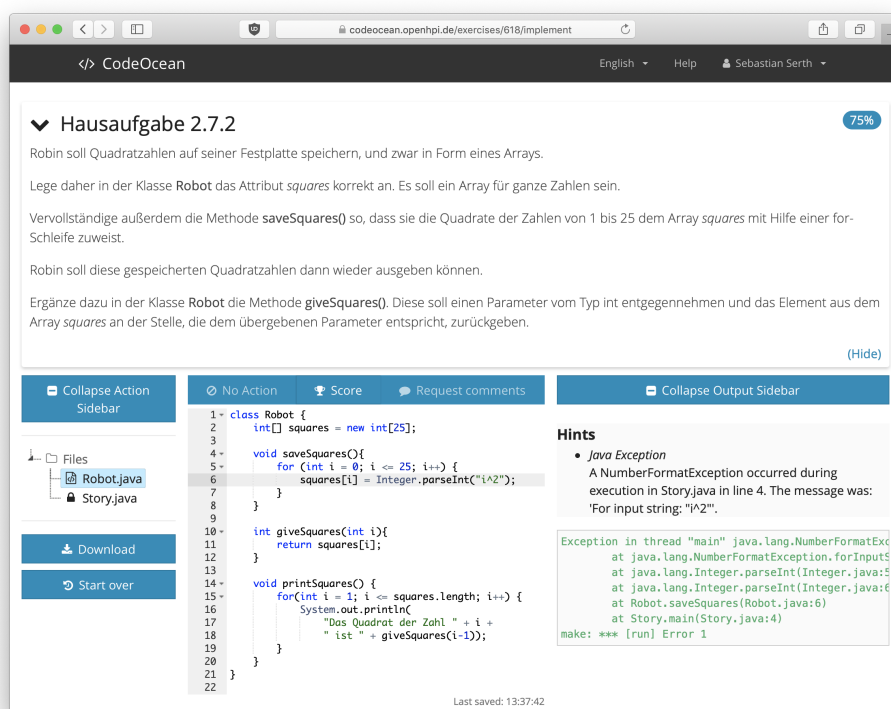


Figure 5.4: Automated hint provided by CodeOcean to help learners identifying mistakes in their source code. The hint is displayed above the console output and rephrases the Java exception. Showing hints is one of the features teachers wish to disable from time to time.

5.3.3 Worksheet Sharing and Content from MOOCs

According to Keutel, course planning and preparation is a time-consuming process for teachers [31]. In his Master's Thesis, Keutel studied how teachers can be encouraged to share learning material using the *Lern-Store* available in the HPI Schul-Cloud (cf. Section 2.4). While the *Lern-Store* is, so far, only available for external resources, Keutel expands the concept to also include teacher-generated content. The interactive worksheets with videos and programming exercises fit into that concept and thus can be shared using the approaches available within the HPI Schul-Cloud with others. Additionally, existing content from MOOCs can be grouped as a worksheet and can be included in the *Lern-Store* to help teachers in reusing content from online courses.

Sharing a complete worksheet with programming exercises preserves the context of individual components and provides teachers with a suitable introduction for the practical programming assignments. Extracting single programming exercises could take them out of context. Thus, the preferred sharing approach for worksheets is the general-purpose *Lern-Store* within the HPI Schul-Cloud. A more detailed sharing on exercise level (instead of worksheet level) is available within CodeOcean. Teachers have the opportunity to browse a list of exercises and make their own creations public for reuse by other instructors (see Section 5.2).

5.4 IMPLICIT SUBMISSION HANDLING

Similar to traditional, paper-based worksheets, the interactive worksheets do not require a learner to explicitly save the progress. Instead, the progress is saved automatically, depending on the plugin type in which the change occurred. Answers given to multiple choice quizzes are transmitted to the server immediately and changes in editable source code files are saved after a few seconds of inactivity or whenever the user compiles and executes the code. Auto-saving the learners' progress prevents any data loss which could otherwise be caused by an unresponsive (school) computer. In addition, there is no need for students to flag their final version or submit it explicitly. CodeOcean will always display the latest version for teachers on demand and also use that for the generating test results (see Section 5.4.1). Additionally, a snapshot of the source code together with information about failing unit tests is being generated in case the student uses the *Request for Comment* feature on CodeOcean.

5.4.1 *Pre-Evaluation of Submissions*

Collecting student submissions through an adequate submission system is only the first step for teachers to review the collected solutions. By using the unit tests defined together with each exercise, CodeOcean is capable of providing teachers with a first impression of the correctness of student submissions. Depending on the quality of unit tests available, this pre-evaluation might be close to a manually scoring. The rough estimation generated by the execution of unit tests provide teachers with a hint where to concentrate on during their review. In our concept, the pre-evaluation is done based on the last submission a student did and independent of the visibility of test results to learners. However, if test scores are shown to students, they reflect the same score that teachers would see. Future work might include tests whose results are exclusively visible to teachers.

5.4.2 *Time Traveling to Understand the Learner's Approach*

As elaborated in [Section 3.5](#), the history of source code changes provides an additional window into the student's cognition and helps educators to understand the approach students used to solve the given exercise [9]. Based on scoring runs, CodeOcean allows teachers to review the evolution of their students' source code on demand and upon approval. The platform lists all intermediate submissions together with the corresponding unit test results. Inspired by a time-travel, teachers can follow the progress of individual students afterwards with an interactive playback of the changes in the source code files. Intended to be used occasionally, the time-travel feature is designed to help teachers in identifying the root cause of problems and to tackle existing misunderstandings of their students individually.

5.5 LEARNING ANALYTICS

Besides providing a tool for creating and delivering content, our concept includes a deep integration of learning analytics. As mentioned in [Section 4.4](#), this data is usually not available to teachers even though they are likely to benefit from a view on prepared data. We designed the architecture of our digital worksheets to support learning analytics per plugin instance (meaning *per exercise* for programming exercises) and a high-level overview of the whole worksheet. The teacher's view will show a short excerpt of key metrics below each part of the worksheet (as sketched in [Figure 5.5](#)) to give teachers a quick overview of the current status of their students. Further, a detailed analysis per programming exercise should also include an overview of time spent

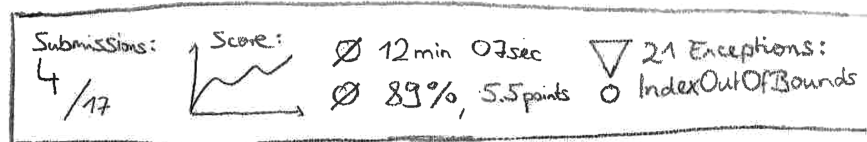


Figure 5.5: Wireframe of key metrics shown below each part of a worksheet. Teachers get the most important information regarding the status of an exercise with a glance.

by students to solve an exercise, exceptions, and errors raised during the work and which questions in form of help requests were posted in the context of the given assignment. Figure 5.6 shows a wireframe for a *per exercise* view of learning analytic data that could be embedded into a worksheet (expanded wireframe with additional examples added as Figure A.4 A.5, A.6 and A.7 in Section A.1). Additional to showing the same view students have on a programming exercise, CodeOcean provides a view for teachers highlighting the analytic insights and offering access to student submissions (as described in Section 5.4).

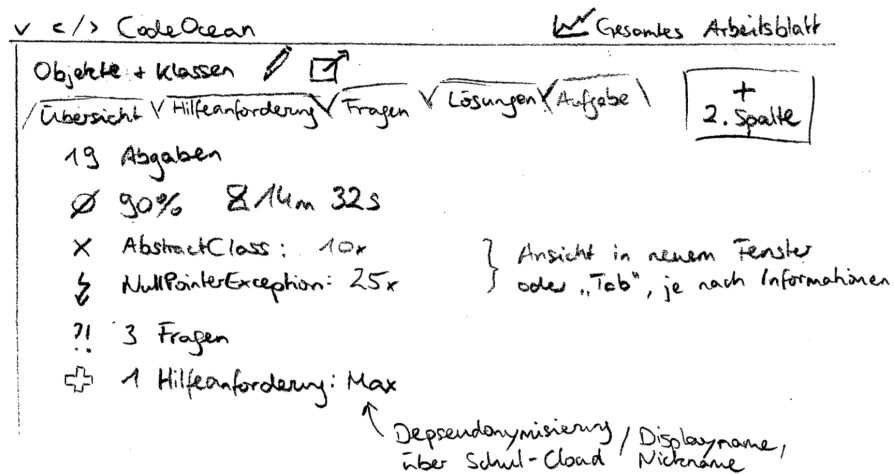


Figure 5.6: Wireframe of learning analytics for a *per exercise* view designed for teachers within a worksheet. The concept features key metrics of student progress, such as a submission count, the average working time, frequent exceptions of code runs and help requests of students.

5.5.1 Integration with External Systems

Based on domain knowledge available, the learning analytics in CodeOcean can include source code specific information, such as common mistakes made by students or exceptions not handled correctly. Furthermore, the *Request for Comments* feature, which supports help requests in the context of code, are exclusive to programming exercises. However, interactive worksheets contain more components than programming exercises, such as multiple-choice quizzes. For a complete view on the progress of the class, teachers wish to get summarized views including all data available. Therefore, CodeOcean transmits selected learning analytic data from students to the surrounding worksheet editor. Thus, the worksheet editor is responsible of processing that data in the overall context of the HPI Schul-Cloud architecture. As outlined by Renz and Meinel in their paper “Can Pseudonymized xAPI-Tracking Solve Data Privacy Issues in German Schools?” and as shown in Figure 5.7, this approach is in compliance with the GDPR and works seamlessly with the pseudonymization engine available within the HPI Schul-Cloud (cf. Section 5.3.1) [51]. Based on the high-level learning data available from various editor components and combined with events from other third parties, the HPI Schul-Cloud is capable of providing a holistic overview for teachers. The aggregated view enables teachers to identify students struggling with the topics and to focus their help to the most problematic content.

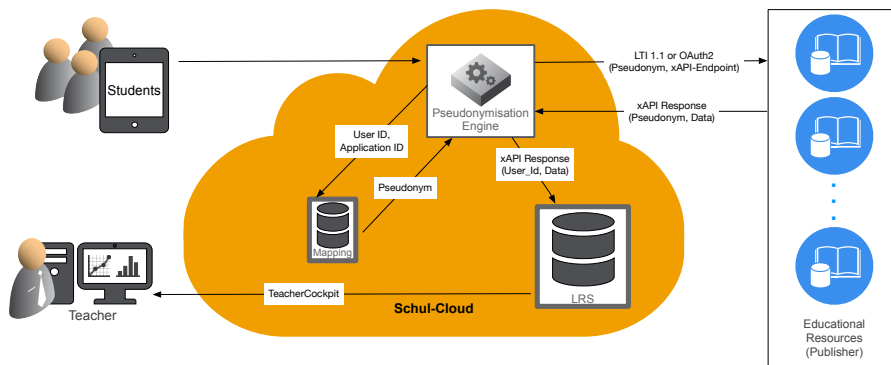


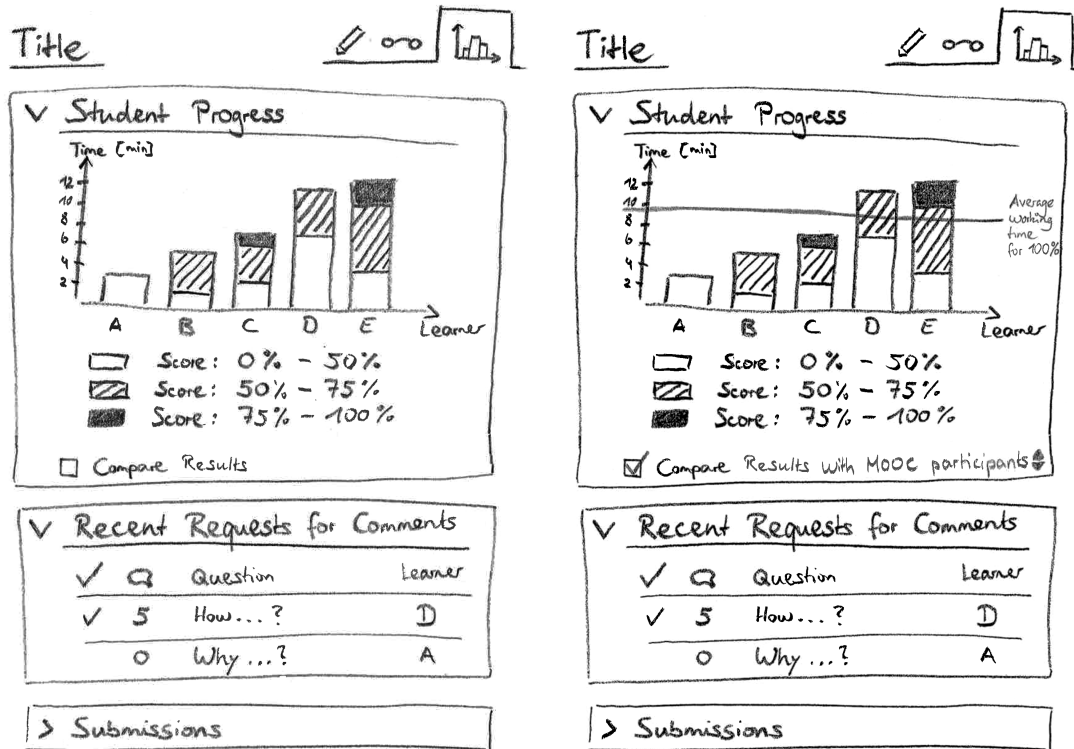
Figure 5.7: The pseudonymization concept designed for transmitting learning analytics to the HPI Schul-Cloud [52]. When students access an external tool, such as CodeOcean, only the pseudonymized ID is transmitted together with an so-called xAPI endpoint for transmitting learning analytics. In our concept, the worksheet editor is the xAPI endpoint. Teachers access learning analytic data, which is stored in a Learning Record Store (LRS) through views in the HPI Schul-Cloud (called *TeacherCockpit*).

5.5.2 *Summary for Teachers During Lessons*

Due to the abstraction and aggregation of data made for the general learning analytics offered by the [HPI Schul-Cloud](#), the resulting overview is designed for long-term analytics and across multiple exercises. During lessons, when teachers provide a small number of exercises to their students, more specific information including the exercise and live progress is valuable to teachers. Similar to the [HPI Schul-Cloud Cockpit for worksheets](#) as described in [Section 5.1.2](#)), our concept includes a specific cockpit for CodeOcean exercises depicted in [Figure 5.8a](#). It includes selected information to provide teachers with a quick overview of the current situation in class and is tailored for one specific exercise. Besides posted *Request for Comments*, it mainly focuses on the time students spent on the assignment and their current progress based on the result of unit test runs. As most exercises in CodeOcean contain multiple unit tests, a percentage score for each interim solution is calculated. By combining the score with the working time spent by students so far, teachers are enabled to identify those students who spent much time on the exercises but only slowly increase their score. With access to that information during their lessons, teachers can intervene rapidly and help students on an individual basis.

5.5.3 *Comparison of Learners from a School Class to MOOC Participants*

With CodeOcean and the code repository CodeHarbor, teachers can create own programming exercises or reuse an existing assignment from colleagues or [MOOCs](#). Reusing an unaltered exercise from someone else does not only save time but also provides additional learning analytics. Many of the exercises available in CodeOcean so far were used by up to 9,000 learners through [MOOCs](#). For each exercise, an average working time across all users is available allowing high-school teachers to compare the average working time of their students with those of [MOOC](#) participants. Using the result of the comparison, teachers might conclude one of the following: (1) Many learners from the comparison group required a long time to solve the exercise (e. g., because the exercise is difficult or the topic is complex) and thus the teacher's class might also take more time on that specific exercise. (2) Only students educated by the specific teacher require significantly longer and thus might struggle with the topic the exercise is about. In the latter case, teachers might use the insights offered to adapt to this situation. The teachers we interviewed valued the concept of a benchmark with other learners. However, they expressed concerns that the comparison is only valuable to them with background knowledge of the group they compare their own class to. Therefore, our approach



- (a) Each bar of the graph on student progress might consist of several sub-bars. These indicate how long a learner worked on the source code to achieve a score. For example, it took learner B about two minutes to reach at least 50% of the overall score. In the additional four minutes spent so far, the score did not exceed 75%.
- (b) Extended version of the wireframe shown in [Figure 5.8a](#). An optional indicator visualizes the average working time of MOOC participants to reach a full score. Teachers might select the comparison group they want their students to be compared with (bottom row) or disable the comparison.

Figure 5.8: Wireframe of the analytical dashboard for one specific exercise within CodeOcean. Teachers are provided with an overview of the scores achieved by their students in correlation with the time they required. In addition, teachers see the most recent *Request for Comments* and have access to submissions of their students.

is to provide a few, predefined user groups as a benchmark and offer these integrated into the existing dashboard as optional extension (as shown in [Figure 5.8b](#)). The comparison groups might include other users accessing the exercises through the HPI Schul-Cloud (addressing the concerns mentioned by teachers) or all users from MOOCs for a larger comparison group.

IMPLEMENTATION

This chapter describes the implementation of the concept previously outlined in [Chapter 5](#) and especially highlights steps required for the evaluation (cf. [Chapter 7](#)). Technically, four main components are involved to provide the functionality of our concept with embedded programming exercises: The [HPI Schul-Cloud](#) platform, a content editor called [edtr.io](#)¹, the programming execution environment [CodeOcean](#), and a [Learning Record Store \(LRS\)](#). All components and their respective communication are shown in [Figure 6.2](#) and described in more detail in [Section 6.1](#). The worksheet editor ([Section 2.3](#)) is the central component delivering and framing the content for students. It consists of multiple plugins providing interactive and multimedia resources. One of the content types provided are practical programming exercises through [CodeOcean](#). [Section 6.3](#) summarizes the changes we made to [CodeOcean](#) to support teachers and enable them to create exercises for their students themselves. Besides other changes, we added support for study groups in [CodeOcean](#) (to provide teachers with learning analytics of their class) and provided teachers with control over the features available within a worksheet. The restrictions defined by the teacher are applied every time a student launches a programming exercise (see [Section 6.4](#)). The results and scores gained by students are transmitted to the [HPI Schul-Cloud](#) (where they are processed by a [LRS](#)) and stored in [CodeOcean](#) for learning analytics ([Section 6.5](#)). Intended for use during lessons, the live dashboard, which is described in [Section 6.6](#), provides teachers with a per exercise view of their students' working times and posted questions. Finally, [Section 6.7](#) summarizes technical learnings from the prototypes and limitations introduced by security considerations in modern web browsers.

6.1 ARCHITECTURE OF WORKSHEETS WITH PRACTICAL PROGRAMMING EXERCISES

The interactive worksheets (see [Figure 6.1](#) for an example) are integrated into the [HPI Schul-Cloud](#), which is a stand-alone service designed to replace other [LMS](#) targeting schools. In our prototype, the [HPI Schul-Cloud](#) acts as the identity provider, manages classes consisting of students and teachers and grants access to course con-

¹ <https://edtr.io>

Vererbung

Dieses Arbeitsblatt besteht aus 7 Teilen, darunter 3 Programmieraufgaben

Motivation:

Ein Zoo hält viele verschiedene Tiere unterschiedlichster Arten. Wenn wir den Zoo mit einer Menge von Klassen und Objekten modellieren möchten, so könnte man für jede Tierart eine eigene Klasse erstellen und pro Tier ein Objekt der jeweiligen Klasse instanzieren. Schauen wir uns dazu ein Beispiel mit Vögeln an ...

```

classDiagram
    class Bird {
        name :String
        order :String
        weight :float
        eat ()
        fly ()
    }
    class Woodpecker
    class Tit
    class Parrot
    class Penguin
    Bird <|-- Woodpecker
    Bird <|-- Tit
    Bird <|-- Parrot
    Bird <|-- Penguin
        
```

Video 3.1:

Quiz 3.1:

Wofür wird Vererbung typischerweise verwendet?

- Zur Erstellung von Beziehungen zwischen verschiedenen Klassen.
- Zur Verringerung von Dopplungen von Codezeilen
- Zur Aufteilung der Arbeit am Code auf verschiedene Entwickler.
- Zum Ausdrücken einer „is-a / ist-ein“-Beziehung.
- Zur Darstellung von verschiedenen Generationen bei Lebewesen, also einer Eltern-Kind-Beziehung

Programmieraufgabe 3.1.2:

➤ OOP2017 Woche3 Kapitel1 Aufgabe2 0%
(Anzeigen)

Ausführen Bewerten Kommentare erbitten

```

1- class Story {
2-     public static void main(String[] args) {
3-         DetectiveRobot ronja = new DetectiveRobot();
4-         ronja.speak();
5-     }
6- }
7-
8-
        
```

Figure 6.1: Exemplary worksheet on the Java topic *inheritance* as used in our study, consisting of text, an image, a video, a multiple choice quiz, and a programming exercise. An enlarged version is available as [Figure B.1](#) and [B.2](#) in [Section B.1](#).

tent, such as digital worksheets. The [HPI Schul-Cloud](#) is built upon a microservice architecture with a dedicated microservice for the worksheet editor called [edtr.io](#). In contrast to other services within the [HPI Schul-Cloud](#), it is directly user-facing and, with minor adjustments, can be used autonomously. More precisely, the editor prototype we used during our evaluation mirrors selected user information. By opening a worksheet, the identity of each user together with the role (student / teacher) is passed to the worksheet editor instance to customize the view being rendered for this user. It is also required to save a user's progress to continue working later and to store submissions, which might be collected and inspected by a teacher.

One of the unique features of the worksheet editor is the ability to embed practical programming exercises for computer science education. The web-based programming environment used is the [CodeOcean](#) platform designed for [MOOCs](#) offered by the [HPI](#). Technically, [CodeOcean](#) is also a stand-alone application containing all tools required to run and store editable source code of user-defined

programs. However, the platform requires an external login for all learners through the **LTI** standard², which is also used to link to a specific exercise. In the context of **MOOCs**, the **LTI** standard is also used to transmit the final result achieved by students back to **MOOC** providers. While the same mechanism could be used in the **HPI Schul-Cloud**, it is limited to returning a single score. Thus, the **HPI Schul-Cloud** prefers using the Experience **API (xAPI)** standard³ to transmit optional learning analytical data. As described by Kleinknecht, the **HPI Schul-Cloud** developers evaluated the integration of an open-source **LRS** to store and process the learning data from multiple sources [32]. The evaluated **LRS** is the open-source solution Learning Locker⁴, a stand-alone tool that also integrates analytical tools for the data received. The resulting architecture of worksheets with all components outlined in this section is shown in **Figure 6.2** together with associations between the different systems.

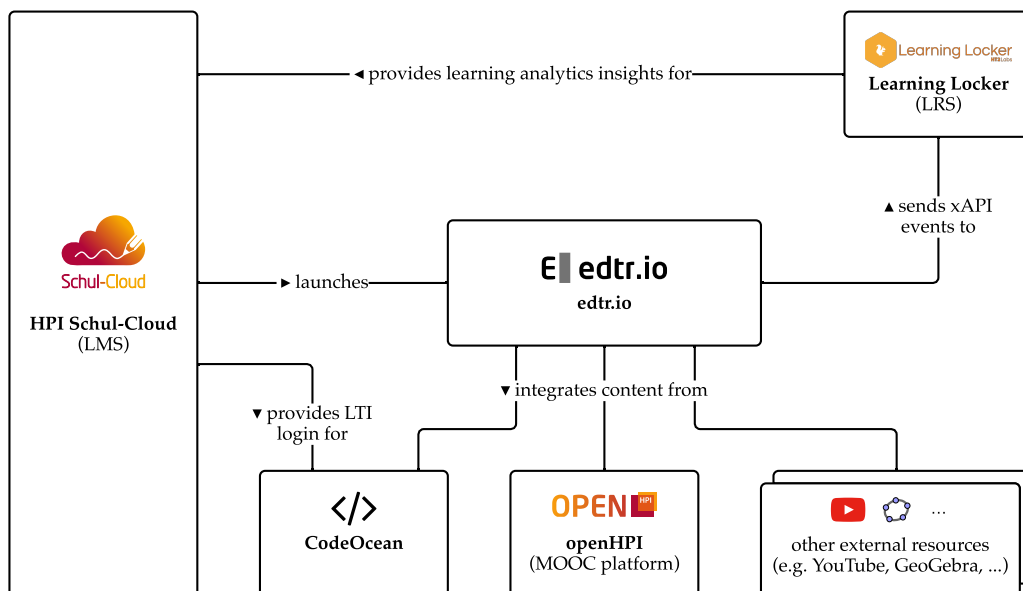


Figure 6.2: System architecture of interactive worksheets with the integration of multimedia content and programming exercises from CodeOcean. An enlarged version is available as **Figure B.3** in **Section B.2**.

² <http://www.imsglobal.org/activity/learning-tools-interoperability>

³ <https://xapi.com/>

⁴ <https://learninglocker.net>

6.2 WORKSHEET EDITOR: EDTR.IO

In his thesis, Wirtz introduced the concept of a modular worksheet editor for use in schools [69]. The concept he described was implemented with *React Components*⁵ to extend the editor with own plugins. *React*⁶ is a client-side JavaScript framework for the creation of interactive user interfaces and is mainly backed by *Facebook*⁷. In a later version, Wirtz extended the concept of a *React*-based editor and rebuilt the modular concept on top of the *Slate*⁸ framework, which is a *React*-based framework to support the development of rich text editors. As the worksheet editor is still under active development, the version we used during our research slightly differs from the current version built in cooperation with *Serlo*⁹, an OER provider based in Munich, Germany. Collaboratively, a WYSIWYG editor called *edtr.io* is developed, which is based on the *React* and *Slate* frameworks. In the implementation designed for usage in the HPI Schul-Cloud, the editor consists of a client-component developed in *React* and a backend server, both depicted with their respective components in Figure 6.3.

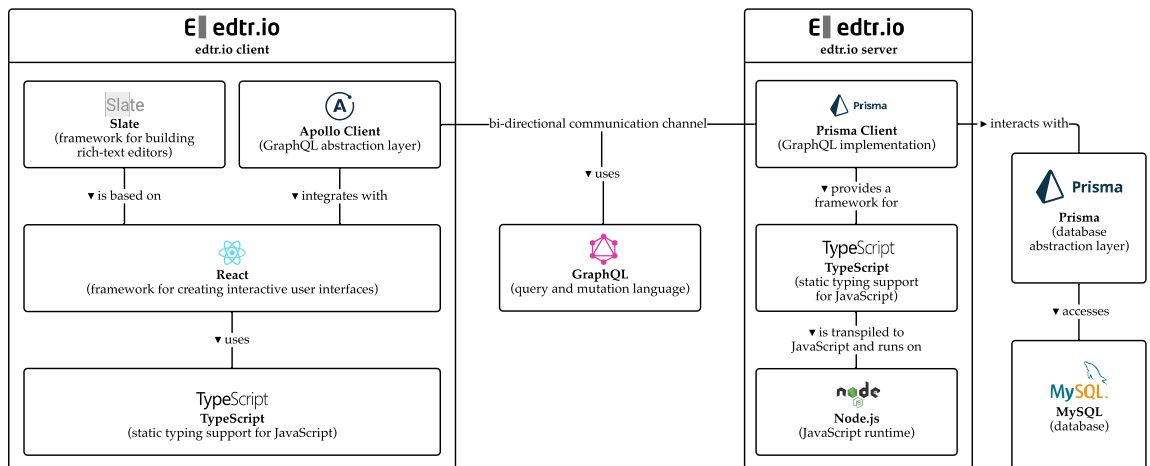


Figure 6.3: System architecture of the worksheet editor *edtr.io* consisting of a backend server and a *React* web application. An enlarged version is available as Figure B.4 in Section B.2.

5 <https://reactjs.org/docs/components-and-props.html>

6 <https://reactjs.org>

7 <https://opensource.facebook.com>

8 <https://www.slatejs.org>

9 <https://serlo.org/>

edtr.io persists all user progress together with documents in a relational *MySQL*¹⁰ database accessed through a *GraphQL*¹¹ backend. *GraphQL* is a query language created by Facebook dedicated to handle resources from within a graph with references to other objects. The relational database is managed by *Prisma*¹², a tool which defines the database schema and migrates it if required. In addition, it provides a framework to create a custom *GraphQL* server. The edtr.io backend server is written in *TypeScript*¹³ and uses the *GraphQL* server framework. *TypeScript* extends JavaScript with strict typing and is transpiled to JavaScript in a build step prior to the deployment. The resulting JavaScript code is interpreted by a *Node.js*¹⁴ server, an open-source runtime for server-side JavaScript.

Using *Prisma* as a backend server provides an additional advantage for building the client-side JavaScript code that is delivered to the web browser. The *Prisma* client provides *TypeScript* compatible types (such as classes for the resources defined in the schema) and, in cooperation with *Apollo*¹⁵, further abstracts access to the backend server. Similar to the build step required for the server, the *React* code designed for the client needs transpilation, which results in a static web page including all editor components. Arndt elaborates on the design decisions that heavily influenced the technology stack [2].

A key design goal of edtr.io was to securely restrict information that is visible to the user and to offer dynamic live updates of worksheets. We decided to address these requirements with a *GraphQL* server and a low-latency bi-directional communication between the worksheet editor and the server. We implemented live updates using the *WebSocket* protocol. It is a standard allowing a browser to keep a communication channel to a backend server and thus provides “a mechanism for browser-based applications that need two-way communication with servers” [17]. The integration of *WebSockets* is especially useful to provide teachers with control over the content available to students. It allows teachers to hide or unhide specific information throughout the lesson. The change will instantly appear on all connected devices — an advantage teachers never had before. In addition, the technology stack eliminates the need for a user to explicitly save progress, thus minimizing the danger of accidental data loss.

Another design goal of edtr.io was to provide an open plugin architecture for the integration of additional content components [69].

10 *MySQL* is an open-source database maintained by Oracle et al. — <https://www.mysql.com>

11 <https://graphql.org>

12 <https://www.prisma.io>

13 <https://www.typescriptlang.org/>

14 <https://nodejs.org/>

15 The *Apollo Client* eases access from *React* to a *GraphQL* server — <https://www.apollographql.com>

Each of those should focus on one specific content or interaction type. In the prototypical implementation, a set of pre-defined plugins is included, allowing teachers to embed a wide range of different content. The worksheet editor features a rich text editor component, an upload for images and supports the integration of videos from YouTube and Vimeo.

One of the features missing in the worksheet editor was an interactive multiple-choice quiz. However, quizzes are one of the most used components in MOOCs and are also present in openHPI courses. The following sections describe the implementation of the multiple-choice plugin together with the integration of learning videos from openHPI and CodeOcean programming exercises. Some plugins, such as the video player, are user-agnostic and only differ in their appearance based on the role of a user, for example, to allow editing textual content. Others, such as a multiple-choice plugin, require knowledge about the current user to associate answers with the corresponding user. In our concept, it is the responsibility of each plugin to display the content or to process integrated learning analytics, as domain knowledge is helpful to create customized views providing a meaningful abstraction of the data.

6.2.1 First-Party Multiple-Choice Quiz Plugin

We decided to add the multiple-choice component natively as an edtr.io plugin. As shown in [Figure 6.4](#), the plugin enables teachers to create multiple-choice questions with pre-defined answers. The multiple-choice plugin we created for edtr.io seamlessly integrates into the architecture of the underlying *Slate* framework.

The editor framework *Slate* structures documents with nested nodes, so that each document is represented through a tree of nodes. As *Slate* has a modular design in mind, the framework keeps a list of available plugins together with the type of nodes they can handle. Thus, each node in a document is processed by the corresponding plugin, which is responsible for rendering the content and for managing nested nodes. The multiple-choice plugin introduces three new node types for documents created with edtr.io, namely the "multiple-choice", "multiple-choice-question" and "multiple-choice-answer" node types. These three node types are rendered by the corresponding *React* components: `MultipleChoiceNode`, `MultipleChoiceQuestionNode` and `MultipleChoiceAnswerNode`.

The "multiple-choice" node is the root node for each multiple-choice plugin. It contains one nested "multiple-choice-question" node containing the question as the first element and at least one "multiple-choice-answer" node with pre-defined answers that stu-

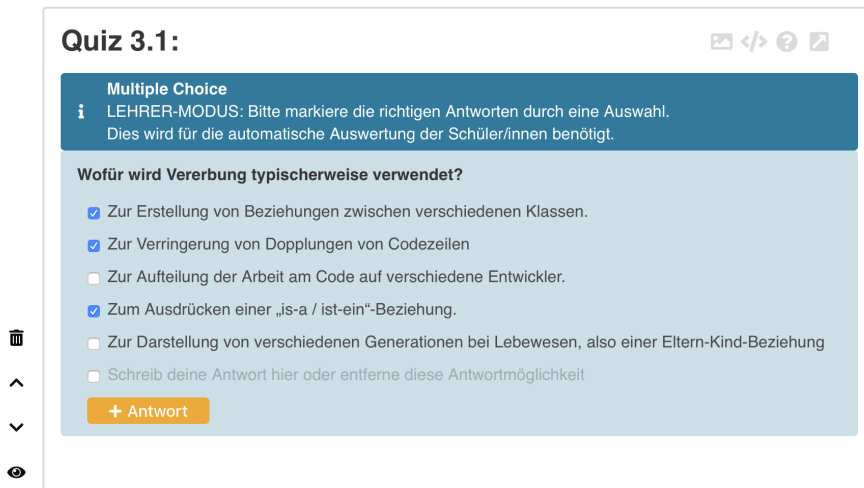


Figure 6.4: The multiple-choice plugin written for edtr.io allows teachers to create own multiple-choice questions together with answers. Each answer is either checked as correct or incorrect to allow an automated scoring of the answers given by students. The internal document representation is listed in [Section B.3](#), [Listing B.1](#).

dents choose from. As shown in [Listing 6.1](#), the document schema formalizes these requirements and further defines how to normalize a document in case it is not in compliance with the defined schema: If the type and order of the nodes is invalid, the missing node is (re-)created. The same mechanism applies if no answer node is present. If, however, the question node is deleted, we decided to remove the whole "multiple-choice" node with all respective child nodes.

As the multiple-choice plugin is natively integrated into edtr.io, it uses the *GraphQL* backend directly to store the state. While questions of the multiple-choice plugin are saved together with the document they are part of, the answer options are persisted separately with own IDs. As a result, each submission of a student refers to an answer option and indicates whether it has been checked by the student or not. Based on this data model (cf. [Listing B.2](#) in [Section B.3](#)), an automated scoring and feedback mechanism for student submissions can be built.

6.2.2 *Embedding Videos from openHPI*

The worksheet editor features an integration of YouTube and Vimeo videos and automatically embeds the respective content with a video player, if an **Uniform Resource Locator (URL)** of one of those platforms is pasted into a document. In order to match the use cases described by the teachers we interviewed, an additional integration of videos from **MOOC** platforms was desired. As the videos integrated on openHPI

```

edtrio-client/src/SlateSchema.ts
1 "multiple-choice": {
2   nodes: [
3     {match: [{type: "multiple-choice-question"}], min: 1, max: 1},
4     {match: [{type: "multiple-choice-answer"}], min: 1},
5   ],
6   normalize: (editor: Editor, {code, node, child, index}: any)=>{
7     switch (code) {
8       case "child_type_invalid": {
9         const type =
10           index === 0
11             ? "multiple-choice-question"
12             : "multiple-choice-answer";
13         return editor.setNodeByKey(child.key, type);
14       }
15       case "child_min_invalid": {
16         if (index === 0) {
17           return editor.removeNodeByKey(node.key);
18         } else {
19           const block = Block.create("multiple-choice-answer");
20           return editor.insertNodeByKey(node.key, index, block);
21         }
22       }
23     }
24     return;
25   },
26 },

```

Listing 6.1: Extract from the *Slate* schema defining the document with the nodes introduced by the multiple-choice plugin. The schema defines the minimum and maximum number for each element and specifies countermeasures if a document got invalid.

are not publicly available on one of those video hosting platforms, a custom integration is required. Furthermore, the integrated video player within the worksheet editor only supports one video stream. Videos on openHPI, however, use two video streams, one containing a recording of the speakers and the other showing slides. Thus, openHPI integrates the *tele-TASK* video player which is designed to synchronize two video streams [40]. The integration of the *tele-TASK* video player as a plugin into edtr.io was previously scheduled¹⁶, but postponed several times. Due to the technical complexity and the fact that our main focus was the integration of programming exercises, we decided for a feasible solution.

As a result, we created a simple web page embedding the *video-player*¹⁷ component and hosted the two video streams ourselves. The latter was necessary as openHPI uses private video hosting on Vimeo which prevents the unauthorized embedding of video content. Within

¹⁶ Ticket: <https://ticketsystem.schul-cloud.org/browse/EDTR-16>

¹⁷ <https://github.com/openHPI/video-player>

the openHPI platform, access to the videos is authorized through a dynamic *GET* parameter that is appended to the video URL on every page load. As the dynamic creation of the video URL was unavailable for our static web page, we served a copy of the videos after consultation with the openHPI team. The web page containing the *video-player* is integrated into edtr.io through the *embed* plugin — a simple component that adds an *iFrame*¹⁸ to the worksheet editor with a customizable location.

6.2.3 Integrating Programming Exercises through an *iFrame* with LTI

Besides integrating a multiple-choice plugin and videos from the openHPI MOOC platform, we aimed to embed practical programming exercises through an integration with the stand-alone web application CodeOcean. The multiple-choice plugin is an example of a plugin completely integrated into edtr.io using the same *GraphQL* backend to store content and user data. For external tools or more complicated use cases, building edtr.io plugins is not feasible. Many third-party web apps designed for educational use cases offer an integration via the LTI standard. LTI allows a learning tool to be linked from an embedding site with some parameters, for example, to specify an exercise and hand over selected user information. The HPI Schul-Cloud implements the LTI standard as a so-called *tool consumer* to allow starting learning apps. Furthermore, CodeOcean provides an LTI interface as a *tool provider* to offer direct access to programming exercises. Based on the LTI implementation in the HPI Schul-Cloud, we extended edtr.io to provide basic functionality as a *tool consumer*. LTI uses standard web technologies so that tools can be accessed through a link or directly embedded via an *iFrame*.

For usage in edtr.io, we decided to embed programming exercises via an *iFrame* allowing us to reuse code within CodeOcean. The required LTI handshake is configurable and either done through the worksheet editor itself or through the HPI Schul-Cloud. In case the existing mechanism within the HPI Schul-Cloud is used, the *iFrame* embedded in a worksheet requests a dedicated web page in the HPI Schul-Cloud. The page loaded contains an *Hypertext Markup Language (HTML)* form with hidden *input* fields and a JavaScript snippet to submit the form automatically. The form contains the required LTI parameters and is submitted as a *POST* request to the LTI endpoint of CodeOcean. The *tool consumer* processes the parameters and redirects the user to the corresponding exercise within the *iFrame*. Section 6.4 describes the LTI handshake in more detail.

¹⁸ An *iFrame* allows a web page to embed another web page in the same view.

The existing mechanism offered by the [HPI Schul-Cloud](#) to start a CodeOcean exercise is initiated by accessing a pre-configured [URL](#) through a *GET* request. Therefore, this [URL](#) can be used in conjunction with the *embed* plugin available within [edtr.io](#). Furthermore, we designed the [LTI](#) handshake in [edtr.io](#) to be compatible with the existing *embed* plugin and discarded managing a customized CodeOcean plugin for the worksheet editor. A dedicated plugin would allow teachers to select an exercise without manually copying the exercise [ID](#) (as described in [Section 5.2.2](#)). In a spike, we were able to send information (e. g., about a selected exercise) from an *iFrame* showing CodeOcean to the parent editor framework with a custom CodeOcean plugin. However, we decided not to use our custom solution in favor of the upcoming version 1.3 of the [LTI](#) standard. It extends the [LTI](#) standard to provide an interface for the automated configuration of [LTI](#) parameters including the selection of learning material that should be accessed (in our case programming exercises). [LTI v1.3](#) is currently in the “Candidate Final status”¹⁹ with a last update on January 14th, 2019. Therefore, it is not yet implemented in CodeOcean, the [HPI Schul-Cloud](#) or [edtr.io](#) but its implementation is intended in the near future.

6.3 INTRODUCTION OF THE TEACHER ROLE AND STUDY GROUPS IN CODEOCEAN

CodeOcean²⁰ is a monolithic web application written in Ruby²¹ using the common Model-View-Controller ([MVC](#)) framework *Ruby on Rails*²² together with many third-party Ruby packages, so-called *gems*. CodeOcean connects to a *PostgreSQL*²³ database to persist information and uses *Docker* containers to execute source code. While [edtr.io](#) is a [Single-Page Application \(SPA\)](#) and uses *GraphQL*, CodeOcean is a more traditional multi-page web application based on [Representational State Transfer \(REST\)](#).

As described in [Section 5.2](#), CodeOcean was built for programming [MOOCs](#) offering different access levels for [MOOC](#) instructors and learners. Therefore, we extended CodeOcean to support teachers and thus included features targeting the use of CodeOcean with classes ([Section 6.3.1](#)). As a pre-requirement for other functionality, the code execution platform needs to be aware of organizational structures, such as students and their membership in classes (see [Section 6.3.2](#)), and thus introduces so-called study groups. Providing

¹⁹ <http://www.imsglobal.org/activity/learning-tools-interoperability>

²⁰ <https://github.com/openHPI/codeocean>

²¹ <https://www.ruby-lang.org>

²² <https://rubyonrails.org>

²³ An open-source relational database — <https://www.postgresql.org>

teachers with additional privileges in comparison to students and reproducing organizational structures with study groups is done automatically in the background through the [LTI](#) handshake and the integration with the [HPI Schul-Cloud](#).

6.3.1 Features Available for Teachers

Previously, two user roles were used in CodeOcean: The role of an administrator having access to all features and the role of a learner using the platform to write and execute source code. In order to manage user roles and access rights, CodeOcean uses the Ruby gem *Pundit*²⁴. The gem follows an object-oriented design and introduces so-called policies to validate access to a given object or, if a concrete object is unavailable, to a class. Thus, we were able to reuse the integration of *Pundit* and extend it with a role designed for high-school teachers. For security reasons, the default policies in CodeOcean limit access to administrators if not stated otherwise (explicit opt-in for non-administrative users). We reviewed every single action as well as the visibility of information in CodeOcean and selected a subset of features for teachers. All features require user login and are made exclusively available for teachers besides platform administrators. In particular, we allowed teachers to browse public exercises and view more details, such as editor settings applied for students or internal tags used to describe the exercise. Furthermore, we outlined support for teachers to access the test cases usually hidden for learners and to clone publicly available exercises so that teachers might adapt them to their needs. As a result of our work, teachers can now create, edit and delete their own practical programming exercises.

Every exercise available in CodeOcean is linked to a so-called execution environment which specifies settings and requirements to execute source code. While only CodeOcean administrators should be capable of defining these environments, we allowed teachers to inspect basic details of execution environments, such as the memory limit for each container or whether internet access is available during code execution. Besides other features, e. g., access to the list of posted *Request for Comments*, teachers get access to a list of study groups and can manage memberships of those they are a member of (cf. [Section 6.3.2](#)). Many of the existing *Pundit* policies in CodeOcean were updated to reflect our new permissions. An example for a concrete policy is shown in [Listing 6.2](#), which defines different permissions for exercise details and the study group dashboard introduced in [Section 6.6](#).

The User Interface (UI) in CodeOcean including all navigation items has been touched to consider the permissions of the currently logged-

²⁴ <https://github.com/varvet/pundit>


```

_____ app/policies/exercise_policy.rb _____
1 def show?
2   admin? || teacher?
3 end
4
5 def study_group_dashboard?
6   admin? || teacher_in_study_group?
7 end
_____

```

Listing 6.2: Two concrete *Pundit* policies regarding exercises. They allow teachers and administrators to show details of an exercises and restrict the visibility of the study group dashboard to teachers of the same study group (using the code shown in Listing 6.3) besides platform administrators.

in user: All links to resources the user is not allowed to access are hidden. We integrated policy checks during the server-side rendering process of web pages. During page loads, the specific user object with the associated permissions is utilized to pre-validate whether access to a linked page would be granted. Only if that is the case, a link to the target is placed. In a few occurrences, e. g., whenever a file type is referenced in an exercise, we show the text (i. e., the file type) but refrain from linking to the detail page if that is unavailable to the user. Without manually editing the [URL](#), it is thus unlikely for users to get an error message about insufficient access rights.

6.3.2 Automated Creation of Study Groups

By granting teachers extended access to CodeOcean (as described in the previous [Section 6.3.1](#)), some of the requirements expressed by teachers are met: They are now enabled to use the same tools as [MOOC](#) instructors to create practical programming exercises using CodeOcean. However, teachers also want access to the submissions of their students (cf. [Section 5.4](#)) on demand. Thus, CodeOcean needs to reflect organizational structures. Therefore, we implemented study groups that are automatically created and extended through information received by the tool launch via [LTI](#) (see [Section 6.4](#)). Class structures and other groups are represented through a `StudyGroup` object in CodeOcean. As shown in [Figure 6.5](#), the classes `User` and `StudyGroup` are connected through a many-to-many relationship²⁵.

Teachers are restricted to view details about student submissions done in the context of their study group. Therefore, each submission of a student is tagged with the study group as a context (cf. [Figure 6.5](#)). We implemented the explicit tagging of submissions for two reasons:

²⁵ Each user might be a member of many different study groups and each study group might contain several users.

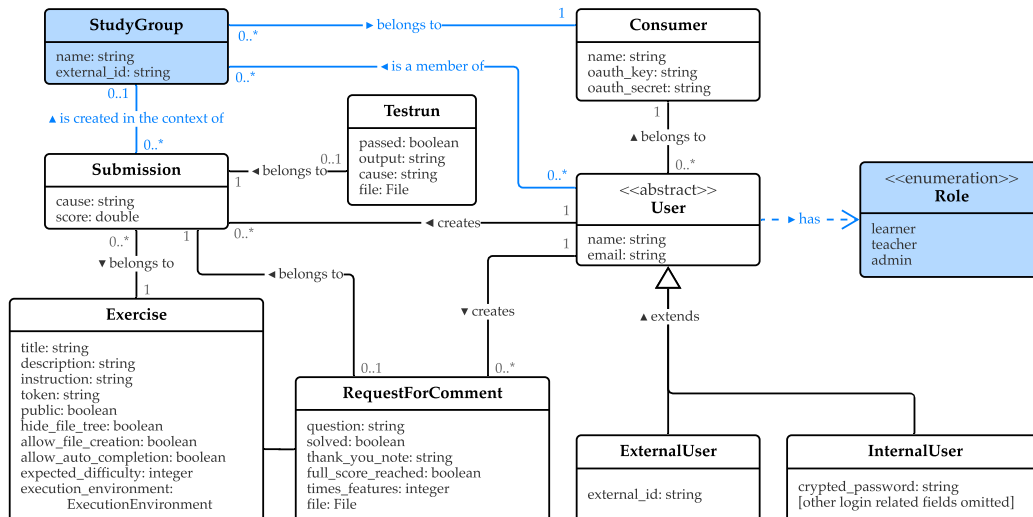


Figure 6.5: UML diagram of the `StudyGroup` and related classes with changes highlighted in blue. The diagram includes those classes which are relevant in the context of a `StudyGroup`. We further extended the existing role model to allow an `ExternalUser` to be assigned with a role (such as the teacher role required for access to learning analytics of a `StudyGroup`).

(1) Teachers can reuse exercises from MOOCs so that a submission to a particular exercise might either be created by a student of the teacher’s class or by any other learner accessing the exercise through a MOOC. (2) Students could access an exercise as part of a programming lesson in school or in their free time. In both cases, teachers should not see other students’ solution and submissions of their students created in another context without explicit consent. A *Pundit* policy (as listed in Listing 6.3) validates whether a teacher is authorized to access a specific submission based on the study group associated with the solution submitted by a student. By persisting the context of a submission through a reference to the study group and by using our custom *Pundit*, we efficiently prevent unauthorized access in compliance with the GDPR. Furthermore, we use study groups to offer teachers learning analytics of their students, e.g., through a live dashboard for use during lessons (see Section 6.6).

6.4 LAUNCHING PROGRAMMING EXERCISES WITH DIFFERENT CONFIGURATIONS

As introduced in Section 6.2.3, CodeOcean exercises are integrated through the Learning Tools Interoperability (LTI) standard. The specification harmonizes links between educational platforms and their resources. LTI is built upon multiple optional so-called *services* that

```

_____ app/policies/application_policy.rb _____
1 def everyone_in_study_group
2   study_group = @record.study_group
3   return false if study_group.blank?
4
5   users_in_same_study_group = study_group.users
6   return false if users_in_same_study_group.blank?
7
8   users_in_same_study_group.include? @user
9 end
10 private :everyone_in_study_group
11
12 def teacher_in_study_group?
13   teacher? && everyone_in_study_group
14 end
15 private :teacher_in_study_group?
_____

```

Listing 6.3: *Pundit* policies that either allow access for every member of a study group or limit access to a teacher in the given study group. These policies are used throughout CodeOcean to restrict the visibility of information.

extend the *Basic Launch* feature. One of those services, the *Basic Outcomes* service, allows a score achieved by the learner to be transmitted back to the LMS which initiated the LTI handshake. Besides the official services defined in the LTI standard, additional attributes are allowed if appropriately prefixed with `custom_` to support own use cases. For these, only the *tool provider* and *tool consumer* need to agree on a mutual understanding of these application-dependent attributes. CodeOcean uses custom parameters to realize feature restrictions and enforce restrictions applied by the teacher. In total, we added support for eleven different restrictions allowing teachers to customize the integration of CodeOcean exercises (see Table 6.1 and Section 6.4.2).

In the context of school education in Germany, privacy and legal considerations have to be a major part of the overall product design. Therefore, all external plugins within a digital worksheet only get limited access to user data if at all. CodeOcean needs to identify returning learners to enable continuing previous work. For this reason, the HPI Schul-Cloud is configured to create a pseudonymous ID for each user and service, which is the only person-related information CodeOcean requires. Additionally, to represent the student-teacher relationship on the programming platform, each tool launch via LTI also includes a context ID and the role of each user (student/teacher)²⁶. The context ID, specified as `resource_link_id`, must be a unique identifier of a school class and grants teachers access to analytics about their students exclusively. Based on the unique course ID,

²⁶ With LTI, the student's role is named Learner and the teacher's role is called Instructor.

CodeOcean builds the study groups introduced in [Section 6.3.2](#) with all learners of the given class being members.

6.4.1 Deep Linking with the LTI Standard

Without additional [LTI](#) services (such as the *Basic Outcomes* service to return results from the *tool provider* to the *tool consumer*) an [LTI Basic Launch](#) is a single uni-directional message. In fact, it is a plain [Hypertext Transfer Protocol \(HTTP\) POST](#) request prepared by the *tool consumer* and sent from the users' web browser to the *tool provider*. As shown in [Listing 6.4](#), the request is signed and verified through the [OAuth 1.0 standard](#)²⁷ to prevent any tampering of the data.

```

1 POST /lti/launch HTTP/1.1
2 Host: codeocean.openhpi.de
3 Content-Length: 561
4 Origin: https://schul-cloud.org
5 Content-Type: application/x-www-form-urlencoded
6 oauth_consumer_key: 19c047f73e313fe1140706ea5f24494d
7 oauth_signature_method: HMAC-SHA1
8 oauth_timestamp: 1556113871
9 oauth_nonce: bWLcGxCuy5yPgYuKJ9Qi7gl1Igyka00qwLL0988y2Nw
10 oauth_version: 1.0
11 oauth_signature: uwzDVnvYaIIqzli6T0GsrxJ3eeY=
12 lti_version: LTI-1p0
13 lti_message_type: basic-lti-launch-request
14 resource_link_id: 5bc48a31db4df00011083c83 ; Course context
15 user_id: 4a98735b-e3e3-44fd-9c0a-4d10b4946253 ; Pseudonymous ID
16 lis_person_name_full: '' ; No name passed
17 lis_person_contact_email_primary: '' ; No email passed
18 roles: Learner ; Student role
19 custom_token: 4d324ad6 ; exercise ID
20 custom_locale: de ; language
21 custom_embed_options_read_only: true ; prevents edits

```

Listing 6.4: Exemplary [HTTP](#) request initiated by the [HPI Schul-Cloud](#) launching a CodeOcean exercise via [LTI](#). The course context is transmitted together with a pseudonymous [ID](#) and the user's role (student/teacher). The exercise is identified through the `custom_token`. In this example, CodeOcean will be displayed in German and disallow students to edit the source code. For the sake of readability, the listing refrains from showing the [URL](#)-encoded parameters and some optional [HTTP](#) headers, such as the browser's user agent.

²⁷ [LTI v1.3](#) is the first version using [OAuth 2.0](#) [55] instead of [OAuth 1.0](#) [53]

In CodeOcean, an application-defined parameter is used to identify a concrete exercise through [LTI](#). The so-called `custom_token` is thus required for every [LTI](#) launch targeting CodeOcean. Using the token, CodeOcean enables deep linking to a dedicated practical programming assignment. Otherwise, as the [URL](#) for [LTI](#) is always <https://codeocean.openhpi.de/lti/launch>, CodeOcean could not be aware of a concrete exercise at all and would display an error indicating the missing `custom_token`.

6.4.2 *Introducing Feature Restrictions through LTI*

In order to realize our concept of adaptive restrictions (cf. [Section 5.3.2](#)), we extended CodeOcean to support additional custom parameters. [Table 6.1](#) lists all application-defined parameters introduced by us and their corresponding impact on CodeOcean. Each restriction enabled through [LTI](#) is stored in the encrypted session cookie of a user. As the cookie is encrypted by the Rails backend server, it cannot be read or modified by an end user. As long as a learner is signed in (which is required for CodeOcean to function properly), the limitations are in place. Each possible restriction influences the [UI](#) of CodeOcean by hiding specific information or disabling some actions. Furthermore, we extended the *Pundit* policies to consider information about applied restrictions as stored in the encrypted session cookie, which is sent along with every request.

6.5 TRANSMITTING RESULTS FROM CODEOCEAN BACK TO THE HPI SCHUL-CLOUD

For each exercise with unit tests, CodeOcean is capable to calculate a score for student submissions. This score is shown to learners (if not disabled through restrictions) and, on demand, accessible by teachers within CodeOcean. However, the views in CodeOcean only include results achieved by programming exercises. To provide students with overall progress details including all interactive element types within a course, the [HPI Schul-Cloud](#) needs to be aware of all available scores. CodeOcean as an external system does not provide an [API](#) for the [HPI Schul-Cloud](#) to query learning progress. Instead, the [HPI Schul-Cloud](#) requires tools (such as native plugins within the worksheet editor and external systems as CodeOcean) to submit progress information on occurrence to a dedicated [LRS](#).

Similar to CodeOcean, the [edtr.io](#) backend stores the current user progress and submissions of some native plugins, such as answers given for multiple-choice questions in a dedicated database (cf. [Section 6.2.1](#)). In addition, selected learning data is sent to Learning

CUSTOM PARAMETER	RESULTING IMPACT ON CODEOCEAN
hide_navbar	The navigation bar displayed at the top of CodeOcean is hidden. Usually, it shows the application name CodeOcean together with an option to select the UI language and provides access to <i>Request for Comments</i> posted by the user.
hide_exercise_description	The exercise description when working on a programming exercise is hidden. The teacher might use edtr.io or the HPI Schul-Cloud to introduce to code.
disable_run	Users are disallowed to compile and execute their source code.
disable_score	Independent of the option to execute the source code, this option disables running the test cases. Thus, no score is available for display.
disable_rfc	CodeOcean prevents the user from creating new <i>Request for Comments</i> to seek for help.
disable_interventions	Usually, users get an intervention while working on an exercise in CodeOcean. The upcoming dialogue suggests to ask for help using <i>Request for Comments</i> or to take a short break. By setting this option, no intervention of learners occurs.
hide_sidebar	The left sidebar within CodeOcean providing a file tree to an exercise is hidden. As a result, learners are unable to switch, download or add files.
read_only	All source code files available in this exercise are read-only and cannot be altered.
hide_test_results	Detailed test results from score runs are hidden but the score itself is calculated and shown to the learner.
disable_hints	Automated hints are not provided to learners. These hints usually rephrase an error message that occurred during program execution and are intended to help learners to find their mistakes more quickly.
disable_download	Users are not allowed to download an archive with their current progress of all visible source code files.

Table 6.1: All new application-specific parameters supported by CodeOcean to disable some of the available features. Each option name is prefixed with “custom_embed_options_” when used via [LTI](#).

Locker, an open-source [LRS](#) deployed in the [HPI](#) Schul-Cloud, which accepts and processes learning activity data defined using the [xAPI](#) standard. Each activity is described with a statement consisting of up to four elements. Statements follow a simple “actor — verb — activity” structure and may contain additional attributes. For example, a valid statement is: John (actor) passed (verb) test A (activity) with 65% (additional attributes). Based on [xAPI](#) data, the [LRS](#) creates custom views

and graphs for a suitable visualization. Thus, it is the responsibility of the HPI Schul-Cloud (instead of the plugins or external tools) to manage access to data stored in the LRS. Ideally, each edtr.io plugin is enabled to publish learning analytic data that is either directly handled by edtr.io and also processed directly (e. g., to provide instant feedback) or stored for future reference in the Learning Locker.

In addition to first-party edtr.io plugins, the HPI Schul-Cloud encourages external systems to publish their learning data to the LRS. In case of CodeOcean, two approaches are available: Either (1) the same xAPI standard as used by first-party edtr.io plugins or (2) the LTI Basic Outcomes service as proven by the MOOCs. As neither the HPI Schul-Cloud nor edtr.io implement the LTI Basic Outcomes service, only the xAPI standard is available so far. However, both solutions are slightly different in their concept of submission handling.

6.5.1 Differences between Final Submissions and Intermediate Submissions

The LTI Basic Outcome service limits the result being sent from a *tool provider* to a *tool consumer* to “a decimal numeric grade in the range from 0.0 – 1.0” [63]. When launching an external tool with support for the LTI Basic Outcome service, a few additional parameters are passed, including a pre-generated ID called `lis_result_sourcedid` which the *tool provider* should use to associate a score with a particular user and exercise. The result is directly sent from the *tool provider* to the *tool consumer* using the URL passed as `lis_outcome_service_url` parameter during the tool launch. Moreover, the LTI Basic Outcome service specification states that grades should be treated “though there is only a single grade for each `lis_result_sourcedid`” [63]. While a grade history can be maintained by the *tool consumer*, the specification does not elaborate how previous grades should be interpreted.

In contrast to LTI, the xAPI standard describes finished actions and thus uses verbs in the past tense (e. g., “John *passed* test A with 65%”). The specification clarifies that “statements are immutable (they cannot be changed)” [70]. In general, the xAPI specification is more flexible than the LTI Basic Outcomes service. In conjunction with the LRS used, xAPI allows arbitrary verbs and activities to be defined²⁸. As described by Glandorf, xAPI statements can either be sent directly from the learner’s web browser (with an adequate authorization) or proxied through the HPI Schul-Cloud [19]. Glandorf further elaborated that a proxy-mechanism is required in the context of pseudonymized usage of third-party tools [19].

²⁸ Individual extensions are explicitly allowed; however, it is recommended to use elements published in a central xAPI Registry, such as <https://xapi.com/registry>.

6.5.2 Using the Worksheet Editor to Forward Analytical Data

Enabling CodeOcean to send **xAPI** statements directly to the **LRS** requires a working de-pseudonymization proxy within the **HPI Schul-Cloud**. Furthermore, CodeOcean needs to know **URL** of this service, which could be sent along as an additional custom **LTI** parameter (similar to the configuration of the **LTI Basic Outcome** service). Another approach is to use **edtr.io** to handle **xAPI** statements and forward these to the **LRS**. As **edtr.io** is a first-party tool provided by the **HPI Schul-Cloud**, it does not require any pseudonymization. For the use in worksheets, we decided to integrate programming exercises from CodeOcean with the **edtr.io embed** plugin and an **iFrame** (Section 6.2.3). Thus, CodeOcean within the **iFrame** can pass messages via JavaScript to the embedding **edtr.io**. In a spike, we were able to transmit arbitrary information using the JavaScript method `targetWindow.postMessage(message, targetOrigin, [transfer]);`²⁹ through the user's web browser. The architecture of CodeOcean is already prepared to pass scores via JavaScript messages: On score runs using the unit tests, a **WebSocket** connection between the learner's browser and the CodeOcean server is established. Score results are transmitted asynchronously and are already processed by JavaScript code³⁰ to update the score that is shown to learners.

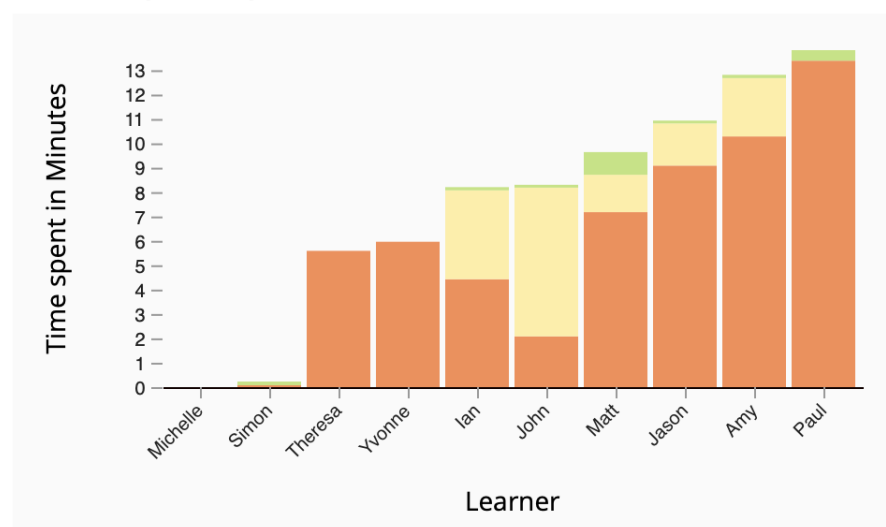
6.6 PER EXERCISE DASHBOARD FOR TEACHERS

Authorized teachers can access the learning analytics through the **LRS** offered by the **HPI Schul-Cloud**. Additionally, they get more detailed information about their students in CodeOcean via a dashboard summarizing activity per exercise. It shows the time spent by the students in correlation with the points achieved for a given exercise. In addition, the dashboard also shows live questions asked by the students while working on the assignment. The teacher can access the dashboard either as a stand-alone page within CodeOcean or as an embedded view within the worksheet. Further, the dashboard will allow a comparison with **MOOC** learners and will show a list of exceptions that occurred. These data are already available; for example, exceptions are extracted from the command line output and displayed separately to help students to identify the mistakes they made.

²⁹ <https://developer.mozilla.org/en-US/docs/Web/API/Window/postMessage>

³⁰ e. g., in method `handleScoringResponse`: `function` (results)

Time spent per Learner



Related Requests for Comments

		Question	Username	Request Date
✓	2	How can I correctly declare a multi-level inheritance hierarchy in Java?	John	5 minutes ago

Figure 6.6: The live dashboard available to teachers during a lesson in CodeOcean. The graph represents the working time with different colors indicating the current progress of students: orange ($< 50\%$), yellow ($50\%–89\%$), green ($\geq 90\%$). In addition, recent *Requests for Comments* posted by students are shown together with the number of comments and whether the question is marked as resolved (green check).

6.6.1 Enabling Live Updates through WebSockets

The analytic dashboard per exercise, which is shown in [Figure 6.6](#), is designed for teachers to be used during lessons. Thus, an important aspect is the ability of the view to reflect progress of students in real time. As elaborated in [Section 6.2](#), the WebSocket protocol was designed to enable a bi-directional communication allowing a server to push new information to the client as soon as they are available [17]. Consequently, WebSockets provide a suitable technology to support our concept of a live dashboard for teachers. Moreover, CodeOcean was already using WebSocket connections successfully with the Ruby

gem `Tubesock`³¹. The gem is used to stream the standard output of a code run in real time to the learner's web browser. In this scenario, a single user gets continuous updates of a directly initiated code run. In CodeOcean, learners submit their code using an `HTTP POST` request and trigger a code run through a second `API` call. The second `HTTP` request is *hijacked* by the `Tubesock` gem, which is responsible for initiating the switch from `HTTP` to a `WebSocket` connection through the following `HTTP` response: `HTTP/1.1 101 Switching Protocols`. The existing use case streaming output to a single learner is well supported by `Tubesock`.

For the live dashboard, however, an additional requirement emerges: The teacher wants information about code runs initiated by other users. Thus, the multi-threaded CodeOcean server is required to use a synchronization mechanism. `Tubesock` focuses on the `WebSocket` connection and does not provide so-called *broadcasting*³² support. Since Ruby on Rails 5.0, the web framework used in CodeOcean has been extended with `ActionCable`³³, a core component adding native support for `WebSockets` and *broadcasting*. In order to use `ActionCable` in CodeOcean, we implemented a major version upgrade of Ruby on Rails from version 4.2 to 5.2 and thereby integrated support for `ActionCable`. Besides handling `WebSocket` connections, `ActionCable` supports *broadcasting* and also includes a client-side JavaScript framework for a seamless integration. `ActionCable` uses so-called *Channels* to differentiate various contexts from each other. For our use case in CodeOcean, a unique channel is created for every combination of a study group and an exercise mapping to an individual *per exercise* dashboard. The *broadcasting* support in `ActionCable` is enabled through a *publish-subscribe* pattern featured by a `Redis`³⁴ or `PostgreSQL` server. We decided to use the `PostgreSQL` integration, even though it is limited to messages with a maximum size of 8 kB, as a `PostgreSQL` server is already deployed and in use for the main database. Switching from `PostgreSQL` to `Redis` is possible at any point in time without modification of the source code.

We enabled the authentication of users through the cookie sent along with the initial request to open a `WebSocket` connection. As CodeOcean is a monolithic app, `ActionCable` has access to the same cookie encryption mechanism used throughout other parts of the system. Furthermore, we integrated `Pundit` policies for the authorization and restricted access to a study group channel for teachers of the study group and administrators. Other authenticated users,

³¹ <https://github.com/ngauthier/tubesock>

³² The established term *broadcasting* is used to describe the forwarding of a specific message to all connected users of a given context. In our case, the *broadcasting* is limited to connected teachers of a study group, which mostly will be a single teacher.

³³ https://guides.rubyonrails.org/action_cable_overview.html

³⁴ `Redis` is an open-source, in-memory key-value store — <https://redis.io>

especially students, might still establish a WebSocket connection, but are unauthorized to subscribe to arbitrary channels, which renders the connection useless for them. So far, we only use ActionCable for live updates within the *per exercise* dashboard. Historical data generated before a user visited the dashboard are not transmitted through the WebSocket connection but are queried on page load from the database and directly included by the server-side rendering engine into the generated HTML. In order to notify connected users of updates, we added custom callbacks to the ActiveRecord³⁵ models. After data is saved in either the `Submission`, `RequestForComment` or `Comment` model, the preparation of data for ActionCable is triggered. We added the `ActionCableHelper` to process changes asynchronously in a new thread and sent the updates to subscribed users of the specific channel.

6.6.2 *Aggregating the Working Times of Students*

One of the features available on the dashboard is a graph visualizing the working time and the score achieved by students. The working time is automatically calculated from the submissions made by students. Each submission is persisted in the database with a timestamp (value of `created_at`). Using the first and last submission to calculate the working time is insufficient, as it would ignore breaks possible made by the learner. Thus, we decided to handle each time difference between two submissions that is longer than five minutes as a break and exclude that time span from the working time calculation. For a fast calculation, we moved the main logic as close as possible to the data and hence created a custom query (as shown in Listing B.3, Section B.4) for our database using Structured Query Language (SQL).

Our query is customizable via Rails and filters for a given exercise and study group. Additionally, we only need the working time of the learner with the newest submission for each live update using ActionCable, as all other working times will be unaffected by that change. Thus, an additional filter criteria is specified unless a full page load is processed. Regardless of the filters used, we do not only want to show the total working time spent by a student, but also which time a learner required to achieve a given score. However, CodeOcean differentiates between submission types and only includes a score of test runs. Therefore, we implicitly extended all other submission types with the score of the last test run performed by the learner. Using multiple window functions³⁶, the time a learner required to achieve a change in the score is calculated together with the total time

³⁵ https://guides.rubyonrails.org/active_record_basics.html

³⁶ “Window functions provide the ability to perform calculations across sets of rows that are related to the current query row.” — <https://www.postgresql.org/docs/11/functions-window.html>

spent on the exercise. In a next step, learners are sorted based on their working time from the shortest total time spent to the highest and indexed with a temporary **ID** per learner. Finally, the result is returned to Rails together with the name of each user (as exemplary shown in [Table B.1](#)). The result, regardless of whether it is used for a full page load or a live update, is converted to JavaScript Object Notation (**JSON**).

The **JSON** is passed to the teacher's web browser and used to draw a graph with the JavaScript visualization library `D3.js`³⁷. Initially, the **JSON** is transmitted as a data attribute³⁸ of an **HTML** element included in the page. The full **JSON** contains data for all users of a specific study group and is used to initialize the graph and the axes. For each learner and score, a sub-bar is created indicating the time spent by the student on an exercise through the height. Additionally, the percentage of the possible score achieved by the learner is mapped through a color to the sub-bar, ranging from 0% (red) to 100% (green). We decided to stack the sub-bars on top of each other for each learner, so that a multi-colored bar is shown per learner. Updates to the bar are directly handled by JavaScript code and modify the existing graph in place. If data for a new student is added or the teacher wishes to re-sort the graph based on the total working times, a full page refresh is triggered.

6.6.3 Request for Comments within Study Groups

In addition to the working time graph, the study group dashboard also features a list of *Request for Comments* that students asked while working on the given exercise. On the initial page load, the list of *Request for Comments* is rendered on the CodeOcean server and transmitted as final **HTML** to the web browser. Thus, it does not involve specific JavaScript code for the initial rendering as it is the case with the working time graph. [Listing 6.5](#) shows how the table body is initially rendered using the *Slim*³⁹ template engine. The variable `@request_for_comments` includes a collection of all *Request for Comments*. The render method automatically invokes a template, called `partial`, for each entry in the collection and concatenates the resulting **HTML**.

Whenever a change to a *Request for Comment* occurs (e. g., because a new one is created or an existing one got a new comment), the `ActionCableHelper` is triggered through an `:after_save` hook within the corresponding model. The method called invokes the server-side

37 <https://d3js.org>

38 https://developer.mozilla.org/en-US/docs/Learn/HTML/Howto/Use_data_attributes

39 <http://slim-lang.com>

```

_____ app/views/exercises/study_group_dashboard.html.slim _____
1 tbody#posted_rfcs
2 = render(partial: 'request_for_comments/list_entry',
   ↪ collection: @request_for_comments,
   ↪ as: :request_for_comment)
_____

```

Listing 6.5: Extract of the *Slim* template used to render the body of the table listing *Request for Comments*. The actual template code for each table row is separated in another partial template and called once per *Request for Comment*.

rendering of a table row (as shown in Listing 6.5) and broadcasts the resulting *HTML* to the client using WebSockets. In this case, the same template previously used by the full page load is reused with a single instance of a *RequestForComment* (cf. Listing 6.6). Based on the new data received, the web browser checks whether the row represents a new *Request for Comment* or is meant to update an existing row in the table. Therefore, each table row within the *HTML* stores the *Request for Comment ID* as a data attribute. In case of an update, the *HTML* code of the existing row is replaced with the new *HTML* received through the WebSocket channel. Thus, it keeps the position within the table. New entries are added to the top, so that the table is always ordered by the creation timestamp of *Request for Comments*.

```

_____ app/helpers/action_cable_helper.rb _____
1 ApplicationController.render(
   ↪ partial: 'request_for_comments/list_entry',
   ↪ locals: {request_for_comment: self})
_____

```

Listing 6.6: Method call within the *ActionCableHelper* responsible for sending updates to the live dashboard using *ActionCable*. The code is run in the context of the *RequestForComment* model, so that *self* refers to the *Request for Comment* that was created or updated. Rendering the single table row is done by reusing the existing *Slim* template that is also used in Listing 6.5.

6.7 LEARNINGS FROM THE IMPLEMENTATION

Our prototype described in the previous sections was used throughout the implementation phase of our concept and the evaluation by different teachers and their students in real lessons. This section summarizes some learnings and their impact due to different web browsers, network configurations and limits we identified. If possible,

only standardized [HTTP](#) ports (such as 80 and 443) should be used for an educational tool as some schools block many uncommon ports.

6.7.1 *Saving Session Information in a Cookie*

While CodeOcean is a stand-alone web application, it can only be used by authenticated users. Thus, teachers and students must access the tool through a tool launch done via [LTI](#). As a result of the [LTI](#) handshake (as described in [Section 6.4](#)), a cookie is sent to the client and used for every subsequent request. According to the specification, [HTTP](#) cookies are used to let “the servers maintain a stateful session over the mostly stateless [HTTP](#) protocol” [54]. Session handling via cookies only works properly, if the web browser accepts the cookie and uses it according to the specification. As the cookie is uploaded as part of every request sent to the server, a smaller cookie reduces the overall time required to complete an [HTTP](#) request. In accordance with the specification, Rails limits the size of a cookie to 4 kB [54]. As a result of the size limitation and due to other values stored in the cookie, we encountered limitations for storing restrictions applied by the teacher. In our implementation, these are stored in the session cookie. Due to size limitations, the restrictions are not assigned to a specific exercise. Therefore, a single worksheet currently does not support two exercises with a different set of restrictions. A possible solution would be to switch to another Session store as offered and supported by the Ruby on Rails framework⁴⁰. By using another session store, the cookie would only include an [ID](#) as a reference to a server-provided store circumventing the size limitation.

6.7.2 *Worksheets with Cross-Origin Frames*

Other issues with cookies arise, if CodeOcean is embedded into another website, i. e., through an *iFrame*. Modern web browsers introduce restrictions for embedded elements that are served from another domain than the main document. These so-called cross-origin elements might be subject to restrictions applied by the web browser due to security and privacy concerns of users. Some browsers, such as Apple Safari⁴¹ do not accept cookies from cross-origin sites by default ⁴². Thus, if the worksheet editor is hosted at [schul-cloud.org](#) and CodeOcean is served as part of an *iFrame* through [codeocean.openhpi.de](#), the cookie required by CodeOcean is not accepted and consequently the login fails. A possible solution we implemented is to provide a proxy

⁴⁰ https://guides.rubyonrails.org/action_controller_overview.html

⁴¹ <https://www.apple.com/safari/>

⁴² <https://webkit.org/blog/8613/intelligent-tracking-prevention-2-1/>

for CodeOcean and serve it through a domain of the [HPI Schul-Cloud edtr.io](#). In general, this solution requires correct configuration of a proxy and needs to be reconsidered for use in production environments with respect to cookie security.

Independent of the problem introduced by the cross-origin usage of CodeOcean, multiple practical programming exercises in a single worksheet require further investigation. Per default, Ruby on Rails provides countermeasures for **Cross-Site-Request-Forgery (CSRF)**, a web attack that initiates an [HTTP](#) request on a third-party site without the user's consent. As a countermeasure, Rails provides a so-called [CSRF](#) token to the client and expects it in return together with most [HTTP](#) request types, such as *POST* requests. For legitimate requests, the correct [CSRF](#) is automatically included to the request by the browser. The Rails backend server validates the [CSRF](#) token with the aid of the encrypted session cookie also included in the request. For every page load, Rails generates a new [CSRF](#) token; it is included in the generated [HTML](#) and put globally in the encrypted cookie. As long as the user interacts with only one CodeOcean view at a time, this mechanism works seamlessly, as all requests are sequential. However, a worksheet might include two or more instances of CodeOcean that are potentially loaded simultaneously. Therefore, we disabled the [CSRF](#) protection temporarily for our evaluation with students. Otherwise, the newest cookie received would overwrite older versions of the same cookie including the matching counterpart for the [CSRF](#) token validation. Thus, the browser might send a request using a [CSRF](#) token from the CodeOcean instance loaded first together with the cookie of the last loaded page view. As a result of that request, the server would reject the request due to the mismatch of the [CSRF](#) token and cookie. Having outlined the issue and conceptualized a corresponding solution, we postpone the implementation to future work, as a clean and tested implementation would exceed the scope of this thesis.

To ensure that our concept and the prototype fit the needs of students and teachers, we regularly asked for feedback within 27 interviews in total. We did not only use those contacts for the initial need finding but also to clarify questions that arose during further development. The teachers we interviewed use different tools and approaches in their day-to-day lessons and have various backgrounds. While some use the [HPI Schul-Cloud](#) or an [LMS](#) regularly, others do not want to work with these tools or are unable to use them due to organizational barriers. Some have used [MOOCs](#) with their classes before, while others have not done so yet. We also presented numerous versions of our prototype to teachers and students to incorporate their feedback as early as possible.

This chapter is structured as follow: First, [Section 7.1](#) provides background information on our initial need finding we conducted. Based on the exploration of requirements, we created a software prototype and evaluated it with students of two different classes. As part of the evaluation, students voluntarily participated in our study (see [Section 7.2](#)). After using our prototype in their lessons, the teachers involved in our study described a shift from instructor-oriented to individual learning (cf. [Section 7.3](#)) and thus a change in their role. Other teachers not involved in the in-class usage got the opportunity to try our concept on a trade fair and many of them expressed their interest in using it for their own lessons. In fact, five of seven teachers would recommend edtr.io to colleagues (see [Section 7.4](#)). Finally, [Section 7.5](#) summarizes the findings of our evaluation.

7.1 EXPLORATION OF REQUIREMENTS

The initial motivation of this thesis (as described in [Section 1.1](#)) is based on direct feedback from teachers and the needs they expressed. We got in touch with teachers on the [HPI Schul-Cloud Forum 2018](#)¹, a yearly meeting of teachers using or being interested in the [HPI Schul-Cloud](#). Additional teachers for our initial exploration were found through a global announcement in the [HPI Schul-Cloud](#) email newsletter. Our goal was to get an understanding of the current situation of teachers and to identify their most important needs.

¹ <https://hpi.de/veranstaltungen/hpi-veranstaltungen/2018/schul-cloud-forum-2018.html>

7.1.1 Methodology

Our work in the context of programming education has two main stake holders in the school context: teachers and students. Throughout our initial need finding, we concentrated on teachers for two reasons: (1) Teachers define the methodology of lessons, their design and decide which tools and resources to use on their own. (2) Only if teachers notice a benefit, they will introduce a new tool to their students and use it regularly. Thus, we decided to address the requirements of teachers in a first step and consider these as top priorities for building our prototype. Otherwise, our concept could be great for students in terms of development support but could miss the needs of teachers (such as submission handling) preventing the regular use in schools. Nevertheless, we were eager to get direct feedback from students using our software prototype in a second step and therefore conducted surveys across students and teachers.

At the [HPI Schul-Cloud Forum](#), we offered a 90-minute workshop for teachers on interactive exercises to foster a discussion about approaches for practical programming exercises. During the workshop, we introduced teachers to the possibilities of CodeOcean and requirements to enable automated feedback through unit tests. At the end of our session, we provided a custom survey to the participants asking about the tools and approaches they use for computer science lessons. For the telephone interviews, we specifically focused on the current usage of worksheets, templates for source code, and the experiences teachers made with submission systems. Depending on the time our interviewees had, we outlined our concept at the end of the phone call and asked for feedback about it.

7.1.2 Results

The initial survey at the [HPI Schul-Cloud Forum](#) was answered by ten teachers, including six computer science teachers. Out of all ten teachers, two already used [MOOCs](#) in the past and four consider using them with their students. Four out of five computer science teachers indicated to be interested in online programming tools (cf. [Section 3.6](#)). Only one of six teachers used automated grading tools in the past for submissions, the others refrain from using such tools, mostly because of missing time or because the usage is too complicated. According to our survey and the telephone interviews, most teachers provide their students with scaffolded source code for practical programming exercises in order to focus their attention. The possibility to enable students to comment their submissions mutually was rated as helpful (4 replies) or very helpful (one reply). Other results from our initial survey are located in the [Section C.1](#). From the

additional eight teachers we interviewed via phone, no one used unit tests for automated grading. One teacher with access to a *Moodle LMS* used digital multiple-choice quizzes sporadically for grading.

7.1.3 Discussion and Interpretation of the Results

The telephone interviews and the results from the initial survey were comparable and provided the starting point for our analysis of the current situation in schools (see [Chapter 4](#)). Our concept, if discussed, was seen as valuable and a number of teachers were interested in following up with the further progress. Two computer science teachers agreed on testing our prototype with their classes as part of the regular lessons. Both teachers were attracted by the openHPI course on Java and intended to use the course with their students directly (confirming our survey). They described the ability of CodeOcean to provide automated feedback to the learners as one of the biggest advantages. One teacher, who already used an earlier iteration of the openHPI course on Java, criticized the missing ability to adapt scores of his students, in case he was satisfied with the solution but the student did not achieve a full score on CodeOcean. Hence, he was looking forward to our prototype allowing teachers to adapt exercises and feedback to their needs. The feedback gathered through the interviews and the survey encouraged us to continue evaluating interactive worksheets with programming exercises.

7.2 STUDENTS: TESTING THE PROTOTYPE

Measuring effects of our concept, it is important to consider how students and teachers evaluate our approach and to understand how digital worksheets with interactive programming exercises might change the in-class situation. Therefore, the main goal of our evaluation was to get early feedback from typical lessons using our prototype. As shown in [Table 7.1](#), we tested a total of four different worksheets in five lessons with two classes (consisting of 21 and 16 students, respectively).

To minimize the side effects of using a web-based programming environment and online videos, we conducted our evaluation with two classes that were learning Java with a free [MOOC](#) on openHPI or a repetition of the same course on mooc.house. In these courses, videos are followed by multiple choice questions and practical programming exercises in CodeOcean. For a first test run, we prepared two worksheets on the Java topics *methods* and *inheritance* in tight consultation with the teachers and inspired by the openHPI [MOOC](#). Both worksheets were used in the same 90-minute lesson, starting

GROUP	TESTED WORKSHEETS			FILLED SURVEYS		
	METHODS	INHERITANCE	ABSTRACT CLASSES	GRADED ASSIGNMENT	FREE TEXT QUESTIONS, NPS, UEQ	VIDEO VS. BOOK
class of the first teacher (21 students)	1.	2.	4.		3.	5.
class of the second teacher (16 students)		2.		1.	3.	4.

Table 7.1: Overview about the worksheets tested in different lessons by two teachers and their students. The numbers represent the order in which the worksheets or surveys were given to the students.

with the worksheet on *methods*. From teachers and previous iterations of our online courses, we know that the *inheritance* topic is one of the more difficult concepts in programming education. By starting with an easier repetition on *methods*, we were able to reduce possible distraction introduced through the digital worksheets for the second, more complex subject. Each worksheet consisted of a mix of text elements, images, videos, multiple-choice quizzes and several practical programming exercises in-lined.

7.2.1 Methodology

The designed worksheets were used during a regular lesson by a combined class of 21 students in their last two years of high-school. We decided not to join the class in person to prevent any disturbance due to our attendance. Therefore, it was the teacher's responsibility to introduce the topic as he normally does and to guide the students through the material. After the lesson, the students were asked to provide anonymous feedback through our survey. Furthermore, we consulted the teacher directly to learn more about the in-class situation and his impressions. In the survey given to students, we asked what they liked or disliked about the concept in general. Additionally, we asked which advantages or disadvantages the different mediums, i. e., the schoolbook, traditional worksheets, static PDF worksheets, and our approach offer.

Moreover, we assessed the Net Promoter Score (NPS; how likely it is that students would recommend the digital worksheet to friends) as described by Reichheld [50]. Based on their answers given on a scale from 0 to 10, students can be assigned to one of three groups: The promoters (rating of 9 or 10, likely to recommend digital worksheets),

the passives (rating of 7 or 8) and the detractors (rating of 0 – 6; these are unlikely to recommend digital worksheets). The NPS is calculated as the difference between the promoters and the detractors divided by the number of participants multiplied by 100. Consequently, the NPS ranges from +100 (every participant is likely provide a recommendation) to -100 (no one is likely to give a recommendation).

We further embedded modules from a questionnaire called **modular evaluation of key Components of User Experience (meCUE)** [43]. The meCUE consists of four modules, covering the product perception (instrumental: module I; non-instrumental: module II), emotions (module III), consequences (module IV) and an overall score (module V) [43]. For our survey, we included the first, third, fourth and fifth module, as we expected them to provide the most suitable insight.

Eight weeks after the first test with two worksheets, we conducted a second evaluation with a third worksheet on the Java topic *abstract classes*. This time, we also included a reference to the schoolbook as students had to work on their own with the teacher being absent. The experiment was complemented by a survey with just three questions comparing the book and the embedded video with regard to the comprehensibility, the perceived enjoyment and the preferred source for the repetition of content.

The female teacher of another class with 16 students was using the course on mooc.house and was interested in a graded assignment. She previously used the weekly homework assignments from the MOOC and scores calculated by the course platform. However, she was surprised by the good results many students achieved. She attributed some of the good results to the structure of the MOOC and the availability of the assignment to all users prior to her lesson. Furthermore, she was unable to review the submissions and had to trust the score calculated by the MOOC platform. Thus, for our third evaluation, we created a worksheet with two graded assignments and disabled the internal scoring feature for students (with the mechanism described in Section 6.4.2).

A few days later, the same class (with 16 students) used our existing worksheet on the Java topic *inheritance* instead of the course content directly available within the course. We also asked the students to participate in the same survey we prepared for the other class. We also asked the students to compare books and videos with regard to the comprehensibility, the perceived enjoyment and the preferred source for the repetition of content.

7.2.2 Results

From the 21 students who accessed the first worksheet, nine students participated in our voluntary survey. Eight students valued the general concept and the overall design of our interactive worksheets. The students further appreciated the provided overview and structure on the content of a lesson. They also valued the general availability of online content, as this allowed them to access content independent of their current location, e. g., while commuting in a bus on their way to school. The complete results of the survey is attached in [Section C.3](#). We received an [NPS](#) of -38 ([Table 7.2](#); ranging from -100 to +100), based on eight answers in our survey. In contrast, answers to the [meCUE](#) show an overall score of 2.7 on average (ranging from -5 to +5; N=7). Other scores in the [meCUE](#) ([Figure 7.1](#); ranging from “strongly agree” = 7 to “strongly disagree” = 1, with their respective average score based on six replies) were usability (6.0), usefulness (4.3), positive emotions (3.2), negative emotions (2.9), intention to use (3.2) and product loyalty (3.2).

GROUP	PROMOTERS	PASSIVES	DETRACTORS	NPS
class of the first teacher (21 students)	1	3	4	-38
class of the second teacher (16 students)	0	2	6	-75
all students	1	5	10	-56
all students and teachers	6	7	10	-17
all teachers	5	2	0	+71

Table 7.2: Overview of the [NPS](#) achieved across all students and teachers. The [NPS](#) ranges from -100 to +100 and describes, how likely it is for a product to be recommended. A higher value indicates that a recommendation to a friend or colleague is more likely.

The second survey comparing the schoolbook with videos from a [MOOC](#) only received three answers from the first class: One student preferred the video for learning and attributed it was more enjoyable than the schoolbook, while the other liked both approaches to the same extent. The two students agreed to use both resources for repeating the content. A third student preferred the schoolbook in all categories and expressed a general rejection of learning with digital resources.

From the second class with 16 students, ten students gave feedback through our voluntary survey after using the interactive worksheet

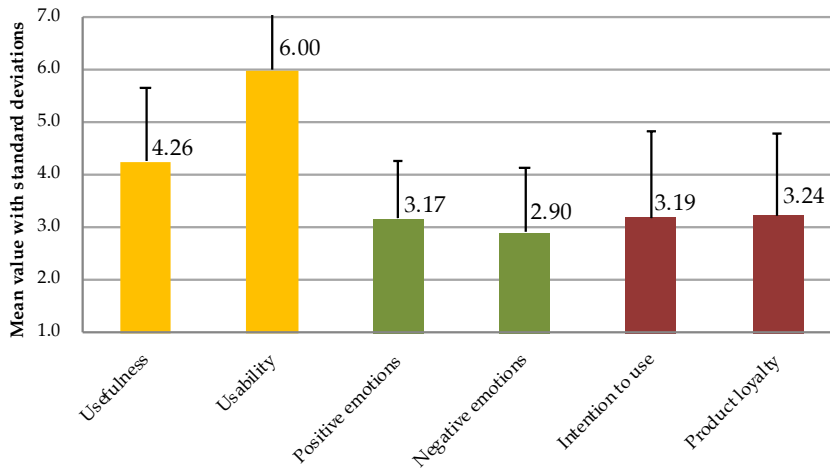


Figure 7.1: Results of meCUE modules I (yellow), III (green) and IV (red) with the standard deviation for the class of the first teacher (N=7).

on *inheritance*. All participants found the worksheet clearly structured and stated to find it easy to learn with different content types (such as videos and texts) as offered by our worksheet. Nevertheless, more students prefer learning with the mooc.house course than with the worksheet (four vs. three answers, three students neither prefer one or the other). Furthermore, eight out of the ten students indicated they had technical issues during the lesson. Consequently, the NPS we received from eight answers in our survey was -75 (as shown in Table 7.2). However, the overall experience measured with the meCUE was positive with 1.4 overall (eight replies from the same students that also answered the NPS). While some other scores of the meCUE (calculated from four replies and shown in Figure 7.2) were

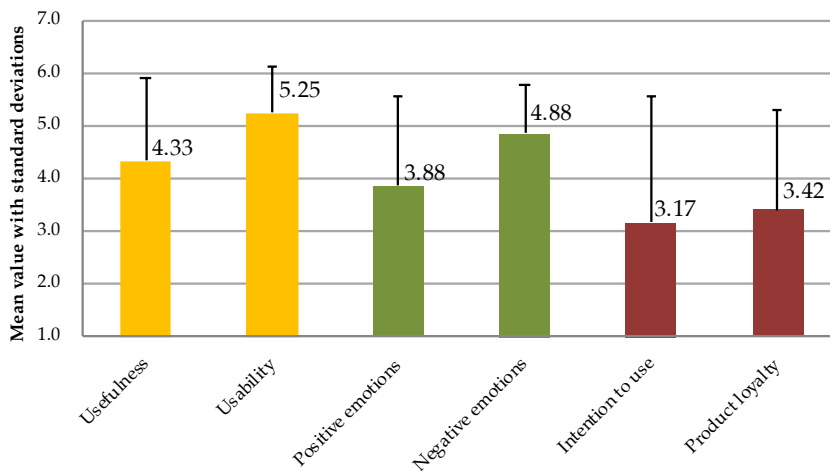


Figure 7.2: Results of meCUE modules I (yellow), III (green) and IV (red) with the standard deviation for the class of the second teacher (N=4). The results were affected by technical issues [48].

significantly different to those of the other class (such as usability (5.3), positive emotions (3.9), negative emotions (4.8)), others were almost the same (e. g., usefulness (4.3), intention to use (3.2) and product loyalty (3.4)). We provide an overview about the scores of the **meCUE** for both classes in [Figure C.4](#) and [Table C.1, Section C.2](#).

In our supplementary survey (cf. [Figure 7.3](#)), we asked students of the second class about their preferences on learning with books or videos. 13 students participated in our survey and provided an answer per question on a four-element-scale indicating whether they agree or disagree with the given statement. Eleven students indicated to understand concepts better with videos than with books while two preferred a book for the initial learning. All students except for one enjoy learning with videos more than they do with books. Even though textual descriptions were not preferred by the majority of students, books are most likely to be used for repeating content. Throughout all three questions, the video surpassed the book and was generally preferred by the students.










Statement	Answer Option	Students	Visualization
I understand concepts better with books than with videos.	Strongly agree	0	
	Somewhat agree	2	
	Somewhat disagree	11	
	Strongly disagree	0	
I generally enjoy learning with videos more than with books.	Strongly agree	6	
	Somewhat agree	6	
	Somewhat disagree	1	
	Strongly disagree	0	
I would rather use a book to repeat learning content.	Strongly agree	3	
	Somewhat agree	3	
	Somewhat disagree	5	
	Strongly disagree	2	

Figure 7.3: Results of the additional questions asked amongst students of the second class. The three scales measured were the comprehensibility, the perceived enjoyment and the preferred source for the repetition of content (class of the second teacher, N=13).

7.2.3 Discussion and Interpretation of the Results

The results gathered with the questionnaires are a first evaluation and motivation to further research the impact of interactive worksheets. Students from both classes highlight the same aspects of interactive worksheets independent of each other. While one class was used to worksheets within computer science classes, they found worksheets

more important than the other class not using worksheets on a regular basis for programming education. Hence, the class with more worksheet experiences recognized more advantages in our interactive worksheets than students from the other class did. That might be, besides the technical influence, one of the reasons for the difference in the [NPS](#) and [meCUE](#). Comparing books and videos, the video outperformed the book in all three categories we measured, based on a total of 16 replies from both classes. Hence, videos (e. g., as part of [MOOCs](#)) are a valuable addition to traditional teaching methods.

In their paper, Partala and Kallinen stated that “the subjects also reported [...] more technical problems for the most unsatisfying than the most satisfying experiences” [48]. Further, the authors conclude “[that] the clear differences in the ratings of technical and usability problems between the most satisfying and most unsatisfying user experiences suggest that the central role of these aspects should not be forgotten, when studying user experience” [48]. Our investigation with the teacher revealed connection issues within the school network that occurred while her students used the interactive worksheet on *inheritance*. As a result of the work presented by Partala and Kallinen and our analysis, we decided to limit responses of the [meCUE](#) and [NPS](#) to those users without known issues. Non-emotional questions (e. g., those regarding the structure provided by the worksheets) were not affected. In comparison, the ability to develop source code within a browser-based code execution environment during a graded assignment was praised by the students.

7.3 TEACHERS: EXPERIENCES FROM TESTING THE PROTOTYPE

In addition to the direct feedback of students we collected by the survey, we were especially interested in the experiences the two teachers made with our interactive worksheets. Therefore, we interviewed them independent of each other via a telephone call. Our main research interest was to understand the differences between learning with the [MOOC](#) and using our worksheets. Thus, we asked teachers to observe the behavior of their students during the lessons.

After the very first lesson including our worksheets, the teacher reported that his students got along well. He valued the deep integration and material mix we provided and repeatedly expressed the intention to create equivalent worksheets himself. From his observation, students revisited previous parts of the worksheets more often while working on the programming exercises than they did in the setting with the [MOOC](#). The other teacher, who also used the same worksheet on *inheritance*, confirmed that observation for students which had no or only minor technical issues.

Using our worksheets in class, the teachers delightedly noticed a shift in their role compared to other lessons without our prototype: Students were able to execute their programs and get automated feedback and hints through CodeOcean as previously, but were further supported by the improved structure to revisit previous learning content on the same worksheet. Thus, students were able to answer more questions themselves and proceed at their own pace. Hence, teachers were able to focus on struggling students and dedicate their time to support them. Consequently, the role of the teachers changed from an instructor to an individual tutor.

During lessons, students like the ability to ask upcoming questions orally and discuss them with their fellow students or the teacher. Students and teachers prefer the direct communication over using any technical tool as no formalization of the questions is required. Answers are given by teachers immediately and can optionally be supported by a visualization. Thus, the possibility to create *Request for Comments* was not used during lessons. However, the teachers we interviewed were interested in commenting submissions to provide feedback for graded assignments, similar to *Request for Comments* in CodeOcean.

Furthermore, we provided the teacher who used one worksheet for a graded assessment with the submissions of her students. The scores of exercises used within the test were neither visible for the teacher nor for the students (due to settings applied for CodeOcean). After she finished the grading on her own, we showed her the scores calculated by CodeOcean based on the unit tests we added for each exercise in advance. She was surprised how close our test results were compared to her manual grading. For example, the maximum score of one exercise was six points and 9/10 results were identical or within a range of ± 0.5 points. For the teacher, the overall dashboard visualizing the progress made by her students had the biggest impact. She would have preferred to have access to the graphs throughout the grading. In her opinion, it would allow her to define an order for her review process based on the pre-evaluation of the submissions. As she was satisfied with the in-class experience, she expressed her interest in using our prototype for additional graded assignments.

7.4 TEACHERS: USABILITY OF THE PROTOTYPE

In addition to the feedback we got from the two teachers who were involved in testing the prototype, we asked other computer science teachers about their impressions of our tool. As we learned from the initial need finding that most teachers have individual approaches to programming education in high-schools, we were eager to show our tool to as many teachers as possible. Our goal of the evaluation

with teachers was to get an understanding of the applicability of our concept and the usability of our prototype for computer science lessons. Specifically, we focused on the (in-class) usage of worksheets, the adaptability of CodeOcean exercises and the subsequent grading of submissions.

7.4.1 Methodology

We introduced our prototype with sample documents and learning analytics to teachers on didacta 2019², one of the largest European education trade fairs and on the HPI Schul-Cloud Forum 2019³. During these events, we presented our tool in one-on-one sessions or to small groups of up to four participants. We gathered different ideas from teachers and asked them for feedback using the standardized User Experience Questionnaire (UEQ) [34]. The UEQ includes a benchmark based on 401 studies and more than 18.000 participants to compare results from own studies. In addition to the UEQ, we also asked teachers how likely it was that they would recommend interactive worksheets to their colleagues (in order to determine an NPS for teachers).

7.4.2 Results

Out of seventeen teachers who tried our prototype (student as well as teacher role), twelve teachers filled the UEQ. The scales tested turned out to be mostly rated good or even excellent (cf. Figure 7.4). The stimulation (mean of 1.7) provided through the use of interactive worksheets was rated the best, together with attractiveness (1.7), dependability (1.5), efficiency (1.5) and novelty (1.1) rated good. The perspicuity (1.4) was seen above average with the highest improvement potential, compared to the benchmark provided by the UEQ. All detailed results gathered by the UEQ are located in Section C.4, Figure C.10 and Figure C.11. Across seven teachers, the NPS of our interactive worksheet editor with integrated learning analytics was +71 (see Table 7.2).

7.4.3 Discussion and Interpretation of the Results

To be able to classify our NPS value of +71, we compared it to other scores publicly available. According to a benchmark offered by De-

² <https://www.didacta-cologne.com/>

³ <https://hpi.de/en/events/hpi-veranstaltungen/2019/schul-cloud-forum-2019.html>

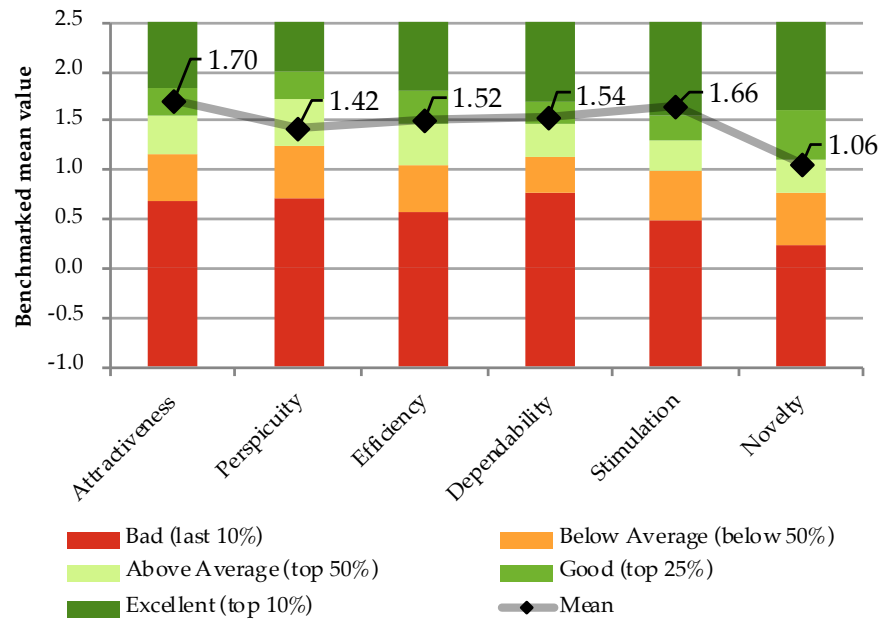


Figure 7.4: Measured scales of the UEQ showing mean values in comparison with the method benchmark (N=12).

lighted⁴, the NPS within the software industry⁵ typically ranges from +28 to +55 with an average of +41. Duolingo⁶, an education software to learn languages, got an NPS of +51.5 according to a study conducted by Vesselinov and Grego [67]. The authors conclude that “this [score] is an excellent result”, which is based on 66 responses [67]. Although future studies should be conducted with additional iterations of our prototype and a larger audience to draw generalizable results, our score of +71 (N=7) provides a first impression and indicates the satisfaction across teachers that participated in our study.

According to Laugwitz et al., “the scales of the UEQ can be grouped into pragmatic quality (Perspicuity, Efficiency, Dependability) and hedonic quality (Stimulation, Originality)” instead of an overall score [34]. The pragmatic quality describes the usefulness, while the hedonic quality defines the pleasure and motivation of users to handle the product [34]. The scale for both is ranging from -3 (“horribly bad”) to +3 (“extremely good”) and values between -0.8 and +0.8 are seen as a “neural [sic] evaluation of the corresponding scale” [34]. For our prototype, both qualities (as shown in Figure 7.5) were rated good within the given interpretation with the pragmatic quality (1.49) being slightly better than the hedonic quality (1.36). As we mainly focused on the features and usability, the scores affirm our main effort towards the pragmatic quality.

4 <https://delighted.com/nps-benchmarks>

5 Companies in the benchmark include Adobe, Apple, Google, Microsoft, Symantec and others

6 <https://duolingo.com>

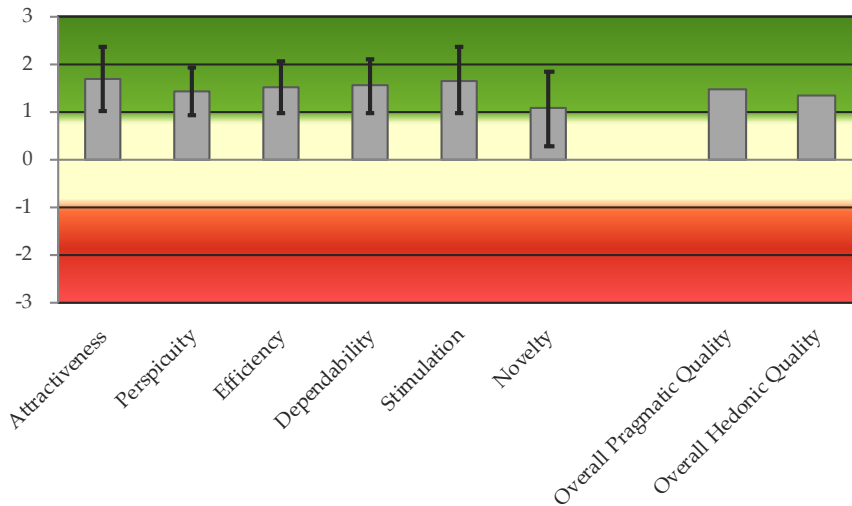


Figure 7.5: Results of the UEQ in the six scales showing the standard deviation and the grouping into pragmatic quality and hedonic quality (N=12).

7.5 OVERALL IMPRESSION

The limited responses from students and teachers allow cautious conclusions to be drawn. The responses of students show contradictions, such as the difference in the NPS (-38) and the meCUE overall experience (2.7). Free text answers also showed minor contradictions, hinting that students might have misinterpreted parts of our survey. The negative NPS is likely to be explained by technical issues caused by connection problems from within the school network as the overall experience was valued, which was also confirmed by the teachers.

We further draw three main conclusions from the text comments of the students received in the surveys: (1) When comparing interactive worksheets with MOOCs, students highlight the improved structure and the enhanced overview offered through the worksheets. (2) They value the mix of different content types on a single web page and see no additional advantage in accessing the MOOC platform instead. (3) Traditional worksheets are seen as static and inflexible; however, students are sometimes allowed to use traditional worksheets for their reference in a written exam. As digital devices are usually not allowed, we derived the need to work on a print mode in the future. Overall, students value the integration of online learning content and the introduction of hybrid classes. In their opinion, MOOCs offer state-of-the-art knowledge while schoolbooks can be outdated.

The biggest advantages of interactive worksheets are voiced by teachers. They favor the efficiency offered by edtr.io to create individual worksheets with customized programming exercises. Our qualitative survey shows that our concept meets the requirements

computer science teachers have and the results of the [NPS \(+71\)](#) and the [UEQ](#) support the proposed design. Due to the insights provided by learning analytics, teachers feel better prepared when planning upcoming lessons and gain a better understanding of the reasons why their students struggle with given exercises. During lessons, teachers observed that students learned individually at their own pace and rearranged the topic order to their needs. Therefore, teachers reported that the worksheets helped to change their own role from an instructor of the class to an individual tutor. The automated feedback freed up some of their time, which they were now able to spend on helping individual learners. The positive feedback of teachers using our initial prototype of digital worksheets in their lessons, combined with the repeatedly voiced demand to create own worksheets, underlines the relevance and practical applicability of our concept.

OUTLOOK AND FUTURE WORK

Identified shortcomings such as the missing usability of digital worksheets during exams should be solved with a feature to print out individual filled worksheets or the material of a complete lecture series. While a simple print-out of an interactive worksheet provides students with a copy of their progress, many of the advantages of digital worksheets cannot be preserved. Traditionally, the exam is also paper-based, resulting in the absence of features students used during learning. Future work should, therefore, concentrate on the special requirements during exams and introduce an “exam-mode” for interactive worksheets. In this scenario, teachers need to get increased control over resources accessed by students and the features enabled in the programming exercises, such as the set of limitations available in edtr.io and CodeOcean (see [Section 6.4.2](#)), e. g., by introducing a time limit and an explicit submission handling.

A cockpit view for teachers, showing the students’ progress and allowing presentation and discussion of individual solutions is valuable in lessons as well as exam situations (as outlined in [Section 7.3](#)). The presentation view enabling teachers to do an in-class comparison of different solutions should be evaluated against currently used approaches.

For exam usage of our tool, we plan to further improve the technical stability. Despite already being stable in [MOOC](#) usage, we consider shifting from our current custom container pooling to Docker Swarm or Kubernetes to reduce maintenance effort and minimize error scenarios. Additionally, the learnings from [Section 6.7](#) should be considered, such as implementing the [CSRF](#) fix we outlined. Future work will also concentrate on ways how to provide debug functionality for novices.

As the adoption of interactive worksheets highly depends on the ease of use for teachers to create own worksheets and the variety of already existing documents, we will further investigate options to share worksheets and embed other types of pre-existing [OER](#). Lastly, we will re-run our experiments (cf. [Chapter 7](#)) with larger study groups to achieve reliable and potentially generalizable outcomes.

CONCLUSION

Computer science education in high-schools includes practical programming exercises which impose technical requirements on school computers. Therefore, teachers like online programming environments that can be used without any local setup. Those environments are available as stand-alone web applications or integrated into MOOCs including learning content and beginner-friendly programming exercises. Computer science teachers value the online content and partially reference it in traditional worksheets, which only offer a cumbersome and static user experience. To enable teachers to adapt learning content to the respective needs of their class, we developed the concept of interactive worksheets with embedded programming exercises. On the basis of our experiments, we are able to answer our research questions:

RQ1. How can we enable teachers to reuse and adapt exercises (e. g., from MOOCs) and create their own interactive worksheets?

Teachers not only need access to the same authoring tools available to MOOC instructors but require further technical help in creating equivalent exercises. In online courses, automated feedback is used to support learners in finding the correct solution, which teachers regard as helpful. To integrate the same feedback mechanism for their own programming exercises, custom unit tests are required. We propose to offer a unit test generator to help teachers developing such tests. In addition, teachers should be enabled to access and share exercises with colleagues. In our concept, all exercises can be embedded in an interactive worksheet with a customizable integration providing teachers with fine-granular control on the features available to students.

RQ2. Which tooling support do teachers need to help students struggling with given programming exercises?

Besides access to student submissions on demand, teachers benefit from the developed dashboard featuring a limited set of learning analytics of their students. Given a visualization of the students' score together with programming errors they made, teachers gain a better understanding of potential problems. Hence, they can support struggling students in a targeted and faster way as well as better prepare upcoming lessons.

RQ3. How can we leverage learning data to enhance teaching effectiveness and help the teacher to achieve lesson goals?

Live processing of learning data, as featured by our tool and supported by the evaluation, is helpful for teachers to answer upcoming questions and to detect potential misconceptions. Independent of single worksheets, existing [LRS](#) in schools offer customizable views for evaluating long-term progress.

RQ4. Which support do students need to get individual help, either from their fellow students or from their teacher?

During a lesson, teachers and students wish no further technical support as they value the direct, oral communication for emerging problems. However, students benefit from tools to help each other or contact the teacher in case of questions while working on homework. CodeOcean supports commenting on lines of source code to provide users with the full context of a question.

RQ5. Which of the results gathered from the questions above can be transferred to the general [MOOC](#) context?

The improved structure of the worksheets was appreciated by students and our evaluation suggests that it supports learners in revisiting learning material while solving programming exercises. [MOOCs](#) will also benefit from grouping learning material of the same topic to ease navigating in the content.

Furthermore, our research suggests that the concept of study groups should be applied to [MOOCs](#), enabling a tutor to guide learners through the course. In this scenario, tutors could be given control over the content visibility or deadlines and might also get access to learning analytics of their study group to enable informed interventions.

As an additional outcome to the answers we provided for the research questions, we contributed to CodeOcean and edtr.io: A multiple-choice plugin was implemented for the worksheet editor together with support for eleven restrictions teachers may apply when embedding practical programming exercises. CodeOcean was extended with a dedicated teacher role, study groups and a live dashboard for in-class learning analytics.

In this thesis, we introduced and evaluated interactive worksheets with embedded programming exercises. Our prototype of a worksheet editor features an extensible plugin architecture for various learning tools. By combining text, videos and interactive content including practical programming exercises and multiple-choice quizzes, we improve the status quo for students as well as teachers: Students learn individually at their own pace. Teachers switch their role to tutors and individually support their students based on well-informed analytical insights. Feedback is positive and shows the vast potential of introducing digital content in the classroom.

Lösen eines Arbeitsblatts in der Schul-Cloud aus Sicht eines Schülers

- ① Schüler meldet sich in der Schul-Cloud an, Auswahl des Kurses und anschließend des Arbeitsblatts
- ② paralleler LTI-Launch, Ablage der LTI-Parameter in der Session nach Aufgaben-ID getrennt (ggf. lti-parameters Tabelle)
 ↘ Deadlines?
 Zeitmessung startet, sobald Code-Editor Fokus bekommt
 jeder Score (falls verfügbar, sonst Run, sonst Ergebnis per xAPI)
 LTI: Zuerst score wie bisher übertragen, dann lis-result-sourcedid speichern und mit replaceScore ersetzen

v </> CodeOcean

Objekte + Klassen <input checked="" type="checkbox"/>	Run <input checked="" type="checkbox"/>	XScore
Dateien ↳ src ↳ superclass ↳ class	1 2 3	

Hilfe anfordern
 Frage stellen
 Output

▷ OPEN

v </> CodeOcean OK ↖ Fokus-Checker

Interfaces

→ Abgeben

1
2
3

Hilfe anfordern

Figure A.3: Conceptual sketch for the integration of CodeOcean into a worksheet and the interface provided for students while working on an exercise.

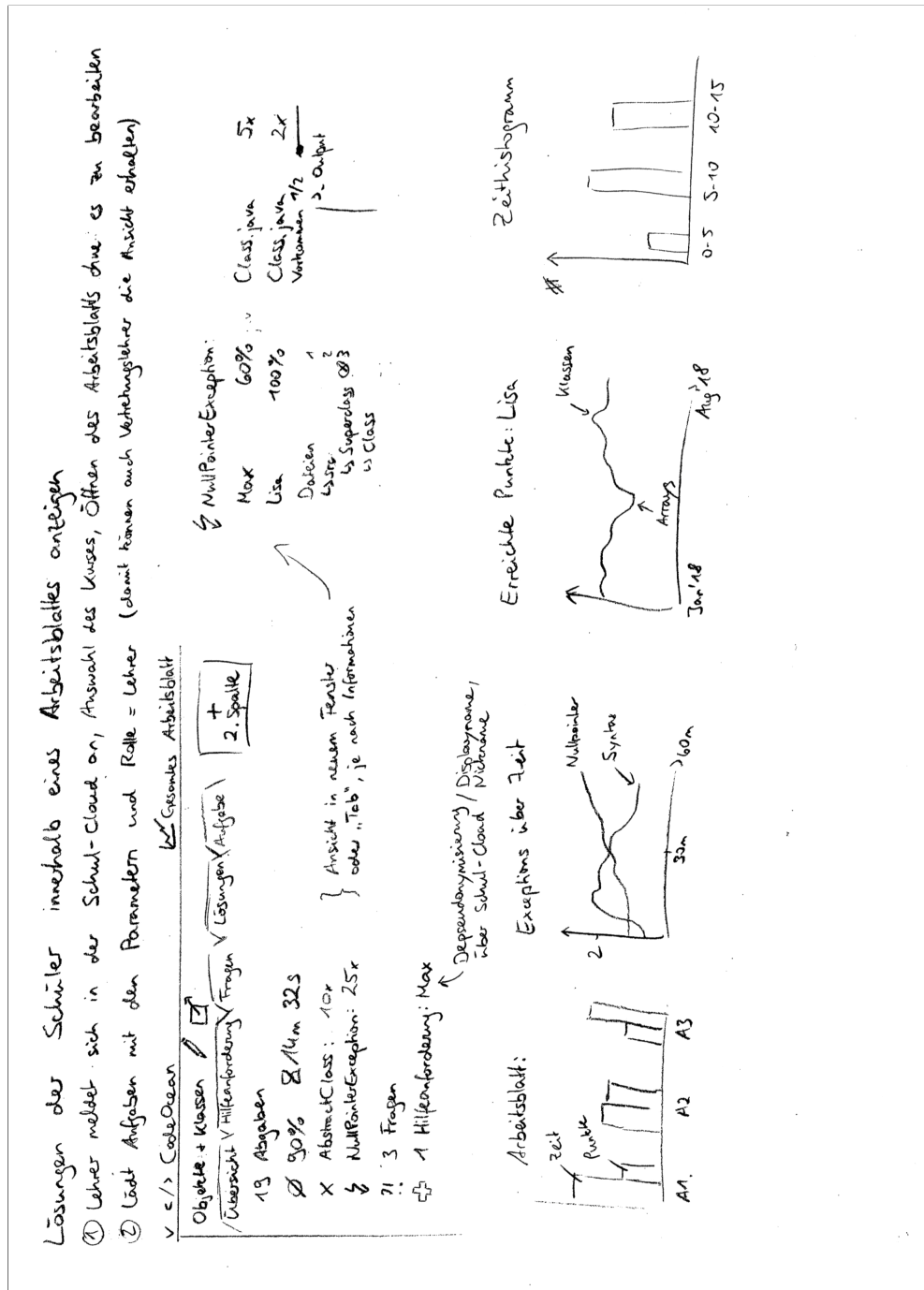


Figure A.4: Conceptual sketch outlining the submission handling for teachers and the integration of learning analytic insights into the worksheet editor.

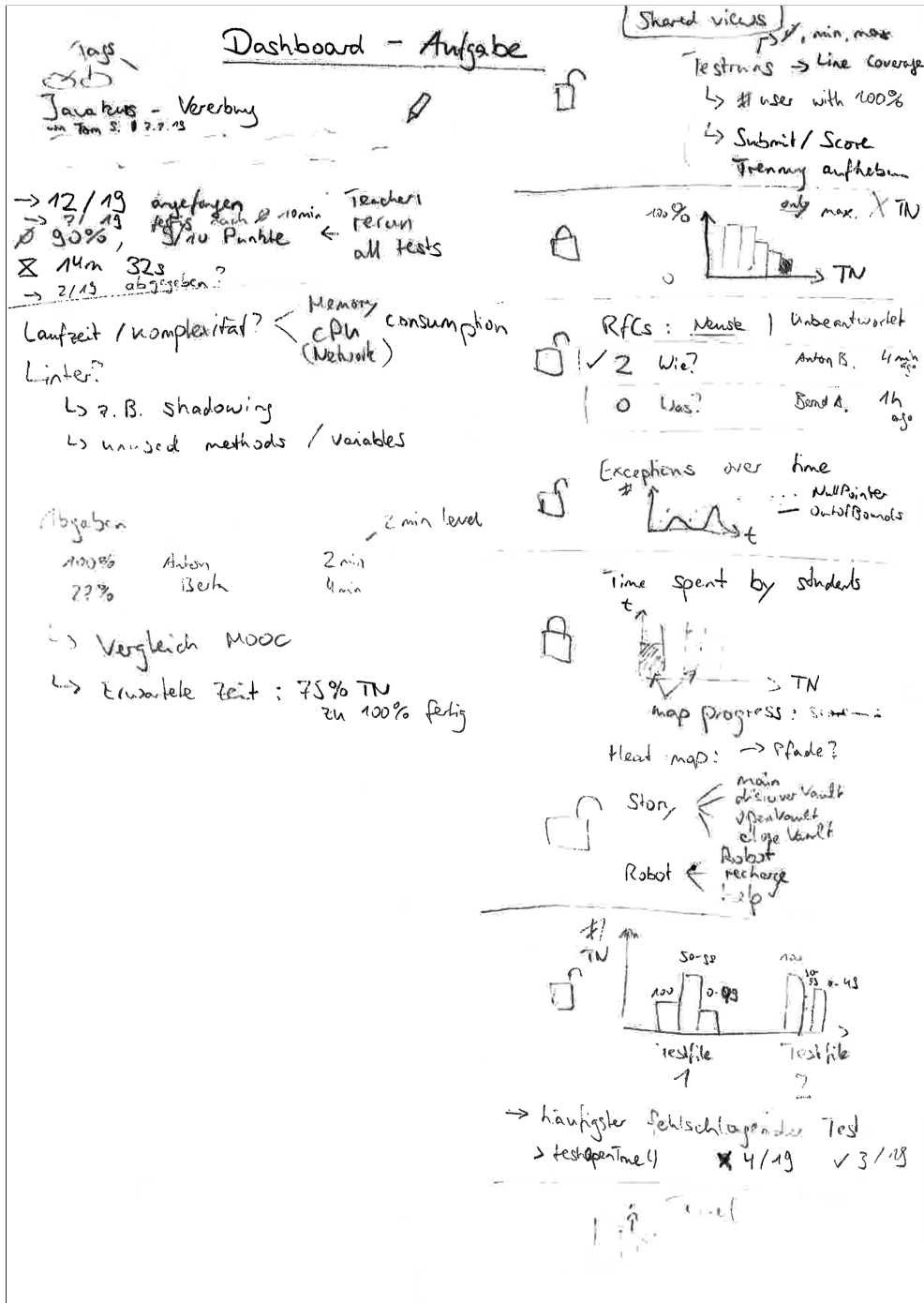


Figure A.5: Conceptual sketch with learning analytic metrics for teachers in the context of an exercise. The lock indicates whether the graph could also be shown to students, e.g., using a projector in class.

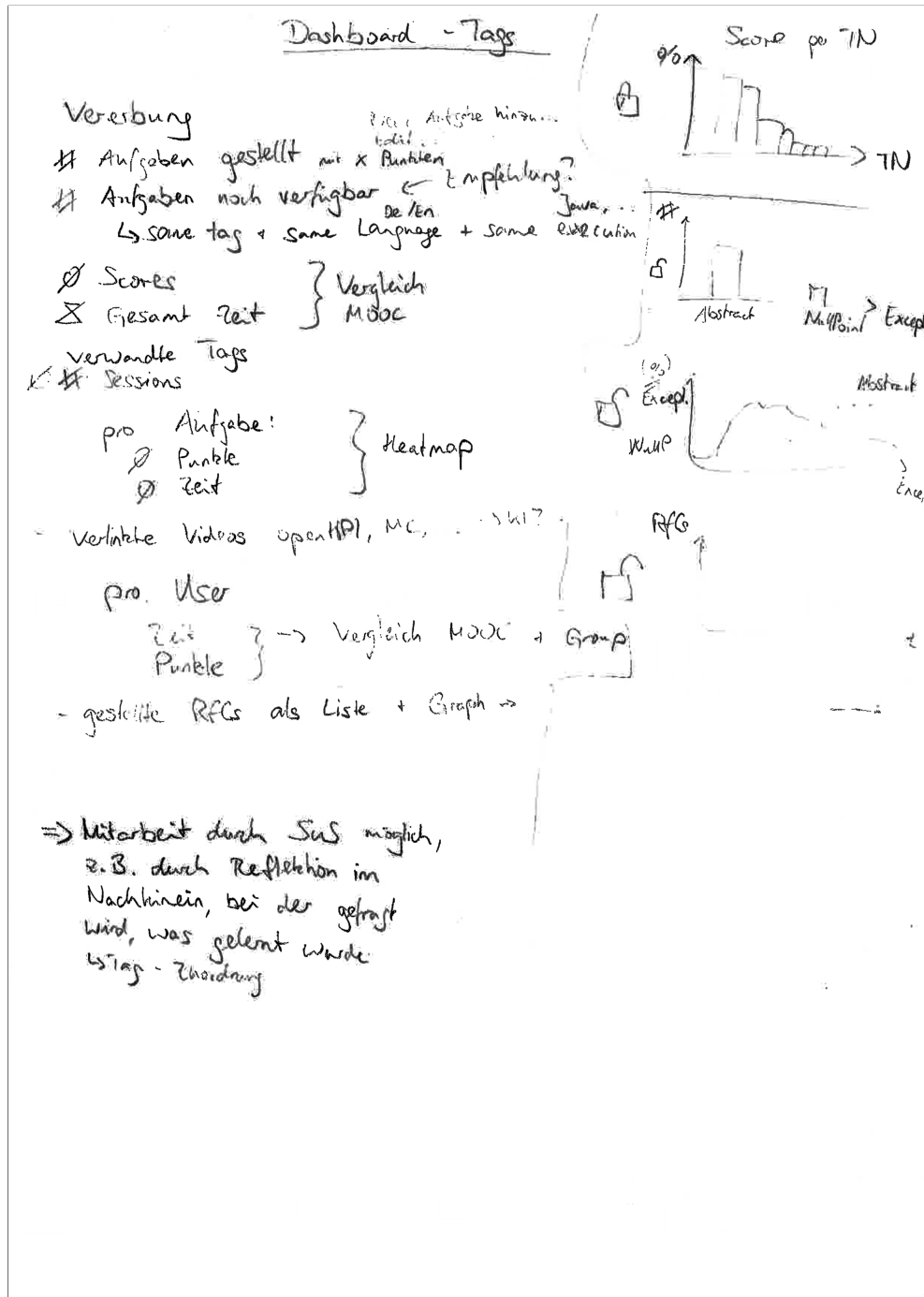


Figure A.6: Conceptual sketch with learning analytic metrics for teachers in the context of a tag (category) in CodeOcean. The lock indicates whether the graph could also be shown to students, e. g., using a projector in class.

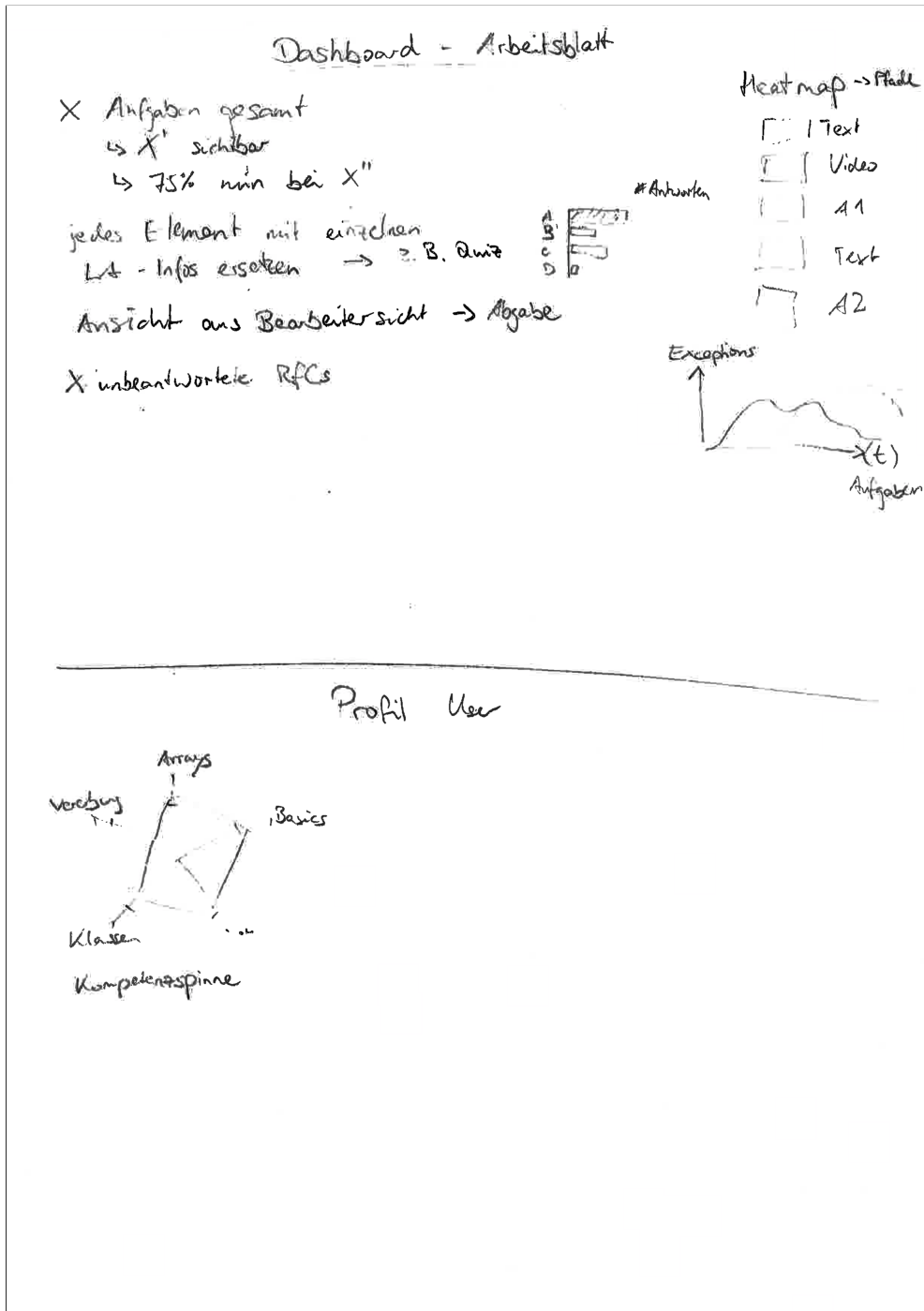


Figure A.7: Conceptual sketch with learning analytic metrics for teachers in the context of a worksheet. The lock indicates whether the graph could also be shown to students, e.g., using a projector in class.

A.2 RELATED CONCEPTS

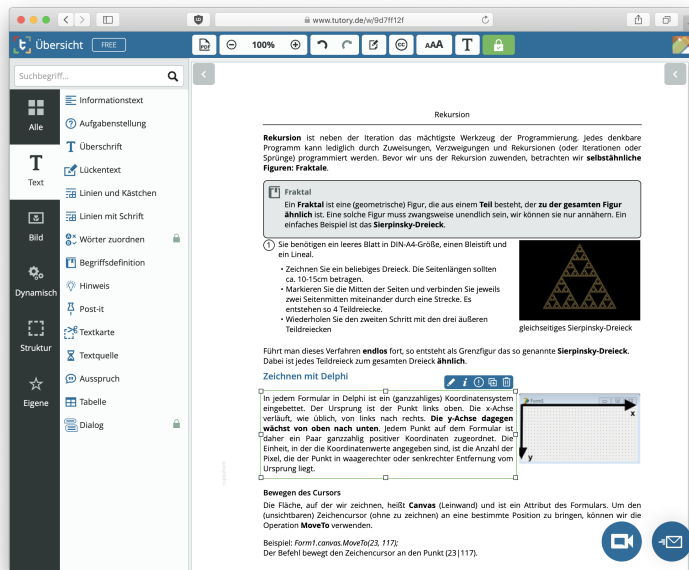


Figure A.9: Modular worksheet editor as offered by *tutary* allowing a free combination of pre-defined text and image components. The exemplary content is the same as shown in Figure 5.1 and introduces recursion in the programming language Delphi [1].

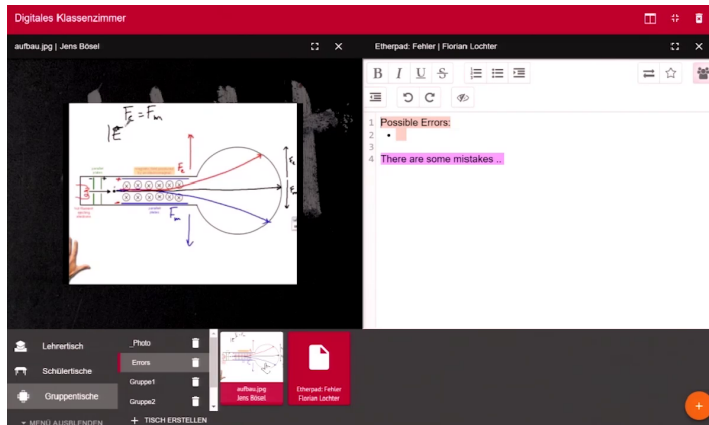


Figure A.10: The *Digital Classroom* within the *HPI Schul-Cloud* allows students to share their current progress with the teacher [29]. The screenshot is taken from the teacher's perspective, so that access to students' work is enabled and shows a text editor. In our concept, the worksheet editor would replace the image and main text.

The screenshot shows the repl.it online Java IDE interface. On the left, a file explorer shows 'Robot.java'. The main editor displays the following Java code:

```

1 class Robot {
2     int[] squares = new int[25];
3
4     void saveSquares(){
5         for (int i = 0; i < 25; i++) {
6             squares[i] = (i-1) * (i-1);
7         }
8     }
9
10    int giveSquares(int i){
11        return squares[i];
12    }
13
14    void printSquares() {
15        for(int i = 1; i <=
16            squares.length; i++) {
17            System.out.println(
18                "Das Quadrat der Zahl
19                "+ i + " ist " +
20                giveSquares(i-1));
21        }
22    }
23 }

```

On the right, the output console shows the following text:

```

Java version "1.8.0_31"
Java(TM) SE Runtime Environment (build 1.8.0_31-b13)
Java HotSpot(TM) 64-Bit Server VM (build 25.31-b07,
mixed mode)
Das Quadrat der Zahl 1 ist 1
Das Quadrat der Zahl 2 ist 0
Das Quadrat der Zahl 3 ist 1
Das Quadrat der Zahl 4 ist 4
Das Quadrat der Zahl 5 ist 9
Das Quadrat der Zahl 6 ist 16
Das Quadrat der Zahl 7 ist 25
Das Quadrat der Zahl 8 ist 36
Das Quadrat der Zahl 9 ist 49
Das Quadrat der Zahl 10 ist 64
Das Quadrat der Zahl 11 ist 81
Das Quadrat der Zahl 12 ist 100
Das Quadrat der Zahl 13 ist 121
Das Quadrat der Zahl 14 ist 144
Das Quadrat der Zahl 15 ist 169
Das Quadrat der Zahl 16 ist 196
Das Quadrat der Zahl 17 ist 225
Das Quadrat der Zahl 18 ist 256
Das Quadrat der Zahl 19 ist 289
Das Quadrat der Zahl 20 ist 324
Das Quadrat der Zahl 21 ist 361
Das Quadrat der Zahl 22 ist 400
Das Quadrat der Zahl 23 ist 441
Das Quadrat der Zahl 24 ist 484
Das Quadrat der Zahl 25 ist 529

```

Figure A.13: The web-based repl.it allows learners to create and run programs without installing local software [16]. Designed as REPL, the functionality offered for Java is comparable to Codeboard (see Figure 3.1) and other web-based IDEs.

APPENDIX: IMPLEMENTATION

B.1 EXEMPLARY WORKSHEET

Vererbung


Dieses Arbeitsblatt besteht aus 7 Teilen, darunter 3 Programmieraufgaben

Motivation:

Ein Zoo hält viele verschiedene Tiere unterschiedlichster Arten. Wenn wir den Zoo mit einer Menge von Klassen und Objekten modellieren möchten, so könnte man für jede Tierart eine eigene Klasse erstellen und pro Tier ein Objekt der jeweiligen Klasse instanzieren. Schauen wir uns dazu ein Beispiel mit Vögeln an ...

```
classDiagram
    class Bird {
        name :String
        order :String
        weight :float
        eat ()
        fly ()
    }
    class Woodpecker
    class Tit
    class Parrot
    class Penguin
    Bird <|-- Woodpecker
    Bird <|-- Tit
    Bird <|-- Parrot
    Bird <|-- Penguin
```

Video 3.1:



Vererbung

openHPI-Java-Team
Hasso-Plattner-Institut

Figure B.1: Exemplary worksheet on the Java topic *inheritance* as used in our study, consisting of text, an image, a video, a multiple choice quiz, and a programming exercise. Continued in [Figure B.2](#).

Quiz 3.1:

Wofür wird Vererbung typischerweise verwendet?

- Zur Erstellung von Beziehungen zwischen verschiedenen Klassen.
- Zur Verringerung von Dopplungen von Codezeilen
- Zur Aufteilung der Arbeit am Code auf verschiedene Entwickler.
- Zum Ausdrücken einer „is-a / ist-ein“-Beziehung.
- Zur Darstellung von verschiedenen Generationen bei Lebewesen, also einer Eltern-Kind-Beziehung

Programmieraufgabe 3.1.2:

➤ OOP2017 Woche3 Kapitel1 Aufgabe2 0%
(Anzeigen)

Ausführen Bewerten Kommentare erbitten

```
1- class Story {
2-     public static void main(String[] args) {
3-         DetectiveRobot ronja = new DetectiveRobot();
4-         ronja.speak();
5-     }
6- }
7- }
8- }
```

Figure B.2: Exemplary worksheet on the Java topic *inheritance* as used in our study, consisting of text, an image, a video, a multiple choice quiz, and a programming exercise.

B.2 ARCHITECTURE

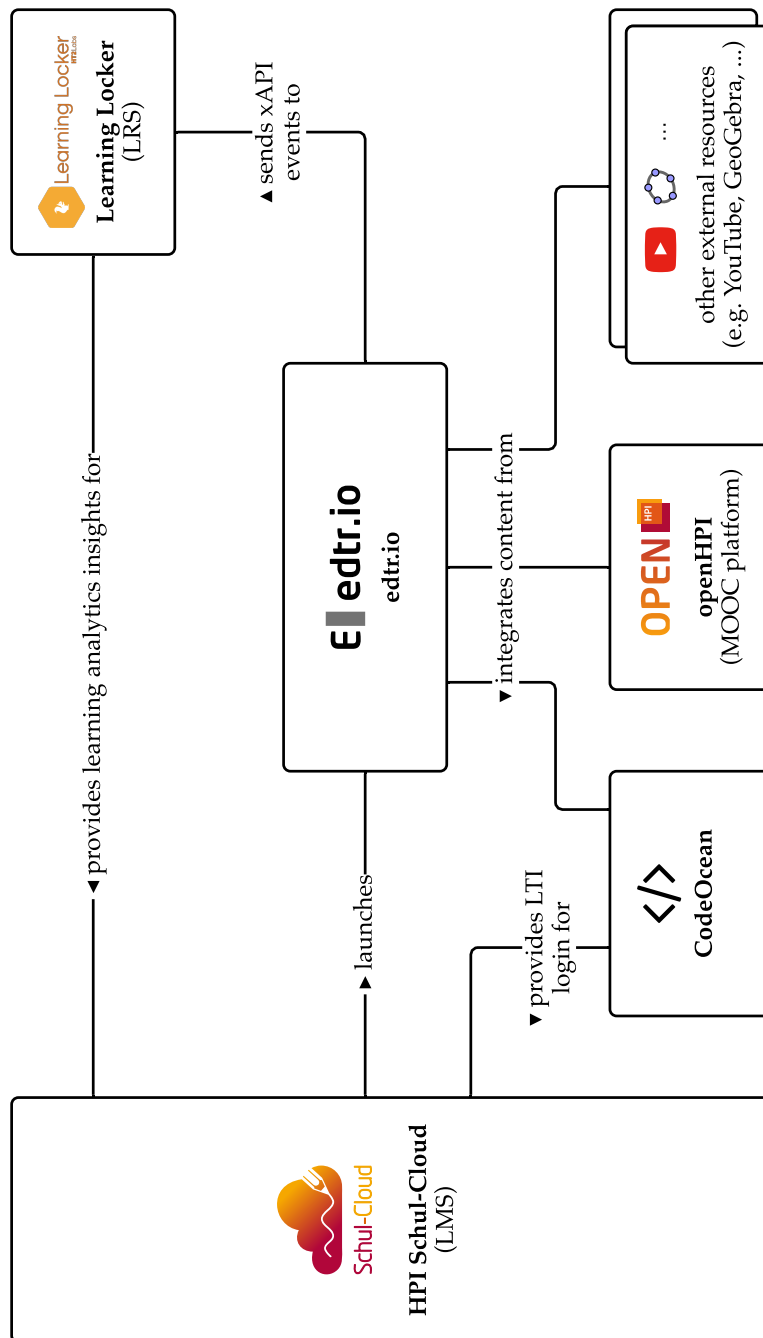


Figure B.3: System architecture of interactive worksheets with the integration of multimedia content and programming exercises from CodeOcean.

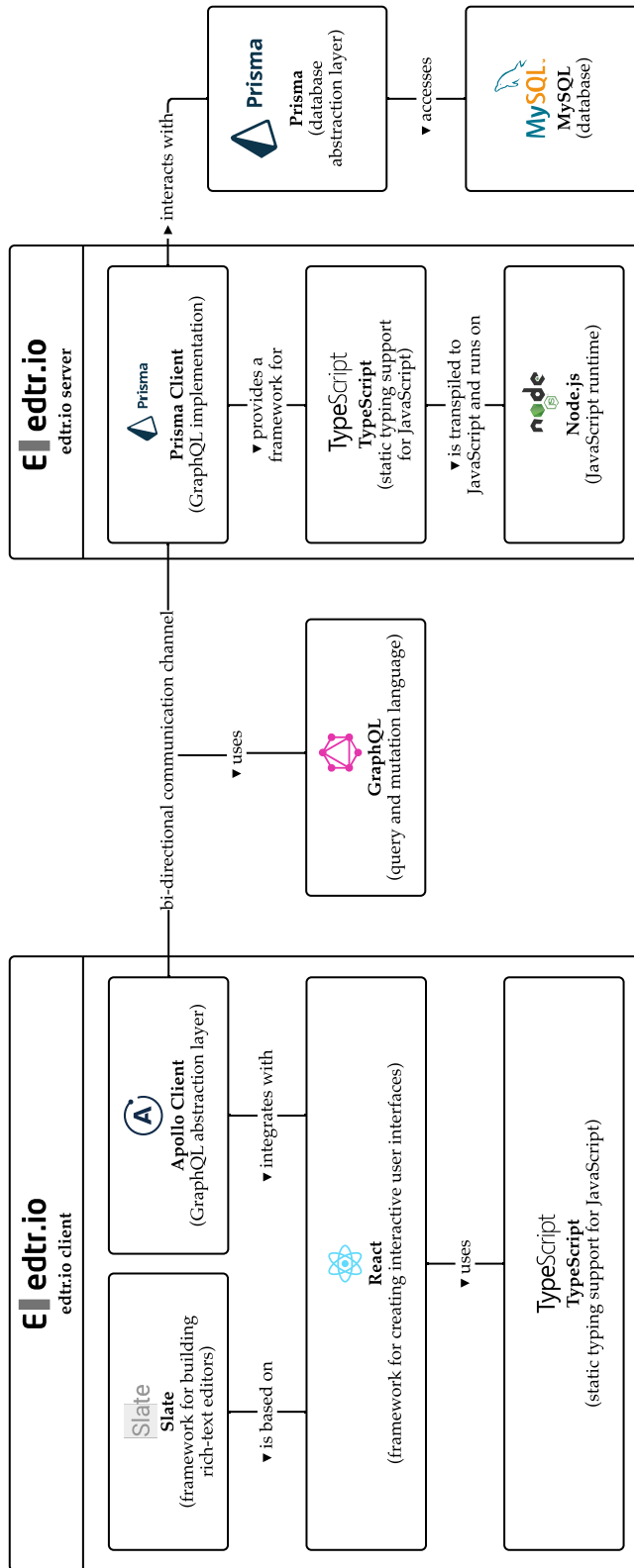


Figure B.4: System architecture of the worksheet editor edtr.io consisting of a backend server and a *React* web application.

B.3 IMPLEMENTATION DETAILS: EDTR.IO

```

_____ Slate Internal Document State _____
1 {
2   "object": "block",
3   "type": "multiple-choice",
4   "data": {},
5   "nodes": [
6     {
7       "object": "block",
8       "type": "multiple-choice-question",
9       "data": {},
10      "nodes": [
11        {
12          "object": "text",
13          "leaves": [
14            {
15              "object": "leaf",
16              "text": "Wofür wird Vererbung typischerweise
17                ↳ verwendet?",
18              "marks": []
19            }
20          ]
21        }
22      ],
23    },
24    {
25      "object": "block",
26      "type": "multiple-choice-answer",
27      "data": {
28        "id": "cjs66mei01mfq070010khauj2"
29      },
30      "nodes": [
31        {
32          "object": "text",
33          "leaves": [
34            {
35              "object": "leaf",
36              "text": "Zur Erstellung von Beziehungen zwischen
37                ↳ verschiedenen Klassen.",
38              "marks": []
39            }
40          ]
41        }
42      ],
43    },
44    {
45      "object": "block",
46      "type": "multiple-choice-answer",
47      "data": {
48        "id": "cjs66mqwn1mgy0700xcsv0u8t"
49      },
50      "nodes": [
51        {
52          "object": "text",

```

```

51     "leaves": [
52         {
53             "object": "leaf",
54             "text": "Zur Verringerung von Dopplungen von
                    ↪ Codezeilen",
55             "marks": []
56         }
57     ]
58 }
59 ]
60 },
61 {
62     "object": "block",
63     "type": "multiple-choice-answer",
64     "data": {
65         "id": "cjs66mvfx1mhi0700jhvvaeg7"
66     },
67     "nodes": [
68         {
69             "object": "text",
70             "leaves": [
71                 {
72                     "object": "leaf",
73                     "text": "Zur Aufteilung der Arbeit am Code auf
                            ↪ verschiedene Entwickler.",
74                     "marks": []
75                 }
76             ]
77         }
78     ]
79 },
80 {
81     "object": "block",
82     "type": "multiple-choice-answer",
83     "data": {
84         "id": "cjs66n0gr1mi20700nle5ia8y"
85     },
86     "nodes": [
87         {
88             "object": "text",
89             "leaves": [
90                 {
91                     "object": "leaf",
92                     "text": "Zum Ausdrücken einer \"is-a /
                            ↪ ist-ein\"-Beziehung.",
93                     "marks": []
94                 }
95             ]
96         }
97     ]
98 },
99 {
100     "object": "block",
101     "type": "multiple-choice-answer",
102     "data": {

```

```

103     "id": "cjs66orn71mkn0700yat4dpsi"
104   },
105   "nodes": [
106     {
107       "object": "text",
108       "leaves": [
109         {
110           "object": "leaf",
111           "text": "Zur Darstellung von verschiedenen
↪ Generationen bei Lebewesen, also einer
↪ Eltern-Kind-Beziehung",
112           "marks": []
113         }
114       ]
115     }
116   ]
117 },
118 {
119   "object": "block",
120   "type": "multiple-choice-answer",
121   "data": {
122     "id": "cjupqfnnt26rm07003mqr24c"
123   },
124   "nodes": [
125     {
126       "object": "text",
127       "leaves": [
128         {
129           "object": "leaf",
130           "text": "",
131           "marks": []
132         }
133       ]
134     }
135   ]
136 }
137 ]
138 }

```

Listing B.1: Extract of the internal document state in *Slate* regarding the multiple-choice plugin with one question and six answers as shown in [Figure 6.4](#). The information, whether an answer is correct or not, is neither stored in the local storage of a web browser nor send to students accessing the document.

```
_____ edtrio-server/src/database/datamodel.prisma _____  
1 type Document {  
2   id: ID! @unique  
3   value: Json!  
4   users: [User!]!  
5   createdAt: DateTime!  
6   updatedAt: DateTime!  
7   answers: [MultipleChoiceAnswer]  
8 }  
9  
10 type MultipleChoiceAnswer {  
11   id: ID! @unique  
12   isCorrect: Boolean!  
13   createdAt: DateTime!  
14   updatedAt: DateTime!  
15   submissions: [MultipleChoiceSubmission]  
16 }  
17  
18 type MultipleChoiceSubmission {  
19   id: ID! @unique  
20   createdAt: DateTime!  
21   updatedAt: DateTime!  
22   author: User!  
23   isChecked: Boolean!  
24   answer: MultipleChoiceAnswer!  
25 }
```

Listing B.2: Partial data model of edtr.io defining how the document is stored together with the answer options and submissions of the multiple-choice plugin. The question is stored as part of the document, which includes a list of all answer options. Each answer option might be chosen by multiple students.

B.4 IMPLEMENTATION DETAILS: CODEOCEAN

```

----- Study Group Working Time Query -----
1  -- Parameters passed from Ruby to identify an Exercise and a study group.
   ↳ An additional filter is used for live updates to transmit only
   ↳ information of a single user. Using the setting from
   ↳ StatisticsHelper::WORKING_TIME_DELTA_IN_SQL_INTERVAL, time deltas
   ↳ longer than five minutes are ignored when calculating the overall
   ↳ working time.
2
3  ==\set exercise_id 265
4  ==\set study_group_id 4
5  ==\set additional_filter 'AND user_id = 40260 AND user_type =
   ↳ \'ExternalUser\'
6  ==\set WORKING_TIME_DELTA_IN_SQL_INTERVAL '\00:05:00\'
7  ==
8
9  WITH working_time_between_submissions AS (
10     SELECT submissions.user_id,
11            submissions.user_type,
12            score,
13            created_at,
14            (created_at - lag(created_at) over
15             (PARTITION BY submissions.user_type, submissions.user_id,
16              ↳ exercise_id ORDER BY created_at)) AS working_time
16     FROM submissions
17     WHERE exercise_id = :exercise_id AND study_group_id = :study_group_id
18           ↳ :additional_filter),
18 working_time_with_deltas_ignored AS (
19     SELECT user_id,
20            user_type,
21            score,
22            sum(CASE WHEN score IS NOT NULL THEN 1 ELSE 0 END) over
23              (ORDER BY user_type, user_id, created_at ASC)
24              ↳ AS change_in_score,
24            created_at,
25            CASE WHEN working_time >= :WORKING_TIME_DELTA_IN_SQL_INTERVAL
26              ↳ THEN '0' ELSE working_time END AS working_time_filtered
26     FROM working_time_between_submissions
27 ),
28 working_times_with_score_expanded AS (
29     SELECT user_id,
30            user_type,
31            created_at,
32            working_time_filtered,
33            first_value(score) over
34              (PARTITION BY user_type, user_id, change_in_score ORDER BY
35               ↳ created_at ASC) AS corrected_score
35     FROM working_time_with_deltas_ignored
36 ),
37 working_times_with_duplicated_last_row_per_score AS (
38     SELECT *
39     FROM working_times_with_score_expanded
40     UNION ALL
41     -- Duplicate last row per user and score and make it unique by setting
42     ↳ another created_at timestamp. In addition, the working time is set
43     ↳ to zero in order to prevent getting a wrong time. This duplication
44     ↳ is needed, as we will shift the scores and working times by one
45     ↳ and need to ensure not to loose any information.
46     SELECT DISTINCT ON (user_type, user_id, corrected_score)

```

```

43     user_id,
44     user_type,
45     created_at + INTERVAL '1us',
46     '00:00:00' as working_time_filtered,
47     corrected_score
48 FROM working_times_with_score_expanded
49 ),
50 working_times_with_score_not_null_and_shifted AS (
51     SELECT user_id,
52            user_type,
53            coalesce(
54                lag(corrected_score) over
55                    (PARTITION BY user_type, user_id ORDER BY created_at ASC),
56                0) AS shifted_score,
57            created_at,
58            working_time_filtered
59 FROM working_times_with_duplicated_last_row_per_score
60 ),
61 working_times_to_be_sorted AS (
62     SELECT user_id,
63            user_type,
64            shifted_score                AS score,
65            MIN(created_at)              AS start_time,
66            SUM(working_time_filtered)   AS working_time_per_score,
67            SUM(SUM(working_time_filtered)) over
68                (PARTITION BY user_type, user_id) AS total_working_time
69 FROM working_times_with_score_not_null_and_shifted
70 GROUP BY user_id, user_type, score
71 ),
72 working_times_with_index AS (
73     SELECT (dense_rank() over
74         (ORDER BY total_working_time, user_type, user_id ASC) - 1) AS index,
75            user_id,
76            user_type,
77            score,
78            start_time,
79            working_time_per_score,
80            total_working_time
81 FROM working_times_to_be_sorted)
82 SELECT index,
83        user_id,
84        user_type,
85        name,
86        score,
87        start_time,
88        working_time_per_score,
89        total_working_time
90 FROM working_times_with_index
91 JOIN external_users ON user_type = 'ExternalUser'
92 AND user_id = external_users.id
93 UNION ALL
94 SELECT index,
95        user_id,
96        user_type,
97        name,
98        score,
99        start_time,
100       working_time_per_score,
101       total_working_time
102 FROM working_times_with_index

```

```

103 JOIN internal_users ON user_type = 'InternalUser'
104 AND user_id = internal_users.id
105 ORDER BY index, score ASC;

```

Listing B.3: PostgreSQL query to aggregate working times for a given exercise, study group and user. The given listing shows how to query the information using the *psql* command line interface and would usually include parameters provided from Ruby on Rails. The query is explained in [Section 6.6.2](#).

INDEX	USER_ID	USER_TYPE	NAME	SCORE	START_TIME	WORKING_TIME_ PER_SCORE	TOTAL_ WORKING_TIME
0	40260	ExternalUser	Sebastian	0	2019-03-12 13:37:19.22547	00:05:02.11990	00:05:02.11990
0	40260	ExternalUser	Sebastian	1	2019-03-12 13:42:21.34537	00:00:36.07017	00:05:38.19007

Table B.1: Result of the PostgreSQL query shown in [Listing B.3](#).

APPENDIX: EVALUATION

C.1 EXPLORATION: RESULTS OF THE INITIAL SURVEY



Figure C.1: Detailed results of our initial survey — Part I. Continued in [Figure C.2](#).



Figure C.2: Detailed results of our initial survey — Part II. Continued in [Figure C.3](#).



Figure C.3: Detailed results of our initial survey — Part III

C.2 STUDENTS: RESULTS OF THE MECUE

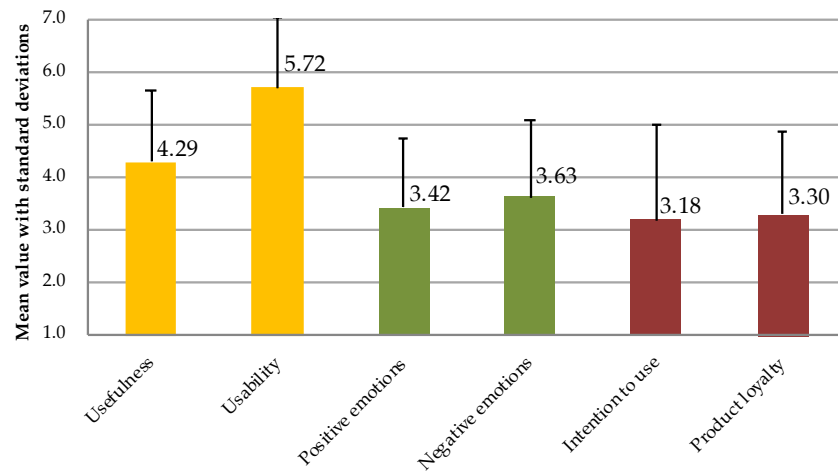


Figure C.4: Results of meCUE modules I (yellow), III (green) and IV (red) with the standard deviation for all students (N=11).

MODULE	SUBSCALE	MEDIAN	MEAN	STD. DEV.	MIN.	MAX.
Module I	Usefulness	5.00	4.29	1.38	2.00	6.00
	Usability	6.00	5.72	1.32	2.67	7.00
Module III	Positive Emotions	3.67	3.42	1.32	1.67	6.00
	Negative Emotions	4.00	3.63	1.48	1.17	6.00
Module IV	Intention to Use	3.18	4.29	1.84	1.00	6.00
	Product Loyalty	3.33	3.30	1.58	1.67	6.00
Module V	Overall evaluationn	2.00	1.90	2.50	-4.00	5.00

Table C.1: Detailed results of the meCUE for all students (N=11). Table C.2 and Table C.3 provide a more fine-granular view on each class.

MODULE	SUBSCALE	MEDIAN	MEAN	STD. DEV.	MIN.	MAX.
Module I	Usefulness	5.00	4.26	1.39	2.00	6.00
	Usability	6.67	6.00	1.51	2.67	7.00
Module III	Positive Emotions	3.50	3.17	1.10	1.67	4.50
	Negative Emotions	2.50	2.90	1.26	1.17	4.67
Module IV	Intention to Use	2.67	3.19	1.65	1.67	5.67
	Product Loyalty	3.33	3.24	1.54	1.67	6.00
Module V	Overall evaluationn	3.00	2.40	3.00	-4.00	5.00

Table C.2: Detailed results of the meCUE for students belonging to the first class (N=7).

MODULE	SUBSCALE	MEDIAN	MEAN	STD. DEV.	MIN.	MAX.
Module I	Usefulness	4.33	4.33	1.59	2.67	6.00
	Usability	5.50	5.25	0.88	4.00	6.00
Module III	Positive Emotions	3.83	3.88	1.71	1.83	6.00
	Negative Emotions	4.75	4.88	0.90	4.00	6.00
Module IV	Intention to Use	2.83	3.17	2.41	1.00	6.00
	Product Loyalty	2.83	3.42	1.89	2.00	6.00
Module V	Overall evaluationn	2.00	1.40	2.10	-3.00	4.00

Table C.3: Detailed results of the meCUE for students belonging to the second class (N=4).

Student	1	2	3	4	5	6	7	8	9
openPE: Vorteile	Beispielsweise: Rechte, Support und den Ressourcen (in Systemgebunden)	individuelle Arbeiten	Von überall erreichbar, klare Strukturierung	Man kann es Zuhause und am LCH nutzen.	Naja, dadurch wird es uns bequemer.	einliche Benutzung durch Code-Ocean	Aus meiner Sicht übersichtlicher	Übersichtliche der Programmierumgebung möglich Quiz zur Rekapitulation des gelernten Stoffes	überall verfügbar (nicht nur im Unterricht, sondern auch im Internat oder Zuhause), modern
openPE: Nachteile	Beispielsweise: Probleme von Plägern der Kommentarfunktion etc.	s. o.	Keine	Keine	Mein Laptop kriegt's halt wie gesagt nicht hin.	zu einheitliche, standardisierte Aufgaben		nur Lernendes -> bei kleinen Gruppen, das macht es nicht bis zur entsprechenden Stelle wieder angeschaut werden; keine geschickte Wiederholung möglich	mir persönlich ist es sehr schwer gefallen, mit dem Online-Kurs Inhalte zu erlernen. Die Aufgaben sind teilweise sehr theoretisch und schwer zu verstehen. Das unsere Rechenfolge anders war und wir deswegen schon Schwierigkeiten hatten, vor allem bei der Programmierung. Ich habe versucht, sie schon behandelt, hat mich das Instruier und mir das Lernen erschwert.
Schul-Cloud: Vorteile	Lehrer kann Inhalte nicht ändern, sondern nur mit optimieren Cloud			Man kann von überall die Blätter aufrufen.	Man kommt alles drum.	ist eventuell bereits bekannter		Materialien immer dabei	Dabei digital hochladen und sichern, bei ausgefallenen Stunden können wir so einfach an unsere Arbeitsblätter kommen. In anderen Fällen kann man sie auch herunterladen, wenn man sie im Unterricht abzugeben.
Schul-Cloud: Nachteile	Benutzerfreundlichkeit und einige Funktionen sind nicht intuitiv (z.B. in Browser etc.)			Niemand anderes benutzt es (Lehrer).	Keine	Keine nur für das Speichern von Daten verwendet werden (Cloud) Aus meiner Sicht unübersichtlicher wird		teilweise Arbeitsblätter dort nicht richtig aufrufbar (das es anders als in anderen Kursen geteilt wird)	technische Probleme! In anderen Fällen hat sich unser Lehrer nicht geäußert, die Schulcloud ist nicht intuitiv, aber die Handlung von technischen Problemen, wo wir bei Hausaufgaben keine Daten übertragen können, ist sehr schmerzhaft aufzugeben.
digitale, interaktive Arbeitsblätter: Vorteile		Geben Überblick über den jeweiligen Stundeninhalt		Man hat alles am gleichen Ort.		flexible und einfache Einbettung der Aufgaben		alles an einem Platz	unpersönliche, laud und gute Erklärungen (an anschaulichen Beispielen)
digitale, interaktive Arbeitsblätter: Nachteile		Ist nutzbar, sie kann weil ich persönlich lieber die Aufgaben bearbeite (gut genauso für andere Arbeitsblätter)		Keine		Aufgaben werden teilweise nicht vollständig angezeigt, das Quiz nicht abgeändert		es muss immer zwischen Arbeitsblatt und Notizblatt keine Notizen gemacht werden können (es Vorteil statisches Arbeitsblatt)	Nutzen machen. Ohne mich bei den Videos Notizen zu machen hätte ich die Inhalte nicht langfristig behalten oder erst verändern können.
statische Arbeitsblätter: Vorteile		nutzbar während der Klausur		Man kann sie auf seinem Stick speichern und nach Hause mitnehmen.		Können überall, ohne Probleme verwendet werden		Änderungen durch den Lehrenden bedürfen jeweils neuer Versionen (z.B. bei Änderungen der Aufgaben zu Beginn der darauffolgenden Stunde)	Mir ist es viel leichter gefallen, mit dem Lehrbuch zu lernen. Die Texte waren informativ und sehr verständlich.
statische Arbeitsblätter: Nachteile				Keine		statisch ID kann nicht flexibel in die Arbeit eingebettet werden		alles verständlich erklärt (teilweise mehr Hintergrund)	
Lehrbuch: Vorteile				Man hat alle Themen an einem Ort.		absolute fachliche Richtigkeit und Fülle an Informationen		Ausdrücklich mit Beispielen und Vertiefungen	
Lehrbuch: Nachteile				Keine		keine oder veraltete Praxis		weniger anwendungsbezogen als beispielsweise die openPE Wissen/Verständnis zu überprüfen als mögliches Gestaltung sind.	Es ist schwieriger, zu sehen, da sie nicht so informativ sind wie die openPE

Womn siehst du die Vor- und Nachteile der einzelnen Tools? Denke hierbei gerne an deine individuelle Lernsituation und wie du die Tools jeweils verwendest.

Figure C.6: Detailed results of the text survey of the first class — Part II. Continued in Figure C.7.

C.4 TEACHERS: RESULTS OF THE UEQ

Item	Mean	Variance	Std. Dev.	No.	Left	Right	Scale
1	↑ 2.3	1.1	1.1	12	annoying	enjoyable	Attractiveness
2	↑ 2.4	0.4	0.7	12	not understandable	understandable	Perspicuity
3	↔ 0.6	3.7	1.9	12	creative	dull	Novelty
4	↔ 0.7	2.2	1.5	12	easy to learn	difficult to learn	Perspicuity
5	↑ 1.8	1.8	1.4	12	valuable	inferior	Stimulation
6	↑ 1.7	1.7	1.3	12	boring	exciting	Stimulation
7	↑ 1.6	2.1	1.4	12	not interesting	interesting	Stimulation
8	↑ 1.5	2.5	1.6	11	unpredictable	predictable	Dependability
9	↑ 0.9	2.1	1.4	12	fast	slow	Efficiency
10	↑ 0.8	2.3	1.5	12	inventive	conventional	Novelty
11	↑ 2.4	0.8	0.9	12	obstructive	supportive	Dependability
12	↑ 2.0	1.4	1.2	11	good	bad	Attractiveness
13	↑ 1.0	2.0	1.4	12	complicated	easy	Perspicuity
14	↑ 1.3	2.9	1.7	12	unlikable	pleasing	Attractiveness
15	↑ 1.3	2.1	1.4	12	usual	leading edge	Novelty
16	↑ 1.8	1.1	1.1	12	unpleasant	pleasant	Attractiveness
17	↑ 1.0	1.6	1.3	12	secure	not secure	Dependability
18	↑ 1.6	2.1	1.4	11	motivating	demotivating	Stimulation
19	↑ 1.3	2.6	1.6	12	meets expectations	does not meet expectations	Dependability
20	↑ 1.8	0.9	0.9	12	inefficient	efficient	Efficiency
21	↑ 1.6	1.0	1.0	12	clear	confusing	Perspicuity
22	↑ 1.5	1.5	1.2	11	impractical	practical	Efficiency
23	↑ 1.8	2.0	1.4	12	organized	cluttered	Efficiency
24	↑ 1.3	2.1	1.4	12	attractive	unattractive	Attractiveness
25	↑ 1.7	2.4	1.6	12	friendly	unfriendly	Attractiveness
26	↑ 1.5	2.6	1.6	12	conservative	innovative	Novelty

Figure C.10: Detailed results of the UEQ indicating the mean values of all 26 items. The colors indicate the corresponding bar in Figure C.11.

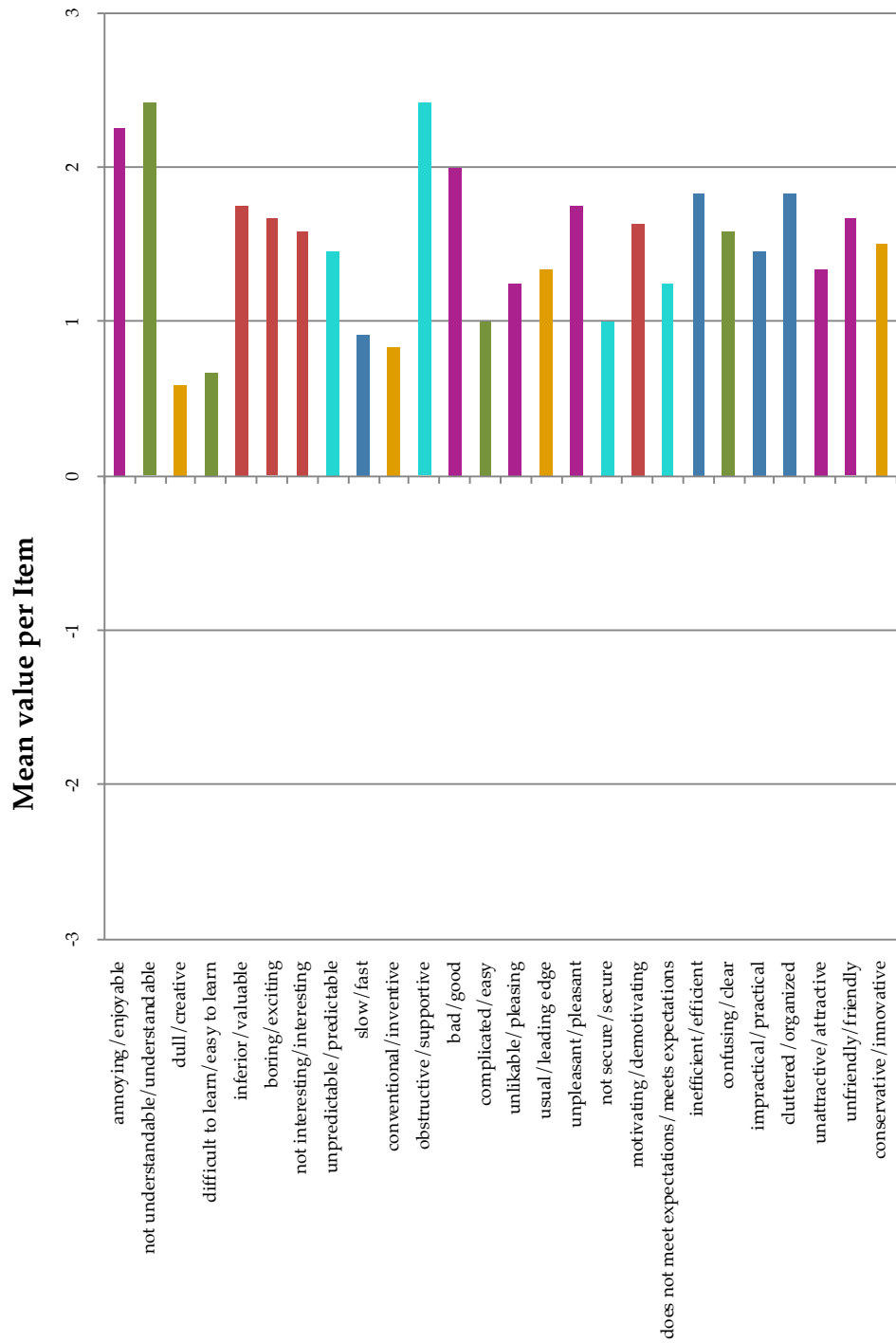


Figure C.11: Mean values of the items shown in Figure C.10 mapped to bars.

BIBLIOGRAPHY

- [1] "CSchunck". *Arbeitsblatt Rekursion - tutoring.de*. Released into the public domain using CCo. Screenshot taken by author. Oct. 18, 2016. URL: <https://www.tutoring.de/w/36c7ef70> (visited on 04/13/2019) (cit. on pp. 33, 101).
- [2] Paul Arndt. "Supporting Internal Differentiation and Cooperative Learning with the HPI Schul-Cloud". Potsdam, Germany: Hasso Plattner Institute, University of Potsdam, May 2019 (cit. on pp. 36, 39, 53).
- [3] Alberto Bartoli, Giorgio Davanzo, Andrea De Lorenzo, Marco Mauri, Eric Medvet, and Enrico Sorio. "Automatic Generation of Regular Expressions from Examples with Genetic Programming". In: *Proceedings of the Fourteenth International Conference on Genetic and Evolutionary Computation Conference Companion - GECCO Companion '12*. The Fourteenth International Conference. Philadelphia, Pennsylvania, USA: ACM Press, 2012, p. 1477. ISBN: 978-1-4503-1178-6. DOI: [10.1145/2330784.2331000](https://doi.org/10.1145/2330784.2331000) (cit. on p. 38).
- [4] *Basic Law for the Federal Republic of Germany*. May 23, 1949. URL: https://www.gesetze-im-internet.de/englisch_gg/ (cit. on p. 1).
- [5] Gary Beauchamp and Steve Kennewell. "Interactivity in the Classroom and Its Impact on Learning". In: *Computers & Education* 54.3 (Apr. 2010), pp. 759–766. ISSN: 03601315. DOI: [10.1016/j.compedu.2009.09.033](https://doi.org/10.1016/j.compedu.2009.09.033) (cit. on pp. 13, 14).
- [6] Matthew Berland, Ryan S. Baker, and Paulo Blikstein. "Educational Data Mining and Learning Analytics: Applications to Constructionist Research". In: *Technology, Knowledge and Learning* 19.1-2 (July 2014), pp. 205–220. ISSN: 2211-1662, 2211-1670. DOI: [10.1007/s10758-014-9223-7](https://doi.org/10.1007/s10758-014-9223-7) (cit. on p. 16).
- [7] Paul Blayney and Mark Freeman. "Automated Formative Feedback and Summative Assessment Using Individualised Spreadsheet Assignments". In: *Australasian Journal of Educational Technology* 20.2 (Aug. 9, 2004), pp. 209–231. ISSN: 1449-5554, 1449-3098. DOI: [10.14742/ajet.1360](https://doi.org/10.14742/ajet.1360) (cit. on p. 14).
- [8] Paul Blayney and Mark Freeman. "Individualised Interactive Formative Assessments to Promote Independent Learning". In: *Journal of Accounting Education* 26.3 (Sept. 2008), pp. 155–165. ISSN: 07485751. DOI: [10.1016/j.jaccedu.2008.01.001](https://doi.org/10.1016/j.jaccedu.2008.01.001) (cit. on p. 14).

- [9] Paulo Blikstein. "Using Learning Analytics to Assess Students' Behavior in Open-Ended Programming Tasks". In: *Proceedings of the 1st International Conference on Learning Analytics and Knowledge - LAK '11*. The 1st International Conference. Banff, Alberta, Canada: ACM Press, 2011, p. 110. ISBN: 978-1-4503-0944-8. DOI: [10.1145/2090116.2090132](https://doi.org/10.1145/2090116.2090132) (cit. on pp. 16, 43).
- [10] Michael Caulfield, Amy Collier, and Sherif Halawa. *Rethinking Online Community in MOOCs Used for Blended Learning*. Oct. 6, 2013. URL: <https://er.educause.edu/articles/2013/10/rethinking-online-community-in-moocs-used-for-blended-learning> (visited on 02/08/2019) (cit. on p. 15).
- [11] Der Landesbeauftragte für den Datenschutz und die Informationsfreiheit Rheinland-Pfalz. *Anforderungen für den schulischen Einsatz von Google-Classroom*. Jan. 10, 2017. URL: <https://www.datenschutz.rlp.de/fileadmin/lfdi/Dokumente/Orientierungshilfen/anforderungen-google-classroom.pdf> (visited on 04/23/2019) (cit. on p. 24).
- [12] Firas Dib. *Testing a Regular Expression with Regex101*. Screenshot taken by author. URL: <https://regex101.com/> (visited on 04/17/2019) (cit. on p. 102).
- [13] Christoph Drösser and Uwe Jean Heuser. "Moocs: Harvard für alle Welt". In: *Die Zeit* (Mar. 14, 2013). ISSN: 0044-2070. URL: <https://www.zeit.de/2013/12/MOOC-Onlinekurse-Universitaeten> (visited on 04/05/2019) (cit. on p. 7).
- [14] *European General Data Protection Regulation*. May 24, 2018. URL: <https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=celex%3A32016R0679> (cit. on pp. XV, 40).
- [15] *Exemplary Screenshot of Codeboard*. Source code based on an assignment from Heiduck [24]. Screenshot taken by author. URL: <https://codeboard.io> (visited on 04/18/2019) (cit. on p. 18).
- [16] *Exemplary Screenshot of Repl.it*. Source code based on an assignment from Heiduck [24]. Screenshot taken by author. URL: <https://codeboard.io> (visited on 04/18/2019) (cit. on p. 103).
- [17] Ian Fette and Alexey Melnikov. *The WebSocket Protocol*. RFC6455. RFC Editor, Dec. 2011, RFC6455. DOI: [10.17487/rfc6455](https://doi.org/10.17487/rfc6455) (cit. on pp. 53, 68).
- [18] *German Federal Data Protection Act*. May 25, 2018. URL: https://www.gesetze-im-internet.de/englisch_bdsgr/ (cit. on pp. XV, 40).
- [19] Dominik Glandorf. "Data-Protection-Compliant Deep Integration of Third-Party Educational Resources into HPI Schul-Cloud". Potsdam, Germany: Hasso Plattner Institute, University of Potsdam, July 2, 2018 (cit. on p. 66).

- [20] Rebecca Griffiths, Matthew Chingos, Christine Mulhern, and Richard Spies. *Interactive Online Learning on Campus: Testing MOOCs and Other Platforms in Hybrid Formats in the University System of Maryland*. New York: Ithaka S+R, May 10, 2014. DOI: [10.18665/sr.22522](https://doi.org/10.18665/sr.22522) (cit. on p. 15).
- [21] Christiane Hagedorn and Christoph Meinel. “Exploring the Potential of Game-Based Learning in Massive Open Online Courses”. In: *2017 IEEE 17th International Conference on Advanced Learning Technologies (ICALT)*. Timisoara, Romania: IEEE, July 2017, pp. 542–544. ISBN: 978-1-5386-3870-5. DOI: [10.1109/ICALT.2017.119](https://doi.org/10.1109/ICALT.2017.119) (cit. on p. 13).
- [22] Stian Håklev. *Exemplary Screenshot of the FROG Editor*. Oct. 27, 2017. URL: <https://github.com/chili-epfl/FROG/blob/3fb7d86df12e296bd16dfb89a2a61e8c18ca8aab/docs/frog-editor.png> (visited on 04/17/2019) (cit. on p. 37).
- [23] Stian Håklev, Louis Faucon, Thanasis Hadzilacos, and Pierre Dillenbourg. “FROG: Rapid Prototyping of Collaborative Learning Scenarios”. In: *EC-TEL Practitioner Proceedings 2017: 12th European Conference On Technology Enhanced Learning*. Tallin, Estonia, Sept. 12, 2017. URL: <https://infoscience.epfl.ch/record/230014> (cit. on p. 36).
- [24] Ulrike Heiduck. *Programming Exercise Assessing the Understanding of Arrays*. Mar. 28, 2019 (cit. on p. 130).
- [25] Peter Hubwieser, Michal Armoni, and Michail N. Giannakos. “How to Implement Rigorous Computer Science Education in K-12 Schools? Some Answers and Many Questions”. In: *ACM Transactions on Computing Education* 15.2 (Apr. 27, 2015), pp. 1–12. ISSN: 19466226. DOI: [10.1145/2729983](https://doi.org/10.1145/2729983) (cit. on p. 15).
- [26] Ville Isomöttönen, Antti-Jussi Lakanen, and Vesa Lappalainen. “K-12 Game Programming Course Concept Using Textual Programming”. In: *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education - SIGCSE '11*. The 42nd ACM Technical Symposium. Dallas, TX, USA: ACM Press, 2011, p. 459. ISBN: 978-1-4503-0500-6. DOI: [10.1145/1953163.1953296](https://doi.org/10.1145/1953163.1953296) (cit. on p. 14).
- [27] Maria Joseph Israel. “Effectiveness of Integrating MOOCs in Traditional Classrooms for Undergraduate Students”. In: *The International Review of Research in Open and Distributed Learning* 16.5 (Sept. 29, 2015), pp. 102–118. ISSN: 1492-3831. DOI: [10.19173/irrodl.v16i5.2222](https://doi.org/10.19173/irrodl.v16i5.2222) (cit. on p. 15).
- [28] Michael Janke. “Digital Classroom: Digitally Supported Group Work in a School Context - Connecting Digital Real-Time Collaboration and Analog Communication”. Potsdam, Germany:

- Hasso Plattner Institute, University of Potsdam, Dec. 7, 2018 (cit. on p. 35).
- [29] Michael Janke. *Exemplary Screenshot of the Digital Classroom*. Screenshot taken by author. Dec. 11, 2018. URL: <https://www.tele-task.de/lecture/video/7231/> (visited on 04/17/2019) (cit. on p. 101).
- [30] Michael Kerres. *Multimediale Und Telemediale Lernumgebungen*. 2nd ed. OCLC: 1091456005. Walter de Gruyter, 2009. 412 pp. ISBN: 978-3-486-59381-5 (cit. on p. 24).
- [31] Jonas Keutel. "Towards Shared Learning Contents - and How to Make Teachers Want to Contribute". Potsdam, Germany: Hasso Plattner Institute, University of Potsdam, Nov. 15, 2018 (cit. on p. 42).
- [32] Juliane Kleinknecht. "Das Lern-Cockpit: Nutzerorientierte Darstellung von Lerndaten in der Schul-Cloud". Potsdam, Germany: Hasso Plattner Institute, University of Potsdam, July 2, 2018 (cit. on p. 51).
- [33] Tobias Kollmann and Holger Schmidt. *Deutschland 4.0: wie die Digitale Transformation gelingt*. OCLC: ocn932096511. Wiesbaden: Gabler, 2016. 186 pp. ISBN: 978-3-658-11981-2. URL: <https://link.springer.com/book/10.1007/978-3-658-13145-6> (visited on 04/19/2018) (cit. on p. 21).
- [34] Bettina Laugwitz, Theo Held, and Martin Schrepp. "Construction and Evaluation of a User Experience Questionnaire". In: *HCI and Usability for Education and Work*. Ed. by Andreas Holzinger. Vol. 5298. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 63–76. ISBN: 978-3-540-89349-3 978-3-540-89350-9. DOI: [10.1007/978-3-540-89350-9_6](https://doi.org/10.1007/978-3-540-89350-9_6) (cit. on pp. XVI, 85, 86).
- [35] Diandra L. Leslie-Pelecky. "Interactive Worksheets in Large Introductory Physics Courses". In: *The Physics Teacher* 38.3 (Mar. 2000), pp. 165–167. ISSN: 0031-921X. DOI: [10.1119/1.880485](https://doi.org/10.1119/1.880485) (cit. on p. 14).
- [36] Phil Long and George Siemens. "Penetrating the Fog: Analytics in Learning and Education". In: *EDUCAUSE review* 46.5 (2011), pp. 30–40. URL: <https://er.educause.edu/~media/files/article-downloads/erm1151.pdf> (cit. on p. 16).
- [37] Christian Matt, Thomas Hess, and Alexander Benlian. "Digital Transformation Strategies". In: *Business & Information Systems Engineering* 57.5 (Oct. 2015), pp. 339–343. ISSN: 2363-7005, 1867-0202. DOI: [10.1007/s12599-015-0401-5](https://doi.org/10.1007/s12599-015-0401-5) (cit. on p. 1).

- [38] Christoph Matthies, Ralf Teusner, and Guenter Hesse. "Beyond Surveys: Analyzing Software Development Artifacts to Assess Teaching Efforts". In: *2018 IEEE Frontiers in Education Conference (FIE)*. San Jose, CA, USA: IEEE, Oct. 2018, pp. 1–9. ISBN: 978-1-5386-1174-6. DOI: [10.1109/FIE.2018.8659205](https://doi.org/10.1109/FIE.2018.8659205) (cit. on p. 15).
- [39] Christoph Meinel, Jan Renz, Matthias Luderich, Vivien Maly-ska, Konstantin Kaiser, and Arne Oberländer. *Die HPI Schul-Cloud: Roll-Out einer Cloud-Architektur für Schulen in Deutschland*. Technische Berichte des Hasso-Plattner-Instituts für Digital Engineering an der Universität Potsdam 125. Potsdam: Universitätsverlag Potsdam, 2019. 57 pp. ISBN: 978-3-86956-453-1. URL: https://hpi.de/fileadmin/user_upload/hpi/dokumente/publikationen/technische_berichte/tbhpi125.pdf (cit. on p. 11).
- [40] Christoph Meinel, Volker Schillings, and Vanessa Walser. "Overcoming Technical Frustrations in Distance Education: Tele-TASK". In: *Proceedings of the IADIS International Conference on E-Society*. E-Society. Lisboa, Portugal, 2003, p. 8. URL: https://hpi.de/fileadmin/user_upload/fachgebiete/meinel/papers/Web-University/2003_Meinel_e-Society.pdf (cit. on p. 56).
- [41] Christoph Meinel and Christian Willems. *openHPI: The MOOC Offer at Hasso Plattner Institute*. In collab. with Hasso-Plattner-Institut für Softwaresystemtechnik. Technische Berichte Des Hasso-Plattner-Instituts Für Softwaresystemtechnik an Der Universität Potsdam 80. OCLC: 885028328. Potsdam: Universitätsverlag Potsdam, 2013. 21 pp. ISBN: 978-3-86956-264-3. URL: <https://publishup.uni-potsdam.de/opus4-ubp/frontdoor/deliver/index/docId/6548/file/tbhpi80.pdf> (cit. on p. 8).
- [42] Zahira Merchant, Ernest T. Goetz, Lauren Cifuentes, Wendy Keeney-Kennicutt, and Trina J. Davis. "Effectiveness of Virtual Reality-Based Instruction on Students' Learning Outcomes in K-12 and Higher Education: A Meta-Analysis". In: *Computers & Education* 70 (Jan. 2014), pp. 29–40. ISSN: 03601315. DOI: [10.1016/j.compedu.2013.07.033](https://doi.org/10.1016/j.compedu.2013.07.033) (cit. on p. 13).
- [43] Michael Minge, Manfred Thüring, Ingmar Wagner, and Carina V. Kuhr. "The meCUE Questionnaire: A Modular Tool for Measuring User Experience". In: *Advances in Ergonomics Modeling, Usability & Special Populations*. Ed. by Marcelo Soares, Christianne Falcão, and Tareq Z. Ahram. Vol. 486. Cham: Springer International Publishing, 2017, pp. 115–128. ISBN: 978-3-319-41684-7 978-3-319-41685-4. DOI: [10.1007/978-3-319-41685-4_11](https://doi.org/10.1007/978-3-319-41685-4_11) (cit. on pp. XVI, 79).
- [44] Mike Monaco. "Regular Expressions 101: Regex101.Com". In: *Technical Services Quarterly* 35.3 (July 3, 2018), pp. 305–306. ISSN:

- 0731-7131, 1555-3337. DOI: [10.1080/07317131.2018.1456868](https://doi.org/10.1080/07317131.2018.1456868) (cit. on p. 38).
- [45] Carlos Monroy, Virginia Snodgrass Rangel, and Reid Whitaker. "A Strategy for Incorporating Learning Analytics into the Design and Evaluation of a K-12 Science Curriculum". In: *Journal of Learning Analytics* 1.2 (2014), pp. 94–125. ISSN: 19297750. DOI: [10.18608/jla.2014.12.6](https://doi.org/10.18608/jla.2014.12.6) (cit. on p. 15).
- [46] Nachiappan Nagappan, Laurie Williams, Miriam Ferzli, Eric Wiebe, Kai Yang, Carol Miller, and Suzanne Balik. "Improving the CS1 Experience with Pair Programming". In: *ACM SIGCSE Bulletin* 35.1 (Jan. 11, 2003), p. 359. ISSN: 00978418. DOI: [10.1145/792548.612006](https://doi.org/10.1145/792548.612006) (cit. on p. 17).
- [47] Aisha Othman, Ahmed Impes, and Crinela Pislaru. "Online Interactive Module for Teaching a Computer Programming Course". In: *Proceedings of the 12th European Conference on E-Learning ECEL 2013*. The 12th European Conference on E-Learning ECEL 2013. 2013. ISBN: 978-1-909507-82-1. URL: <http://eprints.hud.ac.uk/id/eprint/19628> (cit. on p. 13).
- [48] Timo Partala and Aleksi Kallinen. "Understanding the Most Satisfying and Unsatisfying User Experiences: Emotions, Psychological Needs, and Context". In: *Interacting with Computers* 24.1 (Jan. 2012), pp. 25–34. ISSN: 09535438. DOI: [10.1016/j.intcom.2011.10.001](https://doi.org/10.1016/j.intcom.2011.10.001) (cit. on pp. 81, 83).
- [49] Morten Flate Paulsen. "Online Education Systems: Discussion and Definition of Terms". In: *NKI distance education* 202 (July 2002), p. 8. URL: <http://www.porto.ucp.pt/open/curso/modulos/doc/Definition%20of%20Terms.pdf> (cit. on p. 24).
- [50] Frederick F. Reichheld. "The One Number You Need to Grow". In: *harvard business review* (2003), p. 12. URL: <https://hbr.org/2003/12/the-one-number-you-need-to-grow> (cit. on pp. XVI, 78).
- [51] Jan Renz and Christoph Meinel. "Can Pseudonymized xAPI-Tracking Solve Data Privacy Issues in German Schools?" In: *SAILA-ECTEL*. EC-TEL Practitioner Proceedings 2018: 13th European Conference On Technology Enhanced Learning. Vol. 2193. Workshop Paper. Leeds, UK, Sept. 3, 2018, p. 3. URL: https://www.researchgate.net/publication/332319620_Can_pseudonymized_xAPI-Tracking_solve_data_privacy_issues_in_german_schools (cit. on pp. 40, 45, 134).
- [52] Jan Renz and Christoph Meinel. *Pseudonymization Concept of the HPI Schul-Cloud*. Image extracted from "Can Pseudonymized xAPI-Tracking Solve Data Privacy Issues in German Schools?" [51] (cit. on p. 45).

- [53] E. Hammer-Lahav (Ed.) “The OAuth 1.0 Protocol”. In: Internet Request for Comments 5849 (Apr. 2010). Obsoleted by RFC 6749, pp. 1–38. ISSN: 2070-1721. DOI: [10.17487/RFC5849](https://doi.org/10.17487/RFC5849) (cit. on p. 63).
- [54] A. Barth. “HTTP State Management Mechanism”. In: Internet Request for Comments 6265 (Apr. 2011), pp. 1–37. ISSN: 2070-1721. DOI: [10.17487/RFC6265](https://doi.org/10.17487/RFC6265) (cit. on p. 73).
- [55] D. Hardt (Ed.) “The OAuth 2.0 Authorization Framework”. In: Internet Request for Comments 6749 (Oct. 2012). Updated by RFC 8252, pp. 1–76. ISSN: 2070-1721. DOI: [10.17487/RFC6749](https://doi.org/10.17487/RFC6749) (cit. on p. 63).
- [56] Tobias Rohloff, Dominic Sauer, and Christoph Meinel. “On the Acceptance and Usefulness of Personalized Learning Objectives in MOOCs”. In: *Proceedings of the Sixth Annual ACM Conference on Learning at Scale - L@S '19*. The Sixth Annual ACM Conference on Learning At Scale. Chicago, IL, USA: ACM Press, 2019, p. 10 (cit. on p. 16).
- [57] Gerhard Röhner. *Exemplary Screenshot of the Java-Editor*. Apr. 24, 2014. URL: <http://javaeditor.org/lib/exe/detail.php?id=start&media=en:editoren.png> (visited on 04/16/2019) (cit. on p. 29).
- [58] Sebastian Serth, Ralf Teusner, Jan Renz, and Matthias Uflacker. “Evaluating Digital Worksheets with Interactive Programming Exercises for K-12 Education”. In: *2019 IEEE Frontiers in Education Conference (FIE)*. Manuscript Submitted for Publication. Cincinnati, OH, USA: IEEE, 2019 (cit. on p. XIV).
- [59] Jürgen Soose. *Arbeitsblatt selbst-definierte Klasse*. Sept. 30, 2010. URL: <http://jsoose.de> (cit. on p. 26).
- [60] Thomas Staubitz, Hauke Klement, Ralf Teusner, Jan Renz, and Christoph Meinel. “CodeOcean - A Versatile Platform for Practical Programming Exercises in Online Environments”. In: *2016 IEEE Global Engineering Education Conference (EDUCON)*. Abu Dhabi: IEEE, Apr. 2016, pp. 314–323. ISBN: 978-1-4673-8633-3. DOI: [10.1109/EDUCON.2016.7474573](https://doi.org/10.1109/EDUCON.2016.7474573) (cit. on pp. 8, 10).
- [61] Thomas Staubitz, Ralf Teusner, and Christoph Meinel. “Towards a Repository for Open Auto-Gradable Programming Exercises”. In: *2017 IEEE 6th International Conference on Teaching, Assessment, and Learning for Engineering (TALE)*. Hong Kong: IEEE, Dec. 2017, pp. 66–73. ISBN: 978-1-5386-0900-2. DOI: [10.1109/TALE.2017.8252306](https://doi.org/10.1109/TALE.2017.8252306) (cit. on p. 38).
- [62] Thomas Staubitz, Ralf Teusner, and Christoph Meinel. “MOOCs in Secondary Education - Experiments and Observations from German Classrooms”. In: *2019 IEEE Global Engineering Education Conference (EDUCON)*. Dubai, UAE: IEEE, 2019, p. 10 (cit. on p. 3).

- [63] Stephen Vickers (IMS Global). *IMS Global Learning Tools Interoperability® Outcomes Management*. Version 1.0 Final Release. IMS Global Learning Consortium Inc. Jan. 5, 2015. URL: <https://www.imsglobal.org/specs/ltiomv1p0/specification> (cit. on p. 66).
- [64] Ralf Teusner, Thomas Hille, and Thomas Staubitz. "Effects of Automated Interventions in Programming Assignments: Evidence from a Field Experiment". In: *Proceedings of the Fifth Annual ACM Conference on Learning at Scale - L@S '18*. The Fifth Annual ACM Conference. London, United Kingdom: ACM Press, 2018, pp. 1–10. ISBN: 978-1-4503-5886-6. DOI: [10.1145/3231644.3231650](https://doi.org/10.1145/3231644.3231650) (cit. on pp. 10, 39).
- [65] Christian Ullenboom. *Java ist auch eine Insel: Einführung, Ausbildung, Praxis*. 13., aktualisierte und überarbeitete Auflage. Rheinwerk Computing. OCLC: 985977047. Bonn: Rheinwerk Verlag, 2018. 1363 pp. ISBN: 978-3-8362-5869-2. URL: <http://openbook.rheinwerk-verlag.de/javainsel/> (cit. on p. 16).
- [66] University of Southern California, Ann Majchrzak, M. Lynne Markus, Bentley University, Jonathan Wareham, and ESADE – Ramon Llull University. "Designing for Digital Transformation: Lessons for Information Systems Research from the Study of ICT and Societal Challenges". In: *MIS Quarterly* 40.2 (Feb. 2, 2016), pp. 267–277. ISSN: 02767783, 21629730. DOI: [10.25300/MISQ/2016/40:2.03](https://doi.org/10.25300/MISQ/2016/40:2.03) (cit. on p. 1).
- [67] Roumen Vesselinov and John Grego. *Duolingo Effectiveness Study - Final Report*. Dec. 2012. URL: http://static.duolingo.com/s3/DuolingoReport_Final.pdf (visited on 04/30/2019) (cit. on p. 86).
- [68] Ben Williamson. "Digital Education Governance: Data Visualization, Predictive Analytics, and 'Real-Time' Policy Instruments". In: *Journal of Education Policy* 31.2 (Mar. 3, 2016), pp. 123–141. ISSN: 0268-0939, 1464-5106. DOI: [10.1080/02680939.2015.1035758](https://doi.org/10.1080/02680939.2015.1035758) (cit. on p. 16).
- [69] Florian Wirtz. "Enabling User-Generated Content in Cloud-Based E-Learning Environments in Schools". Potsdam, Germany: Hasso Plattner Institute, University of Potsdam, July 2, 2018 (cit. on pp. 52, 53).
- [70] *xAPI Specification*. Version 1.0.3. Advanced Distributed Learning (ADL). Sept. 23, 2016. URL: <https://github.com/adlnet/xAPI-Spec/tree/1.0.3> (visited on 02/15/2019) (cit. on pp. XVI, 66).

DECLARATION

I certify that the material contained in this thesis is my own work and does not contain unreferenced or unacknowledged material.

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel verwendet habe.

Potsdam, 6th May 2019

Sebastian Serth